

CSRI SUMMER PROCEEDINGS 2010

The Computer Science Research Institute
at Sandia National Laboratories

Editors:

Eric C. Cyr and S. Scott Collis
Sandia National Laboratories

December 17, 2010



SAND2010-8783P

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energys National Nuclear Security Administration under contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: <http://www.doe.gov/bridge>

Available to the public from

U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: <http://www.ntis.gov/ordering.htm>



Preface

The Computer Science Research Institute (CSRI) brings university faculty and students to Sandia National Laboratories for focused collaborative research on computer science, computational science, and mathematics problems that are critical to the mission of the laboratories, the Department of Energy, and the United States. CSRI provides a mechanism by which university researchers learn about and impact national- and global-scale problems while simultaneously bringing new ideas from the academic research community to bear on these important problems.

A key component of CSRI programs over the last decade has been an active and productive summer program where students from around the country conduct internships at CSRI. Each student is paired with a Sandia staff member who serves as technical advisor and mentor. The goals of the summer program are to expose the students to research in mathematical and computer sciences at Sandia and to conduct a meaningful and impactful summer research project with their Sandia mentor. Every effort is made to align summer projects with the student's research objectives and all work is coordinated with the ongoing research activities of the Sandia mentor in alignment with Sandia technical thrusts and the needs of the NNSA Advanced Scientific Computing (ASC) program that has funded CSRI from its onset.

Starting in 2006, CSRI has encouraged all summer participants and their mentors to contribute a technical article to the CSRI Summer Proceedings, of which this document is the fifth installment. In many cases, the CSRI proceedings are the first opportunity that students have to write a research article. Not only do these proceedings serve to document the research conducted at CSRI but, as part of the research training goals of CSRI, it is the intent that these articles serve as precursors to or first drafts of articles that could be submitted to peer-reviewed journals. As such, each article has been reviewed by a Sandia staff member knowledgeable in that technical area with feedback provided to the authors. Several articles have or are in the process of being submitted to peer-reviewed conferences or journals and we anticipate that additional submissions will be forthcoming.

For the 2010 CSRI Proceedings, research articles have been organized into the following broad technical focus areas — *computational mathematics and algorithms, uncertainty quantification and sensitivity analysis, meshing and optimization, computational applications, architectures and networking, and visualization and software engineering* — which are well aligned with Sandia's strategic thrusts in computer and information sciences.

We would like to thank all participants who have contributed to the outstanding technical accomplishments of CSRI in 2010 as documented by the high quality articles in this proceedings. The success of CSRI hinged on the hard work of 36 enthusiastic student collaborators and their dedicated Sandia technical staff mentors. It is truly impressive that the research described herein occurred primarily over a three month period of intensive collaboration.

CSRI benefited from the administrative help of Dee Cadena, Deanna Ceballos, Denis Laporte, Mel Loran, and Bernadette Watts. The success of CSRI is, in large part, due to their dedication and care, which are much appreciated. We would also like to thank those who reviewed articles for this proceedings — their feedback is an important part of the research training process and has significantly improved the quality of the papers herein. Finally, we want to acknowledge the ASC program for their continued support of the CSRI and its activities which have benefited both Sandia and the greater research community.

Eric C. Cyr
S. Scott Collis

December 17, 2010

Table of Contents

Preface

<i>E.C. Cyr and S.S. Collis</i>	iii
---	-----

Computational Mathematics and Algorithms

<i>E.C. Cyr and S.S. Collis</i>	1
The Nonlocal Cattaneo-Vernotte Equation	
<i>N.J. Burch and R.B. Lehoucq</i>	3
Discontinuous Velocity Least Squares Finite Element Methods for Improved Mass Conservation	
<i>J. Lai, P. Bochev, L. Olson, K. Peterson, D. Ridzal, and C. Siefert</i>	15
Application of a Discontinuous Petrov-Galerkin Method to the Stokes Equations	
<i>N.V. Roberts, D. Ridzal, P.B. Bochev, L.D. Demkowicz, K.J. Peterson, and C.M. Siefert</i>	32
An Investigation of Block preconditioners for Unsteady Navier-Stokes	
<i>E.G. Phillips, E.C. Cyr, and J.N. Shadid</i>	47
Efficiently Computing Tensor Eigenvalues on a GPU	
<i>G. Ballard, T. Kolda, and T. Plantenga</i>	59

Uncertainty Quantification and Sensitivity Analysis

<i>E.C. Cyr and S.S. Collis</i>	77
Stochastic Galerkin FEM	
<i>R. Tiptredy, E.T. Phipps and R.G. Ghanem</i>	79
Uncertainty Quantification of a Radiation Damaged Bipolar Junction Transistor	
<i>C.W. Miller, R.S. Tuminaro, E.T. Phipps, and H.C. Elman</i>	91
Krylov Recycling for Climate Modeling and Uncertainty Quantification.	
<i>K. Ahuja, M.L. Parks, E.T. Phipps, A.G. Salinger, and E. de Sturler</i>	103
Stability of ODEs with Colored Noise Forcing	
<i>T.J. Blass and L.A. Romero</i>	112
Comparison of Sensitivity Analysis Methods for Nuclear Reactor Neutronics	
<i>W. C. Proctor, B. M. Adams, C. Rabiti and H. S. Abdel-Khalik</i>	124

Meshing and Optimization

<i>E.C. Cyr and S.S. Collis</i>	139
Quadratic Element Mesh Untangling and Shape Optimization via the Target-Matrix Paradigm	
<i>N. Voshell, P. Knupp, and J. Kraftcheck</i>	141
A New Strategy for Untangling 2D Meshes	
<i>J.W. Franks and P. Knupp</i>	152
Mesh Vertex Reordering for Local Mesh Quality Improvement	
<i>J.Park and P.Knupp</i>	166
Multifractal Dimensions using Maximal Simplices and Python Extensions to TEVA-SPOT	
<i>J. Berwald, D. Day, S. Mitchell and A. Zomorodian</i>	178
Benders Decomposition in Pyomo	
<i>P. Steele and J. Watson</i>	196
Stochastic Optimization of an Energy Model	
<i>K. Hunter, J. DeCarolis, S. Sreepathi, J. Watson</i>	206
An ACRO Implementation of the Hybrid Optimization Algorithm EAGLS	
<i>J.W. Orsini and G.A. Gray</i>	214
Computational Applications	
<i>E.C. Cyr and S.S. Collis</i>	219

An Electron Force Field Study of Shocked Polyethylene	
<i>P.L. Theofanis, T.R. Mattsson and A.P. Thompson</i>	221
Flux-Corrected Transport Algorithm for the Remapping step of a FEM-ALE method	
<i>A. Lopez Ortega and G. Scovazzi</i>	233
Molecular Dynamics Study of Single Conjugated Polymer Nanoparticle	
<i>S. Maskey, F. Pierce, D. Perahia, S.J. Plimpton and G.S. Grest</i>	250
Charge Traps and Local Atomic Relaxations in Amorphous Silicon Dioxide	
<i>N.L. Anderson, P.A. Schultz, and A. Strachan</i>	258
Development of algorithm for nanoparticle collision simulation using Molecular Dynamics method	
<i>Y. Takato, J.B. Lechman, and S. Sen</i>	272
Numerical Simulation of the Performance of a Resonant Tunneling Diode	
<i>A.S. Costolanski and A.G. Salinger</i>	284
Architecture and Networking	
<i>E.C. Cyr and S.S. Collis</i>	295
A Lightweight, GPU-Based Software RAID System	
<i>M.L. Curry, H.L. Ward, A. Skjellum, and R.B. Brightwell</i>	297
Super-Scale Real-Time Network Simulation on the Cray XT	
<i>Nathanael Van Vorst, Kevin Pedretti, Ron Oldfield</i>	309
Application Support for Resilience in Exascale Systems	
<i>M.R. Varela and K.B. Ferreira and R. Riesen</i>	320
Statistical Analysis of HPC Alerts and Developments in Root Cause Analysis	
<i>J.M. Vaughan, J.R. Stearley, S.A. Mitchell, and G. Michailidis</i>	331
Techniques for Managing Data Distribution in NUMA Systems	
<i>A.M. Merritt and K.T. Pedretti</i>	343
Integrating Router Power Models into the Structural Simulation Toolkit	
<i>K.D. Thompson, A.F. Rodrigues, and M. Hsieh</i>	354
Process Layers for Discrete Event Simulation of Computer Systems	
<i>C.D. Kersey and A.F. Rodrigues</i>	361
Reliability Simulation for Structural Simulation Toolkit	
<i>A.S. Williams and A.F. Rodrigues</i>	371
Visualization and Software Engineering	
<i>E.C. Cyr and S.S. Collis</i>	377
Information Visualization using VisTrails Technology	
<i>W.B. Silva and J.F. Shepherd</i>	379
Comparison of Open Source Visual Analytics Toolkits	
<i>J.R. Harger and P.J. Crossno</i>	389
Optika: A GUI Framework for Parameterized Applications	
<i>K. Nusbaum and M. A. Heroux</i>	401
Epetra/AztecOO and related to Tpetra/Stratimikos and related: A Conversion Guide	
<i>K.F. Fermoye and M.A. Heroux</i>	410
Testing Engineering Software: Development and Testing of the Cubit Program	
<i>B.J. Hardy and B.W. Clark</i>	422

Computational Mathematics and Algorithms

Articles in this section focus on development of numerical algorithms and novel computational models. This includes discretization techniques, preconditioning methodologies, and implementation of numerical algorithms on novel architectures.

Burch and Lehoucq discuss the nonlocal Cattaneo-Vernotte equation for anomalous diffusion, introducing nonlocal boundary conditions. The effects of relaxation and nonlocality are studied with a finite element formulation. *Lai et al.* present a new least squares finite element formulation for the Stokes equations that has improved mass conservation. To achieve this, the formulation utilizes a novel discontinuous approximation of the velocity. Results demonstrating the usefulness of this approach are given. *Roberts et al.* develop a discontinuous Petrov-Galerkin method for the Stokes equations. Their study and the numerical experiments exposed several interesting avenues for future study. *Phillips et al.* investigate block preconditioners for the unsteady Navier-Stokes equations. The authors perform numerical experiments comparing several techniques for approximate the pressure Schur complement. *Ballard et al.* present an efficient implementation of the shifted symmetric higher order power method for computing tensor eigenvalues. They show that a 70x speedup over a serial implementation can be achieved when implemented on a GPU.

E.C. Cyr
S.S. Collis

December 17, 2010

THE NONLOCAL CATTANEO-VERNOTTE EQUATION

NATHANIAL J. BURCH* AND RICHARD B. LEHOUCQ†

Abstract. We introduce the nonlocal Cattaneo-Vernotte equation by including a relaxation effect in the nonlocal diffusion equation, both of which are models for anomalous diffusion. This equation has two different interpretations: as a generalization of Fick's first law in terms of a nonlocal flux and memory kernel and as an equation arising from the generalized master equation for a continuous time random walk. Both interpretations are discussed and the latter describes a scaling of relaxation time and nonlocality. A relationship to fractional diffusion equations in the limit of vanishing relaxation time and nonlocality is also established. The main contribution of this paper is to introduce nonlocal boundary conditions for the nonlocal Cattaneo-Vernotte equation, and the ensuing variational and finite element formulations. Examples are given where the effects of relaxation time and nonlocality are studied.

1. Introduction. The classical Cattaneo-Vernotte equation

$$w_t + \frac{\tau}{2} w_{tt} = a w_{xx}, \quad (1.1)$$

where $\tau/2 > 0$ is the relaxation time and $a > 0$ is the diffusion coefficient, is a model for diffusion that admits finite speeds of propagation, specifically $\sqrt{2a/\tau}$. When w is a temperature field, (1.1) is a model of hyperbolic heat conduction [14]. Further, (1.1) arises from the classical balance law, $w_t(x, t) = -q_x(x, t)$, and a generalization of Fick's first law in which the flux is given by a convolution of the gradient of the field w and a relaxation kernel [12],

$$q(x, t) = -a \int_0^t \frac{2}{\tau} \exp\left(-\frac{t-t'}{\tau/2}\right) w_x(x, t') dt'. \quad (1.2)$$

The assumption (1.2) also takes the more familiar form of Cattaneo's equation [7],

$$q + \frac{\tau}{2} q_t = -a w_x. \quad (1.3)$$

The classical diffusion equation

$$w_t = a w_{xx} \quad (1.4)$$

yields an infinite speed of propagation because its fundamental solution, i.e., the solution to (1.4) with an initial condition given by the Dirac delta function, is

$$w(x, t) = \frac{1}{\sqrt{4\pi at}} \exp\left(-\frac{x^2}{4at}\right),$$

which is positive for all x , for any arbitrarily small t . This property has been referred to in the literature as “unphysical” since disturbances are instantaneously propagated. Moreover, (1.4) is incapable of capturing transient dynamics of the field in situations involving short times, high frequencies, and short wave lengths [14]. One approach to remedy these issues is to introduce a relaxation time [12] and a special case of this is the classical Cattaneo-Vernotte equation (1.1), which overcomes the unphysical properties associated with infinite speeds of propagation present in (1.4).

The diffusion equation (1.4) may be derived by combining the classical balance law $w_t = -q_x$ and Fick's first law

$$q = -a w_x. \quad (1.5)$$

*Colorado State University, burch@math.colostate.edu

†Sandia National Laboratories, rblehou@sandia.gov

When the diffusion process is anomalous, e.g., does not obey Fick's first law (1.5), other models have been proposed. Examples include the fractional diffusion equation

$$v_t = -c(-\Delta)^{\alpha/2} v, \quad 0 < \alpha \leq 2, \quad (1.6)$$

which includes (1.4) as the special case $\alpha = 2$ and $c = a$, and the nonlocal diffusion equation

$$u_t(x, t) = \frac{1}{\lambda} \int_{\mathbb{R}} (u(y, t) - u(x, t)) \phi(x - y) dy. \quad (1.7)$$

The equation (1.7) has a probabilistic interpretation as a generalized master equation for a continuous time random walk (CTRW). The rate of diffusion associated with $u(x, t)$ depends upon points $y \neq x$, e.g., the rate of diffusion is the difference in the rate at which u enters x at time t , $\lambda^{-1} \int_{\mathbb{R}} u(y, t) \phi(y - x) dy$, and the rate at which u departs x at time t , $\lambda^{-1} u(x, t)$. The mean wait-time between steps is λ and, given the radial probability density ϕ , the diffusion coefficient is given by

$$\frac{1}{2\lambda} \int_{\mathbb{R}} s^2 \phi(s) ds.$$

Like (1.4), both (1.6) and (1.7) give rise to infinite speeds of propagation. We note (1.7) has been used as a model for peridynamic heat conduction [4] and variations of it have appeared in numerous applications [2, 6, 10].

This paper focuses on the nonlocal Cattaneo-Vernotte equation

$$u_t(x, t) + \frac{\tau}{2} u_{tt}(x, t) = \frac{1}{\beta} \int_{\mathbb{R}} (u(y, t) - u(x, t)) \phi(x - y) dy, \quad (1.8)$$

where ϕ is a radial probability density function, β is the mean wait-time between steps, and $\tau/2 > 0$ is again the relaxation time. In (1.8), the diffusion coefficient is

$$\frac{1}{2\beta} \int_{\mathbb{R}} s^2 \phi(s) ds.$$

In the spirit of (1.7), (1.8) has replaced the second-order spatial derivative in (1.1) with the nonlocal integral operator and, consequently, is a model for anomalous diffusion. With the introduction of nonlocal boundary conditions, following [5], (1.8) becomes a model for nonlocal hyperbolic heat conduction on bounded domains. The nonlocal Cattaneo-Vernotte equation (1.8), like (1.7) has a probabilistic interpretation. Thus, (1.8) is a model for anomalous diffusion that can be derived from a CTRW framework.

The contribution of this paper is to investigate the effect of a nonzero relaxation time by comparing solutions of nonlocal boundary value problems corresponding to (1.7) and (1.8). The former was studied extensively in [5] and a conforming finite element method, which depends on the variational framework presented in [11], to approximate solutions to these nonlocal boundary value problems is used. We demonstrate a relationship between relaxation time and nonlocality and the solutions of (1.8) converge to those of (1.6), as both relaxation time and nonlocality vanish. Consequently, a relationship between (1.6)–(1.8) in the limit of vanishing relaxation time and nonlocality is established.

The rest of this paper is organized as follows. Section 2 demonstrates how (1.8) arises from a generalization of Fick's first law in which the flux is given by a convolution of a memory kernel and a nonlocal spatial operator acting as the gradient of the field, in contrast to (1.2). The relationships between (1.6)–(1.8) are also reviewed. Section 3 relates the nonlocal Cattaneo-Vernotte equation to a continuous time random walk via the generalized master equation. Nonlocal boundary conditions for (1.8) are reviewed in Section 4, as is the variational formulation and ensuing finite element method. Section 5 provides numerical examples to illustrate the effects of nonzero relaxation time and nonlocality.

2. Generalization of Fick's first law and a nonlocal flux. In this section, we demonstrate (1.8) arises via the classical balance law, $u_t = -\varrho_x$, and a generalization of Fick's first law,

$$\varrho(x, t) = \int_0^t \frac{2}{\tau} \exp\left(-\frac{t-t'}{\tau/2}\right) \left(\frac{1}{\beta} p(x, t')\right) dt', \quad (2.1)$$

where

$$p(x, t) := -\frac{1}{2} \int_{\mathbb{R}} \int_0^1 (u(x + (1-\lambda)z, t) - u(x - \lambda z, t)) z \phi(z) d\lambda dz.$$

Differentiating (2.1) with respect to t and rearranging reveals

$$\varrho + \frac{\tau}{2} \varrho_t = \frac{1}{\beta} p, \quad (2.2)$$

and so

$$u_t = -\frac{1}{\beta} p_x + \frac{\tau}{2} \varrho_{xt} = -\frac{1}{\beta} p_x - \frac{\tau}{2} u_{tt} \quad (2.3)$$

Noll's Lemma I [15, 18] implies that

$$-\frac{1}{\beta} p_x(x, t') = \frac{1}{\beta} \int_{\mathbb{R}} (u(x+z, t') - u(x, t')) \phi(z) dz, \quad (2.4)$$

and so (1.8) is established.

We now demonstrate a formal relationship between (1.8) and (1.1) in the presence of vanishing nonlocality. Fix $\tau > 0$, let $\beta = \varepsilon^2$, where $\varepsilon > 0$, and define the symmetric probability density

$$\phi_\varepsilon(s) := \frac{1}{\varepsilon} \phi(s/\varepsilon), \quad (2.5)$$

where the given symmetric density ϕ satisfies¹

$$\int_{\mathbb{R}} s^{2k} \phi(s) ds < \infty, \quad k = 0, 1, 2, \dots$$

As $\varepsilon \rightarrow 0$, $\phi_\varepsilon(x-y)$ weights points nearby x more heavily, relative to points further away. Necessarily, by specifying the second moment appropriately, the Fourier transform of ϕ has an expansion of the form

$$\widehat{\phi}(\xi) = 1 - a|\xi|^2 + o(|\xi|^2), \quad a > 0.$$

With ϕ_ε in place of ϕ and assuming a formal Taylor expansion is valid for sufficiently small ε ,

$$\begin{aligned} \frac{1}{\beta} p(x, t) &= -\frac{1}{2} \int_{\mathbb{R}} \int_0^1 \frac{1}{\varepsilon^2} \left(u_x(x, t) z^2 + \sum_{k=2}^{\infty} \frac{1}{k!} z^{k+1} \frac{\partial^k u(x, t)}{\partial x^k} \right) \phi_\varepsilon(z) d\lambda dz \\ &= -a u_{xx}(x, t) + \frac{1}{\varepsilon^2} \sum_{k=2}^{\infty} \frac{\partial^{2k-1} u(x, t)}{\partial x^{2k-1}} \frac{1}{(2k-1)!} \int_{\mathbb{R}} z^{2k} \phi_\varepsilon(z) dz \\ &= -a u_{xx}(x, t) + \sum_{k=2}^{\infty} \frac{\partial^{2k-1} u(x, t)}{\partial x^{2k-1}} \frac{\varepsilon^{2(k-1)}}{(2k-1)!} \int_{\mathbb{R}} z^{2k} \phi(z) dz \end{aligned}$$

¹The assumption of symmetry of ϕ implies that the odd moments are zero.

and, utilizing (2.2), we obtain an approximation of (1.3),

$$\varrho + \frac{\tau}{2}\varrho_t = -au_x + O(\varepsilon^2). \quad (2.6)$$

Thus, in the absence of nonlocality, the nonlocal Cattaneo-Vernotte equation (1.8) reduces to the classical Cattaneo-Vernotte equation (1.1). The effect of the density ϕ_ε with $\beta = \varepsilon^2$ as ε decreases is to “localize” the diffusion of (1.8). Indeed, taking $\phi(x-y) = \delta(x-y) + a\delta''(x-y)$ in (1.8) recovers (1.1). Moreover, if $\tau = O(\varepsilon^2)$, (2.6) reduces to

$$\varrho = -au_x + O(\varepsilon^2), \quad (2.7)$$

approximating (1.5). Thus, in the absence of both relaxation time and nonlocality, the nonlocal Cattaneo-Vernotte equation (1.8) reduces to the classical diffusion equation (1.4).

Finally, we establish a relationship to the fractional diffusion equation (1.6). Suppose ϕ is a symmetric probability density function with the expansion

$$\widehat{\phi}(\xi) = 1 - c|\xi|^\alpha + o(|\xi|^\alpha), \quad 0 < \alpha \leq 2, \quad (2.8)$$

for $c > 0$, so that, defining ϕ_ε via (2.5),

$$\widehat{\phi}_\varepsilon(\xi) = 1 - c\varepsilon^\alpha|\xi|^\alpha + o(\varepsilon^\alpha|\xi|^\alpha).$$

Assuming $\beta = \varepsilon^\alpha$, the Fourier transform of (1.8) gives

$$\begin{aligned} \widehat{u}_t(\xi, t) + \frac{\tau}{2}\widehat{u}_{tt}(\xi, t) &= \frac{1}{\varepsilon^\alpha}(\widehat{\phi}_\varepsilon(\xi) - 1)\widehat{u}(\xi, t) \\ &= \frac{1}{\varepsilon^\alpha}(-c\varepsilon^\alpha|\xi|^\alpha + o(\varepsilon^\alpha|\xi|^\alpha))\widehat{u}(\xi, t) \\ &= -c|\xi|^\alpha\widehat{u}(\xi, t) + O(\varepsilon^\alpha|\xi|^\alpha), \end{aligned}$$

implying that u , in a formal sense, is approximately given by the fractional Cattaneo-Vernotte equation

$$v_t(x, t) + \frac{\tau}{2}v_{tt}(x, t) = -c(-\Delta)^{\alpha/2}v(x, t). \quad (2.9)$$

Further, if $\tau = O(\varepsilon^\alpha)$, u is approximately given by the fractional diffusion equation (1.6), where equality can be shown in special cases of this limit via characteristic function techniques. Evidently, the nonlocal boundary value problems corresponding to (1.8) are then related to those of the fractional Cattaneo-Vernotte equation (2.9) and fractional diffusion equation (1.6) in this limit.

3. The Nonlocal Cattaneo-Vernotte Equation and CTRWs. Another perspective of the nonlocal Cattaneo-Vernotte equation (1.8) comes from CTRWs. We consider a separable² continuous time random walk with a wait-time density ω and a radial step-length density ϕ . One form of the generalized master equation, which is an equation for the time evolution of the joint probability density function for the state of the CTRW, is

$$u_t(x, t) = \int_0^t \Lambda(t-t') \int_{\mathbb{R}} (u(y, t') - u(x, t'))\phi(x-y) dy dt', \quad (3.1)$$

²The assumption of separable simply states that wait-times and step-lengths are independent.

where the Laplace transform of the memory kernel Λ is determined by that of ω via

$$\widehat{\Lambda}(s) = \frac{s\widehat{\omega}(s)}{1 - \widehat{\omega}(s)}. \quad (3.2)$$

We refer the reader to [3, 16] for a derivation and thorough discussion of (3.1). The memory kernel Λ captures potential memory effects due to the wait-times. In fact, only when $\Lambda(t) \propto \delta(t)$ is the underlying CTRW Markovian, i.e., wait-times are exponentially distributed and are therefore memoryless.

Literature demonstrating the use of (3.1) in the modeling of diffusion processes is plentiful. For instance, taking $\Lambda(t) = \frac{1}{\lambda}\delta(t)$ gives rise to the nonlocal diffusion equation (1.7). If ϕ is a weighted average of Dirac measures on an integer lattice, then (3.1) describes the probability density of the CTRW on that lattice; see [13]. Moreover, see [16, 17], subdiffusive processes have been studied by taking $\widehat{\Lambda}(s) = s^{1-\mu}$, $\mu \in (0, 1)$. The following lemma provides conditions for (3.1) to yield (1.8).

LEMMA 3.1. *The nonlocal Cattaneo-Vernotte equation (1.8) is obtained from (3.1) by taking*

$$\Lambda(t - t') = \frac{1}{\beta} \frac{2}{\tau} \exp\left(-\frac{t - t'}{\tau/2}\right) \quad (3.3)$$

and imposing the restriction $\beta \geq 2\tau$. The assumption (3.3) is tantamount to

$$\omega(t) = \begin{cases} \frac{t}{\tau^2} \exp\left(-\frac{1}{\tau}t\right), & \beta = 2\tau, \\ \frac{2}{\sqrt{\beta(\beta - 2\tau)}} \exp\left(-\frac{t}{\tau}\right) \sinh\left(\frac{\sqrt{\beta(\beta - 2\tau)}}{\beta\tau}t\right), & \beta > 2\tau. \end{cases} \quad (3.4)$$

Proof. First, (3.4) is implied from (3.2) and (3.3) via Laplace transform techniques and, in the process, we find $\beta \geq 2\tau$ if and only if $\omega(t) \geq 0$ for all $t > 0$, which is necessary for ω to be a probability density. Insertion of (3.3) into (3.1) and differentiating with respect to t yields

$$u_{tt}(x, t) = -\frac{2}{\tau}u_t(x, t) + \frac{1}{\beta} \frac{2}{\tau} \int_{\mathbb{R}} (u(y, t) - u(x, t))\phi(x - y) dy$$

and thus, upon rearranging, (1.8). \square

The special case $\beta = 2\tau$ implies $W \sim \text{Gamma}(2, \tau)$, where W is the wait-time random variable. For the duration of the paper, we restrict³ ourselves to $\beta \geq 2\tau$ so that we have an interpretation from a CTRW perspective. This restriction has appealing consequences as well, e.g., positivity of solutions and conservation of mass. We note

$$\widehat{\omega}(s) = \frac{2}{\beta\tau s^2 + 2\beta s + 2} = \sum_{k=0}^{\infty} \frac{(-1)^k}{k!} \mathbb{E}(W^k) s^k = 1 - \beta s + o(s),$$

which shows that the mean wait-time is indeed β . Recalling also (2.8), we compute the so-called pseudo mean square displacement [17] of a random walker,

$$\int_{-L_1 t^{1/\alpha}}^{L_2 t^{1/\alpha}} x^2 u(x, t) dx \sim t^{2/\alpha},$$

³This restriction is necessary for a CTRW interpretation, but might be relaxed in other contexts.

which is the mean square displacement on a bounded interval that grows in size with t . When $\alpha \in (0, 2)$, $2/\alpha > 1$ and (1.8) is thus a model for anomalous superdiffusion [17]. One considers a pseudo mean square displacement in this situation, since the true mean square displacement is infinite for any $\alpha \in (0, 2)$. In the special case when $\alpha = 2$, the diffusion is not anomalous.

4. The Nonlocal Cattaneo-Vernotte Equation on Bounded Domains. The results in [11] provide a variational formulation for nonlocal boundary value problems for (1.8). This follows closely to that presented for the nonlocal diffusion equation (1.7) in [5]. Before giving these variational formulations and describing the ensuing finite element method, we establish some notation.

We consider the bounded domain $\Omega = (0, 1)$. Define the bilinear form

$$B_I(u, v) := \frac{1}{2} \int_I \int_I (u(y, t) - u(x, t))(v(y) - v(x)) \phi_\varepsilon(x - y) dy dx, \quad (4.1)$$

where $I \in \{\mathbb{R}, (0, 1)\}$. Let $V_I = L^2(I)$, where

$$L^2(I) := \left\{ v \mid \int_I |v|^2 dx < \infty \right\},$$

and \bar{V}_I denote possible choices for the subspaces of test and trial functions, with

$$\bar{V}_{\mathbb{R}} := \left\{ v \in V_{\mathbb{R}} \mid v|_{\mathbb{R} \setminus (0,1)} = 0 \right\} \text{ and } \bar{V}_{(0,1)} := \left\{ v \in V_{(0,1)} \mid \int_0^1 v dx = \int_0^1 u_0 dx \right\},$$

where $u(x, 0) = u_0(x)$ is a given initial density.

The nonlocal homogeneous Dirichlet ($I = \mathbb{R}$) and Neumann ($I = (0, 1)$) boundary value problems for (1.8) are presented together: Find $u \in \bar{V}_I \times (0, \infty)$ such that

$$\begin{cases} u_t(x, t) + \frac{\tau}{2} u_{tt}(x, t) = \frac{1}{\beta} \int_I (u(y, t) - u(x, t)) \phi_\varepsilon(x - y) dy, & x \in (0, 1), \\ u(x, 0) = u_0(x), & x \in (0, 1), \\ u_t(x, 0) = 0, & x \in (0, 1). \end{cases} \quad (4.2)$$

We present a useful result from [9].

THEOREM 4.1 (Emmrich and Weckner (2006)). *Suppose*

$$\kappa_0 := \text{esssup}_{x \in I} |K_0(x)| < \infty \quad \text{and} \quad \kappa := \int_0^1 \int_0^1 |K(x, y)|^2 dy dx < \infty.$$

For a given $u_0 \in \bar{V}_I$, there is a unique mild solution $u \in C^2([0, T]; \bar{V}_I)$ to

$$\frac{2}{\tau} u_t(x, t) + u_{tt}(x, t) = \int_0^1 K(x, y) u(y, t) dy - K_0(x) u(x, t).$$

We remark that existence and uniqueness of solutions to (4.2) follows from Theorem 4.1 with

$$K(x, y) := \frac{2}{\tau\beta} \phi(x - y) \quad \text{and} \quad K_0(x) = \int_I \frac{2}{\tau\beta} \phi(x - y) dy.$$

The variational formulations to (4.2) are: Find $u \in \bar{V}_I \times (0, \infty)$ such that

$$\begin{cases} \int_0^1 u_t v dx + \frac{\tau}{2} \int_0^1 u_{tt} v dx + \frac{1}{\beta} B_I(u, v) = 0, & \forall v \in V_I, \\ u(x, 0) = u_0(x), & x \in (0, 1), \\ u_t(x, 0) = 0, & x \in (0, 1). \end{cases} \quad (4.3)$$

We refer the reader to [5, 11] for more details concerning the variational formulations.

The nonlocal Dirichlet boundary condition constrains the field u on $\mathbb{R} \setminus (0, 1)$, which is analogous to the classical Dirichlet boundary condition that does so on $\{0, 1\}$. For the nonlocal Neumann boundary condition, the integral in (4.2) is only over $(0, 1)$ rather than all of \mathbb{R} . This constrains diffusion to occur only inside $(0, 1)$, i.e., density neither enters nor exits $(0, 1)$, which is analogous to the classical Neumann boundary condition. Further, since $B_{(0,1)}(u, 1) = 0$, the compatibility condition necessary for the Neumann problem to possess a solution is

$$\int_0^1 u(x, t) dx := \bar{u}_0, \quad \forall t \geq 0, \quad (4.4)$$

which is a statement that the integrated quantity u is conserved for all time.

THEOREM 4.2. *Let $u \in C^2([0, T]; \bar{V}_I)$ be the unique solution to (4.2). Then, $u_t(x, t) \rightarrow 0$, as $t \rightarrow \infty$, for almost every $x \in (0, 1)$.*

Proof. Multiply (4.2) by $u_t(x, t)$, integrate over $x \in (0, 1)$, and then integrate in t to obtain

$$\frac{\tau}{4} \int_I u_t^2(x, t) dx = \frac{1}{2\beta} (B_I(u_0, u_0) - B_I(u, u)) - \int_0^t \int_I u_t^2(x, s) dx ds$$

and thus

$$B_I(u_0, u_0) \geq B_I(u, u) + 2\beta \int_0^t \int_I u_t^2(x, s) dx ds \geq 2\beta \int_0^t \int_I u_t^2(x, s) dx ds.$$

Since $B_I(u_0, u_0) < \infty$, $u_t(x, t) \in L^2(I)$ for all t and

$$\|u_t(x, t)\|_{L^2(I)}^2 = \int_I u_t^2(x, t) dx \rightarrow 0.$$

The completeness of $L^2(I)$ implies that $u_t \rightarrow g$ with $\|g\|_{L^2(I)} = 0$, i.e., $g = 0$ almost everywhere and, thus, $u_t \rightarrow 0$ for almost every $x \in (0, 1)$. \square

A stationary solution to (4.2), $u_s \in \bar{V}_I$, solves

$$\int_I (u_s(y) - u_s(x)) \phi(x - y) dy = 0, \quad \forall x \in (0, 1).$$

The results in [8, 11] demonstrate that the unique stationary solution of the homogeneous Dirichlet problem is $u_s = 0$ and that of the homogeneous Neumann problem is $u_s = \bar{u}_0$. Consequently, a simple corollary to Theorem 4.2 is $u(x, t) \rightarrow u_s(x)$ as $t \rightarrow \infty$ for almost every $x \in (0, 1)$.

4.1. A Semi-discrete Finite Element Formulation. To formulate the finite element method, we partition $(0, 1)$ into n subintervals Ω_i and let $\chi_i(x)$ be the indicator function for Ω_i . We denote the space of piecewise constant functions on the subintervals Ω_i by $V_{(0,1)}^h$. Note any $u_h \in V_{(0,1)}^h \times (0, \infty)$ can be written

$$u_h(x, t) = \sum_{j=1}^n \gamma_j(t) \chi_j(x).$$

The discrete variational problem is then: Find $u_h \in V_{(0,1)}^h \times (0, \infty)$ such that

$$\mathbf{M}\dot{\gamma} + \frac{\tau}{2} \mathbf{M}\ddot{\gamma} = -\mathbf{A}\gamma,$$

where \mathbf{M} and \mathbf{A} are the mass and stiffness matrices defined by

$$M_{ii} = |\Omega_i| \quad \text{and} \quad A_{ij} = \frac{1}{\beta} \begin{cases} - \int_{\Omega_i} \int_{\Omega_j} \phi_\varepsilon(x-y) \, dy \, dx, & i \neq j, \\ \int_{\Omega_i} \int_{I \setminus \Omega_i} \phi_\varepsilon(x-y) \, dy \, dx, & i = j. \end{cases}$$

For the Neumann problem, in light of (4.4), $u_h \in \bar{V}_{(0,1)}^h \times (0, \infty)$ is extracted by enforcing that

$$\sum_{j=1}^n \gamma_j(t) |\Omega_j| = \bar{u}_0.$$

5. Numerical Experiments and Examples. In this section, we present two examples to demonstrate various properties of numerical solutions of the nonlocal Cattaneo-Vernotte equation on bounded domains. In each example, ϕ_ε is defined in (2.5) and we use the scaling

$$\beta = 2\tau = c\varepsilon^\alpha, \quad (5.1)$$

where α and c are given in (2.8), so that we have both the probabilistic interpretation in Section 3 and a relationship to fractional diffusion established in Section 2.

The first example examines a nonlocal Cattaneo-Vernotte equation with homogeneous Neumann boundary conditions that admits an analytic solution for any initial condition. We demonstrate that solutions can be viewed as perturbations of solutions to the corresponding nonlocal diffusion equation (1.7) and we investigate the effects of a nonzero relaxation time. In Example 2, we consider the discontinuous initial condition

$$u_0(x) = \begin{cases} 0, & 0 < x < 0.5, \\ 1, & 0.5 \leq x < 1. \end{cases} \quad (5.2)$$

and investigate the effects of vanishing relaxation time and nonlocality, i.e., letting $\varepsilon \rightarrow 0$, on the solutions to a nonlocal Dirichlet boundary value problem. We use Lévy stable densities of various stability indices to illustrate the relationship to classical and fractional diffusion in this limit.

Example 1. Consider the nonlocal homogeneous Neumann Cattaneo-Vernotte equation

$$\begin{cases} u_t + \frac{\varepsilon^2}{24} u_{tt} = \frac{6}{\varepsilon^2} \int_0^1 (u(y, t) - u(x, t)) \phi_\varepsilon(x-y) \, dy, & x \in (0, 1), \\ u(x, 0) = u_0(x), & x \in (0, 1), \\ u_t(x, 0) = 0, & x \in (0, 1), \end{cases} \quad (5.3)$$

where

$$\phi_\varepsilon(s) = \frac{1}{2\varepsilon} \chi_{(-\varepsilon, \varepsilon)}(s), \quad \varepsilon \geq 1,$$

so that $\alpha = 2$, $c = 1/6$, and, consequently, $\beta = \varepsilon^2/6$ and $\tau = \varepsilon^2/12$. The goal of this example is to consider the case of increasing ε , e.g., increasing nonlocality.

In this example, since $\varepsilon \geq 1$ and $\text{supp}(\phi(x-y))$ contains $(0, 1)$ for all $x \in (0, 1)$, (5.3) reduces to an ordinary differential equation whose solution can be given as a convex combination of the initial condition $u_0(x)$ and the constant \bar{u}_0 ,

$$u_c(x, t) = \bar{u}_0(1 - \zeta_c(t)) + \zeta_c(t)u_0(x), \quad (5.4)$$

where

$$\zeta_c(t) = \exp\left(-\frac{12}{\varepsilon^2}t\right) \left(\sqrt{\frac{1}{1-\frac{1}{2\varepsilon}}} \sinh\left(\frac{12}{\varepsilon^2} \sqrt{1-\frac{1}{2\varepsilon}}t\right) + \cosh\left(\frac{12}{\varepsilon^2} \sqrt{1-\frac{1}{2\varepsilon}}t\right) \right).$$

The function $\zeta_c(t) \in (0, 1]$ is a strictly decreasing function that tends to zero as $t \rightarrow \infty$. If $u_0(x) = \bar{u}_0$ for some $x \in (0, 1)$, then x is a fixed point, i.e., $u(x, t) = u_0(x)$, for all $t \geq 0$. Also, the monotonicity of ζ_c implies $u(x, t) \nearrow \bar{u}_0$ if $u_0(x) < \bar{u}_0$ and, likewise, $u(x, t) \searrow \bar{u}_0$ if $u_0(x) > \bar{u}_0$ as $t \rightarrow \infty$. As $\varepsilon \rightarrow \infty$, $\zeta_c(t) \rightarrow 1$ for any fixed $t < \infty$. Thus, $u_c(x, t)$ can be well-approximated by $u_0(x)$ for arbitrary large finite time by choosing ε sufficiently large.

In [5], it was shown that the solution of (1.7) with homogeneous Neumann boundary conditions,

$$\begin{cases} u_t = \frac{6}{\varepsilon^2} \int_0^1 (u(y, t) - u(x, t)) \phi_\varepsilon(x - y) dy, & x \in (0, 1), \\ u(x, 0) = u_0(x), & x \in (0, 1), \end{cases} \quad (5.5)$$

for the same ϕ_ε as in (5.3), is also given by a convex combination of $u_0(x)$ and \bar{u}_0 ,

$$u_d(x, t) = \bar{u}_0(1 - \zeta_d(t)) + \zeta_d(t)u_0(x), \quad (5.6)$$

where

$$\zeta_d(t) = \exp\left(-\frac{3}{\varepsilon^3}t\right).$$

Thus, solutions of (5.3) can be given by

$$u_c(x, t) = u_d(x, t) + (\zeta_c(t) - \zeta_d(t))(u_0(x) - \bar{u}_0),$$

the sum of the solution to (5.5) and a perturbation $(\zeta_c(t) - \zeta_d(t))(u_0(x) - \bar{u}_0)$ due to a nonzero relaxation time. Since $u_0(x)$ and \bar{u}_0 are fixed for a given initial condition, we study the difference $u_c(x, t) - u_d(x, t)$ simply by investigating $\zeta_c(t) - \zeta_d(t)$.

In Fig. 5.1, we plot $\zeta_c(t) - \zeta_d(t)$ for $t \in [0, 3]$ and $\varepsilon \in [1, 3]$. As $t \rightarrow \infty$, $\zeta_c(t) - \zeta_d(t) \rightarrow 0$, but more slowly for increasing ε . This reflects agreement of stationary solutions for the two problems. For small values of t , $\zeta_c(t) > \zeta_d(t)$, which is an effect of the nonzero relaxation time. After this short time frame, $\zeta_c(t) - \zeta_d(t) = 0$, i.e., the solutions agree exactly at some point in time $t > 0$, and then $\zeta_c(t) < \zeta_d(t)$ for the duration of time. These observations hold for all ε , but are less dramatic as ε increases.

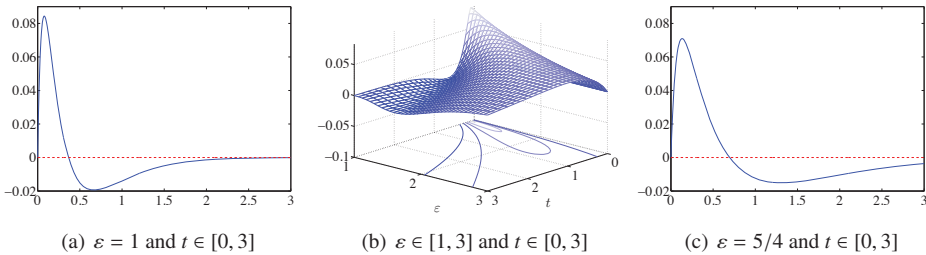


FIG. 5.1. The vertical axis is $\zeta_c(t) - \zeta_d(t)$ in all three panels. In panels (a) and (c) the horizontal axis is $t \in [0, 3]$.

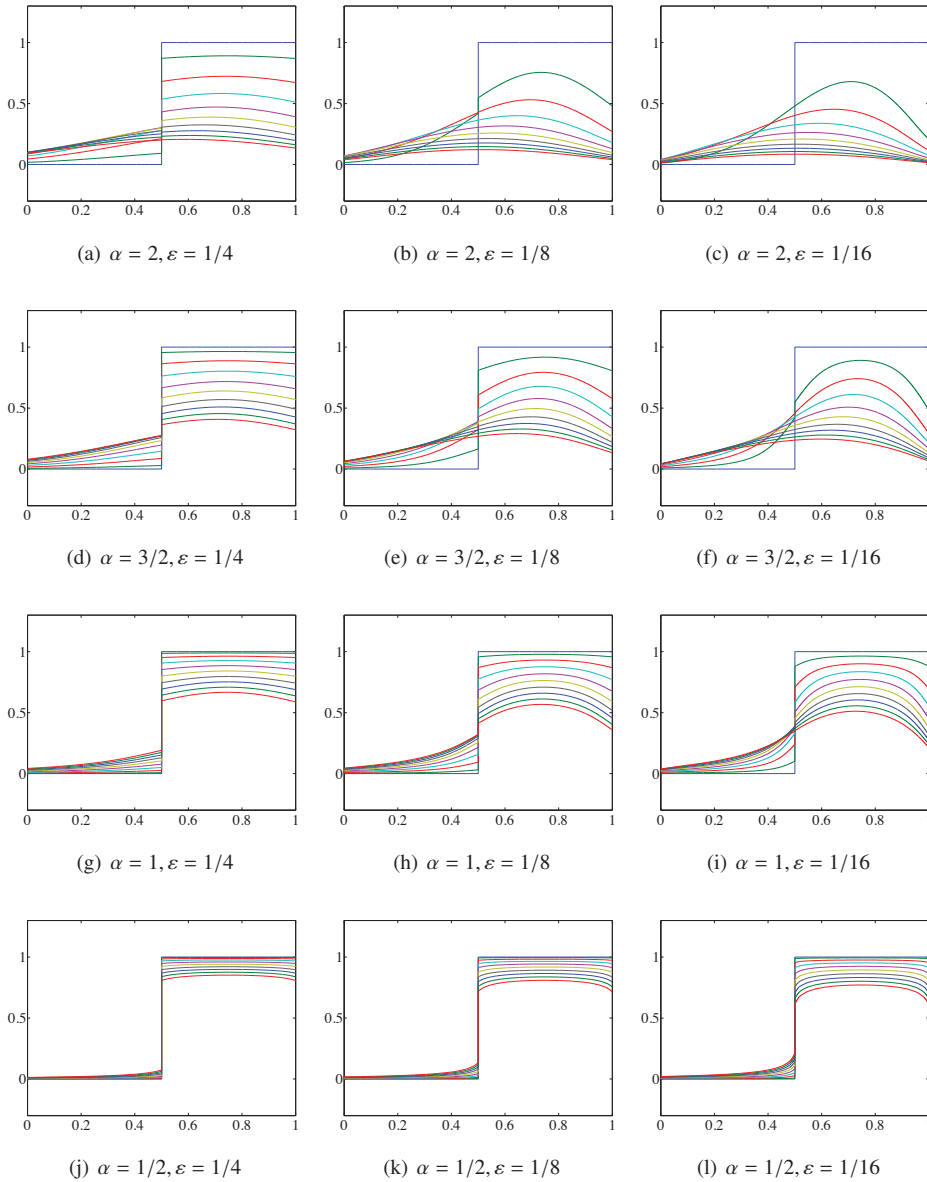


FIG. 5.2. Each panel shows solutions to the nonlocal homogeneous Dirichlet problem for different α and ε . The density ϕ_ε^α is used, where ϕ^α is a Lévy stable density with index of stability α . Since $c = 1$, we take $2\tau = \varepsilon^\alpha$. The vertical axis in each panel is the value of $u_h(x, t)$ and the horizontal axis is x . The ten different solution profiles in each panel correspond to the solutions at ten different times, $t \in [0, 0.25]$.

Example 2. The fractional diffusion behavior of boundary value problems for (1.8) is examined by choosing $\phi = \phi^\alpha$ to be a symmetric and centered Lévy stable density with stability index $\alpha \in \{2, 3/2, 1, 1/2\}$. As explained in Section 2, α represents the fraction of the Laplacian in the equations (1.6) and (2.9). Such Lévy stable densities, normalized so that $c = 1$, are characterized, via the Lévy-Khintchine representation, through their Fourier transforms,

i.e.,

$$\phi^\alpha(s) = \mathcal{F}^{-1}(\exp(-|\xi|^\alpha))(s), \quad (5.7)$$

see [1, §§ 1.2.5]. We use (2.5) to define ϕ_ε^α and the cases $\alpha = 2, 1$ yield closed-form expressions for ϕ_ε^α :

$$\phi_\varepsilon^\alpha(s) = \begin{cases} \left(\frac{1}{4\pi\varepsilon^2}\right)^{\frac{1}{2}} \exp\left(-\frac{s^2}{4\varepsilon^2}\right), & \alpha = 2, \\ \frac{1}{\pi(s^2 + \varepsilon^2)}, & \alpha = 1, \end{cases}$$

which are Gaussian and Cauchy densities, respectively. Although other values of α do not admit a closed-forms for $\phi_\varepsilon^\alpha(s)$, they can be estimated by approximating (5.7). Regardless of α , ϕ^α is symmetric and unimodal. For $\alpha < 2$, the second moment is infinite and for $\alpha < 1$, all moments are infinite.

Fig. 5.2 plots the time-evolutions of the approximate solutions to the nonlocal homogeneous Dirichlet boundary value problem described in (4.2) given by the finite element method with mesh spacing $h = 5 \cdot 10^{-4}$ and $t \in [0, 0.25]$. We consider $\alpha \in \{2, 3/2, 1, 1/2\}$ and various ε . The solutions of with ϕ_ε^2 behave asymptotically, with respect to ε , as solutions to the classical diffusion equation (1.4). However, the asymptotic behavior of solutions of with $\phi_\varepsilon^{\alpha \neq 2}$ is given by a fractional Laplace parabolic equation (1.6). Consequently, the magnitude of the jump discontinuity in the initial data decays more slowly in these latter cases.

REFERENCES

- [1] D. Applebaum. *Lévy Processes and Stochastic Calculus*, volume 93 of *Cambridge studies in advanced mathematics*. Cambridge, 2004.
- [2] G. L. Aranovich and M. D. Donohue. Eliminating the mean-free-path inconsistency in classical phenomenological model of diffusion for fluids. *Physica A: Statistical Mechanics and its Applications*, 373:119–141, 2007.
- [3] K. Barmak, M. Emelianenko, D. Golovaty, D. Kinderlehrer, and S. Taasan. Towards a statistical theory of texture evolution in polycrystals. *SIAM Journal on Scientific Computing*, 30(6):3150–3169, 2008.
- [4] F. Bobaru and M. Duangpanya. The peridynamic formulation for transient heat conduction. *International Journal of Heat and Mass Transfer*, 53(19–20):4047–4059, 2010.
- [5] N. Burch and R. B. Lehoucq. Classical, Nonlocal, and Fractional Diffusion Equations. *International Journal for Multiscale Computational Engineering*, 2010. To appear.
- [6] C. Carrillo and P. Fife. Spatial effects in discrete generation population models. *Journal of Mathematical Biology*, 50(2):161–188, 2005.
- [7] C. Cattaneo. Sulla conduzione del calore. *Atti Sem. Mat. Fis. Univ. Modena*, 3(3):21, 1948.
- [8] E. Chasseigne, M. Chaves, and J. D. Rossi. Asymptotic behavior for nonlocal diffusion equations. *Journal de mathématiques pures et appliquées*, 86(3):271–291, 2006.
- [9] E. Emmrich and O. Weckner. The peridynamic equation of motion in non-local elasticity theory. In *III European Conference on Computational Mechanics. Solids, Structures and Coupled Problems in Engineering, Lisbon, Springer*, volume 19, 2006.
- [10] G. Gilboa and S. Osher. Nonlocal operators with applications to image processing. *UCLA CAM Report*, pages 07–23, 2007.
- [11] M. Gunzburger and R. B. Lehoucq. A nonlocal vector calculus with application to nonlocal boundary value problems. *Multiscale Modeling and Simulation*, 2010. To appear.
- [12] D. D. Joseph and L. Preziosi. Heat waves. *Reviews of Modern Physics*, 61(1):41–73, 1989.
- [13] V. M. Kenkre, E. W. Montroll, and M. F. Shlesinger. Generalized master equations for continuous-time random walks. *Journal of Statistical Physics*, 9(1):45–50, 1973.
- [14] G. Lebon, D. Jou, and J. Casas-Vázquez. *Understanding Non-equilibrium Thermodynamics: Foundations, Applications, Frontiers*. Springer Verlag, 2007.
- [15] R. B. Lehoucq and O. Anatole von Lilienfeld. Translation of Walter Noll’s “Derivation of the Fundamental Equations of Continuum Thermodynamics from Statistical Mechanics”. *Journal of Elasticity*, 100:1–20, 2010.

- [16] F. Mainardi, M. Raberto, R. Gorenflo, and E. Scalas. Fractional calculus and continuous-time finance II: the waiting-time distribution. *Physica A: Statistical Mechanics and its Applications*, 287(3-4):468–481, 2000.
- [17] R. Metzler and J. Klafter. The random walk’s guide to anomalous diffusion: a fractional dynamics approach. *Physics Reports*, 339(1), 2001.
- [18] W. Noll. Die Herleitung der Grundgleichungen der Thermomechanik der Kontinua aus der statistischen Mechanik. *Indiana Univ. Math. J.*, 4:627–646, 1955. Original publishing journal was the J. Rational Mech. Anal. See the English translation [15].

A DISCONTINUOUS VELOCITY LEAST SQUARES FINITE ELEMENT METHOD FOR THE STOKES EQUATIONS WITH IMPROVED MASS CONSERVATION

JAMES LAI[§], PAVEL BOCHEV[¶], LUKE OLSON^{||}, KARA PETERSON^{**}, DENIS RIDZAL^{††}, AND CHRIS SIEFERT^{‡‡}

Abstract. Conventional least squares finite element methods (LSFEM) for incompressible flows conserve mass approximately. In some cases, this can lead to an unacceptable loss of mass and unphysical solutions. In this report we formulate a new, locally conservative LSFEM for the Stokes equations which computes a discrete velocity field that is point-wise divergence free on each element. To this end, we employ discontinuous velocity approximations which are defined by using a local stream-function on each element. The effectiveness of the new LSFEM approach on improved local and global mass conservation is compared with a conventional LSFEM employing standard C^0 Lagrangian elements.

1. Introduction. Least-squares finite element methods (LSFEMs) have been applied to incompressible flows with varying success. The key issue is that LSFEMs are residual minimization schemes and hence conserve mass only approximately. For some problem configurations, this can lead to an unacceptable loss of mass and unphysical solutions. A locally conservative mimetic LSFEM has been defined for the Stokes equations in [4] and [3, Section 7.7] using compatible finite element spaces. However, the mimetic LSFEM requires non-standard boundary conditions specifying the normal velocity and the tangential vorticity on the domain boundary. So far, it has not been extended to the more common and practically important velocity boundary condition and it is not clear whether or not this can be done.

Mass conservation in least squares methods for the Stokes equations with the velocity boundary condition has been studied extensively in literature [6, 9, 10, 13, 14, 15]. Loss of mass in LSFEMs can be countered by mesh refinement [13], high order elements [16], modifying the least-squares functional [15], weighting the continuity equation more strongly [10], or by enforcing it on each element by Lagrange multipliers [9]. However, neither one of these approaches can be deemed completely satisfactory.

Mass conservation does not improve proportionally with mesh refinement—leading to an impractical alternative. High order elements require an increased amount of storage and computation and the improvements to mass conservation are not commensurate with the additional cost [13, 15]. Modifying the least squares functional with terms that promote mass conservation has proven to be very successful [15], however, it is an ad hoc way of improving mass conservation and may depend on the problem on hand. Another alternative is to enforce element-wise mass conservation using Lagrange multipliers [9]. While this approach yields exact mass conservation on each element, it also results in a saddle-point system, thereby negating the main reason one may want to consider LSFEMs.

An idea that has not been explored much in the context of LSFEMs is the use of discontinuous elements. Discontinuous LSFEM can be viewed as generalizations of LSFEMs for transmission and mesh-tying problems; see [1], [3, Section 12.10] and [8], from a fixed number of subdomains to an arbitrary number of subdomains.

In this report we formulate, in two stages, a new locally conservative LSFEM for the Stokes equations with the velocity boundary condition by using discontinuous velocity approximations. Our starting point is a weighted L^2 least-squares formulation [2] employing

[§]University of Illinois at Urbana-Champaign, Department of Computer Science, jhlai2@illinois.edu

[¶]Sandia National Laboratories, pbboche@sandia.gov

^{||}University of Illinois at Urbana-Champaign, Department of Computer Science, lukeo@illinois.edu

^{**}Sandia National Laboratories, kjpeter@sandia.gov

^{††}Sandia National Laboratories, dridzal@sandia.gov

^{‡‡}Sandia National Laboratories, csiefer@sandia.gov

conventional C^0 elements and the velocity-vorticity-pressure (VVP) form of the Stokes equations. The first stage relaxes the continuity of the velocity field only and adds new terms which penalize the normal and the tangential jumps of the velocity across the element interfaces. We show that by adjusting the relative importance of the normal and tangential jump terms this intermediate *discontinuous velocity* LSFEM can lead to noticeable improvements in the mass conservation. However, the weights required for improved mass conservation differ from problem to problem, thereby making this formulation insufficiently robust for practical problems.

At the second stage, we proceed to define the discontinuous velocity field on each element as the curl of a local stream-function. This guarantees that the velocity is pointwise divergence free on each element. Thus, our approach can be interpreted as implementation of the intermediate *discontinuous velocity* LSFEM using locally divergence free basis for the velocity. This idea bears some similarity with the *discrete* LSFEM in [7] with two crucial distinctions. First and foremost, the method in [7] is not a discontinuous formulation; in order to cope with the discontinuity in the approximating space this method replaces the differential operators by weak discrete versions defined using integration by parts. The second distinction is that we eliminate completely the velocity and work directly with the stream function, whereas [7] retains the original fields.

The resulting discontinuous stream-function-vorticity-pressure (SVP) LSFEM is locally conservative and offers a much improved global and local mass conservation compared to its parent LSFEM employing C^0 elements. We demonstrate the usefulness of the new formulation through a series of numerical examples.

1.1. Notation. For simplicity we restrict attention to two space dimensions and bounded, simply connected regions $\Omega \subset \mathbb{R}^2$ with Lipschitz-continuous boundary. In what follows we use the standard notation $H^k(\Omega)$ for the Sobolev space of all square integrable functions which have square integrable derivatives of orders up to k . The norm and inner product on H^k are $\|\cdot\|_k$ and $(\cdot, \cdot)_k$, respectively.

As usual, when $k = 0$ we write $L^2(\Omega)$, (\cdot, \cdot) and $\|\cdot\|_0$. The symbol $H_0^1(\Omega)$ denotes a subspace of $H^1(\Omega)$ of functions whose trace vanishes on $\partial\Omega$ and $L_0^2(\Omega)$ is the subspace of L^2 fields with vanishing mean. The dual of $H_0^1(\Omega)$ is the space $H^{-1}(\Omega)$ with norm

$$\|u\|_{-1} = \sup_{v \in H_0^1(\Omega)} \frac{(u, v)}{\|v\|_1}. \quad (1.1)$$

Vector valued fields and their associated function spaces are denoted by bold face symbols, e.g., $\mathbf{u} = (u_1, u_2)$ is a vector field in two dimensions and $\mathbf{H}^1(\Omega)$ is the Sobolev space of vector fields with components are in $H^1(\Omega)$. In two dimensions, the curl is defined for scalar and vector functions as

$$\nabla \times \omega = \begin{bmatrix} \omega_y \\ -\omega_x \end{bmatrix}, \quad \nabla \times \mathbf{u} = u_{2x} - u_{1y}. \quad (1.2)$$

We use \mathcal{K} to denote a partition of Ω into finite elements K . In two dimensions K can be a quadrilateral or a triangle and the interface between two elements is an edge e . The sets of all interior and boundary edges in the mesh are denoted by $\mathcal{E}(\Omega)$ and $\mathcal{E}(\Gamma)$, respectively. Finally, $\mathcal{E} = \mathcal{E}(\Omega) \cup \mathcal{E}(\Gamma)$ is the set of all edges in the mesh.

The standard C^0 finite element spaces of degree $r > 0$ on quadrilateral and triangular grids are denoted by Q_r and P_r , respectively. We will also need their discontinuous versions $[Q_r]$ and $[P_r]$. When the type of the element is not important we write R_r and $[R_r]$ with the understanding that $R_r = Q_r$ on quadrilaterals and $R_r = P_r$ on triangles.

Discontinuous finite element methods require various jump terms on element interfaces. Let K_+ and K_- be two adjacent elements that share edge e and denote the velocities on each element by \mathbf{u}^+ and \mathbf{u}^- respectively. Define the jump in normal and tangential components across e as

$$[\mathbf{u} \cdot \mathbf{n}] = \mathbf{u}^+ \cdot \mathbf{n}^+ + \mathbf{u}^- \cdot \mathbf{n}^-, \quad [\mathbf{u} \times \mathbf{n}] = \mathbf{u}^+ \times \mathbf{n}^+ + \mathbf{u}^- \times \mathbf{n}^-. \quad (1.3)$$

where \mathbf{n}^+ and \mathbf{n}^- are the outer normals on ∂K^+ and ∂K^- respectively. The jump of a scalar function is defined as usual by the difference

$$[\psi] = \psi^+ - \psi^-. \quad (1.4)$$

2. The continuous prototype least-squares method. In this section we review the weighted L^2 least-squares method for the Stokes equations which is the prototype for the discontinuous, locally conservative LSFEM. In terms of the primitive variables the governing equations assume the form

$$\begin{cases} -\Delta \mathbf{u} + \nabla p &= \mathbf{f} & \text{on } \Omega \\ \nabla \cdot \mathbf{u} &= 0 & \text{on } \Omega \end{cases} \quad (2.1)$$

where \mathbf{u} and p are the velocity and the pressure, respectively, and \mathbf{f} is a given vector function specifying the body force. The system (2.1) is augmented with the velocity boundary condition

$$\mathbf{u} = 0 \quad \text{on } \partial\Omega \quad (2.2)$$

and the zero mean pressure constraint

$$\int_{\Omega} p \, d\Omega = 0. \quad (2.3)$$

The first equation in (2.1) governs conservation of momentum while the second (continuity equation) governs conservation of mass.

Least-squares methods for (2.1), (2.2) and (2.3) are usually defined using an equivalent first-order form of the Stokes equations. This eliminates the need for globally H^2 -conforming finite elements which require C^1 continuity and are difficult to construct. There are several first order formulations of the Stokes equations to choose from, the most common being the velocity-vorticity-pressure formulation in which the vorticity

$$\omega = \nabla \times \mathbf{u} \quad (2.4)$$

is introduced as a new variable. Using the identity,

$$\nabla \times \nabla \times \mathbf{u} = -\Delta \mathbf{u} + \nabla(\nabla \cdot \mathbf{u}) \quad (2.5)$$

and the continuity equation, we arrive at the velocity-vorticity-pressure (VVP) first order formulation of the Stokes equations

$$\begin{cases} \nabla \times \omega + \nabla p &= \mathbf{f} & \text{on } \Omega \\ \omega - \nabla \times \mathbf{u} &= 0 & \text{on } \Omega \\ \nabla \cdot \mathbf{u} &= 0 & \text{on } \Omega \end{cases} \quad (2.6)$$

The VVP system is augmented with the velocity boundary condition (2.2) and the zero mean constraint (2.3).

2.1. Weighed L^2 least-squares method. LSFEMs define unconstrained minimization problems via residual minimization over an appropriate Hilbert space. Thus, the LSFEM solution is given by the solution to the minimization of a norm-equivalent functional devised using the squares of the residuals of each equation of the partial differential equation in the appropriate norm. The resulting discretized system that minimizes the functional over the finite element subspace is guaranteed to be symmetric and positive definite.

One can show that for the VVP system with velocity boundary conditions, the negative norm functional

$$J_{-1}(\mathbf{u}, \omega, p; \mathbf{f}) = \|\nabla \times \omega + \nabla p - \mathbf{f}\|_{-1}^2 + \|\nabla \times \mathbf{u} - \omega\|_0^2 + \|\nabla \cdot \mathbf{u}\|_0^2 \quad (2.7)$$

is norm equivalent on $X_{-1} = [H_0^1(\Omega)]^2 \times L^2(\Omega) \times L_0^2(\Omega)$. A least squares principle for (2.6) is to minimize (2.7) over X_{-1} .

The negative norm (1.1) admits the following characterization [3].

THEOREM 2.1. *For any $u \in H^{-1}(\Omega)$*

$$\|u\|_{-1}^2 = \|(-\Delta)^{-1/2}u\|_0^2 \quad (2.8)$$

This theorem reveals that the negative norm is not easily computable because it requires inversion of the Laplace operator. Therefore, to obtain a practical LSFEM it must be approximated. The diagonal operator

$$(-\Delta)^{-1/2} \mapsto h\mathcal{I} \quad (2.9)$$

gives a simple, yet sufficiently accurate for our purposes approximation of the negative norm [3]. Using (2.9) the first term of (2.7) is approximated by

$$\|\nabla \times \omega + \nabla p - \mathbf{f}\|_{-1}^2 \approx h^2 \|\nabla \times \omega + \nabla p - \mathbf{f}\|_0^2 \quad (2.10)$$

We arrive at the following discrete version of (2.7)

$$J_{-1}^h(\mathbf{u}^h, \omega^h, p^h; \mathbf{f}) = h^2 \|\nabla \times \omega^h + \nabla p^h - \mathbf{f}\|_0^2 + \|\nabla \times \mathbf{u}^h - \omega^h\|_0^2 + \|\nabla \cdot \mathbf{u}^h\|_0^2 \quad (2.11)$$

where $(\mathbf{u}^h, \omega^h, p^h) \in X_r^h = [R_r(\Omega) \cap H_0^1(\Omega)]^2 \times R_{r-1}(\Omega) \cap H^1(\Omega) \times R_{r-1} \cap L_0^2(\Omega)$, $r > 1$. We refer to (2.11) as the weighted L_2 method¹ since it is composed of L_2 norms of the squares of the residuals of each equation scaled by an approximate mesh weight. In what follows we restrict attention to the lowest-order admissible space, i.e., $r = 2$.

One can show that (2.11) is well-posed and optimally convergent formulation [3]. In particular, the following theorem holds [3].

THEOREM 2.2. *Let $(\mathbf{u}^h, \omega^h, p^h) \in X_2^h$ be a solution to (2.11), and $(\mathbf{u}, p, h) \in X$ be the exact solution to (2.6), such that $\mathbf{u} \in \mathbf{H}^3(\Omega)$, $\omega \in H^2(\Omega)$ and $p \in H^2(\Omega)$. There exists a constant $C > 0$ such that*

$$\|\mathbf{u} - \mathbf{u}^h\|_1^2 + \|\omega - \omega^h\|_0 + \|p - p^h\|_0 \leq Ch^2 (\|\mathbf{u}\|_3 + \|\omega\|_2 + \|p\|_2). \quad (2.12)$$

¹For simplicity, in our implementation of the weighted method the one dimensional nullspace of the pressure is eliminated by setting the pressure on the boundary to zero at one point instead of enforcing (2.3). These two approaches to handling the one dimensional nullspace of the pressure are equivalent; however, the choice affects the convergence of the iterative method used to solve the system. A comparison can be found in [3].

From Theorem 2.2, we see that using a quadratic/biquadratic approximation for the velocity and linear/bilinear approximations for the vorticity and pressure result in optimal convergence rates. Nonetheless, for simplicity we work the equal-order version of the finite element space

$$\overline{X}_2^h = (R_2(\Omega) \cap H_0^1(\Omega))^2 \times R_2(\Omega) \cap H^1(\Omega) \times R_2 \cap L_0^2(\Omega). \quad (2.13)$$

We use (2.11) as a basis for our discontinuous velocity LSFEM.

2.2. Mass conservation in the weighted L^2 least-squares method. Theorem 2.2 asserts that the weighted L^2 method is optimally accurate for all sufficiently smooth exact solutions of the Stokes equations. This means that asymptotically $\|\nabla \cdot u\| \rightarrow 0$, as $h \rightarrow 0$. However, on a given fixed mesh size this term cannot be guaranteed to be as small as may be required, nor could its convergence to zero be assured for insufficiently smooth velocity fields. In this section we show that these concerns are not unfounded and that in some cases mass loss in the weighted least-squares method can be significant.

To this end we consider two standard test problems: the backward-facing step flow, shown in Fig. 2.1, and a channel flow past a cylinder, shown in Fig. 2.2. For the backward-facing step problem the domain is the rectangle $[0, 10] \times [0, 1]$ with a reentrant corner at $(2, 0.5)$. The velocity boundary condition is specified as follows. On the inflow ($x = 0$) and outflow ($x = 10$) walls

$$\mathbf{u}_{in} = \begin{bmatrix} 8(y - 0.5)(1 - y) \\ 0 \end{bmatrix} \quad \text{and} \quad \mathbf{u}_{out} = \begin{bmatrix} y(1 - y) \\ 0 \end{bmatrix}, \quad (2.14)$$

respectively. Along all other portions of the boundary $\mathbf{u}_{wall} = \mathbf{0}$ is enforced.

The geometry of the second test problem is given by the rectangle $[-1, 3] \times [-1, 1]$ with a disk of radius $r = 0.6$ centered at $(0, 0)$, removed from the domain. The velocity boundary condition for this problem is set as follows. On the inflow ($x = -1$), outflow ($x = 3$), top ($y = 1$) and bottom ($y = -1$) sides

$$\mathbf{u}_{in} = \mathbf{u}_{out} = \mathbf{u}_{wall} = \begin{bmatrix} (1 - y)(1 + y) \\ 0 \end{bmatrix}. \quad (2.15)$$

and on the surface of the “cylinder” $\mathbf{u}_{cyl} = \mathbf{0}$. Therefore, velocity is set to zero on all parts of the boundary except for the inflow and the outflow portions of $\partial\Omega$.

Note that in both test problems specification of the velocity boundary condition is compatible with $\nabla \cdot \mathbf{u} = 0$ because fluid enters and leaves the domain only through the inflow and the outflow boundaries, respectively and

$$\int_{\Gamma_{in}} \mathbf{u}_{in} \cdot \mathbf{n} \, d\ell = \int_{\Gamma_{out}} \mathbf{u}_{out} \cdot \mathbf{n} \, d\ell \quad (2.16)$$

To assess the mass conservation properties of the least-squares methods considered in this report we measure the total mass flow across several vertical surfaces connecting the top and the bottom sides of the computational domains. The lines marked by “S” in Figures 2.1-2.2 show two typical examples of such surfaces for the two test problems. Because the greatest mass loss for the backward-facing step is expected near the reentrant corner we always place one of the surfaces at $x = 2$. For the second test problem we always measure the flow across the surface at $x = 0$ where the domain narrows due to the cylindrical cutout.

Because for both test problems velocity is zero on all parts of the boundary except Γ_{in} and Γ_{out} , from the divergence theorem it follows that

$$\int_{\Gamma_{in}} \mathbf{u} \cdot \mathbf{n}_{in} \, d\ell = \int_S \mathbf{u} \cdot \mathbf{n}_S \, d\ell. \quad (2.17)$$

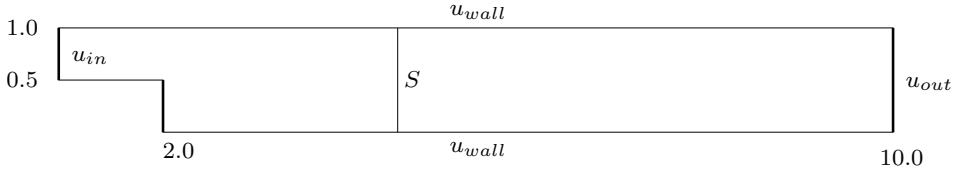


FIG. 2.1. Geometry of the first test problem: backward-facing step.

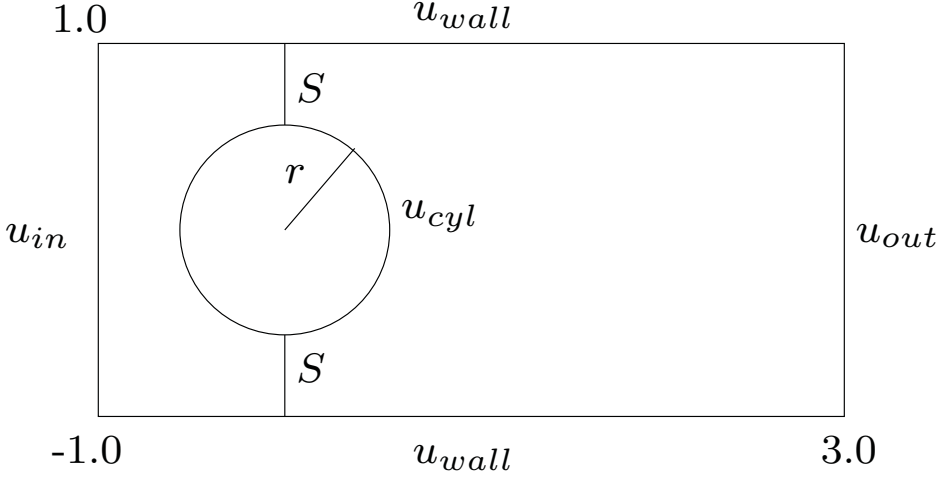


FIG. 2.2. Geometry of the second test problem: flow past a cylinder.

Therefore, mass conservation can be quantified by the present mass loss across the surface S , defined as follows:

$$m_{loss} = \frac{\int_{\Gamma_{in}} \mathbf{u} \cdot \mathbf{n}_{in} d\ell - \int_S \mathbf{u} \cdot \mathbf{n}_S d\ell}{\int_{\Gamma_{in}} \mathbf{u} \cdot \mathbf{n}_{in} d\ell}. \quad (2.18)$$

To assess mass conservation properties of the weighted L^2 formulation we solve the two test problems using the following modified version of the weighted L^2 least-squares functional

$$J_\mu^h(\mathbf{u}^h, \omega^h, p^h; \mathbf{f}^h) = h^2 \|\nabla \times \omega^h + \nabla p^h - \mathbf{f}^h\|_0^2 + \|\nabla \times \mathbf{u}^h - \omega^h\|_0^2 + \mu \|\nabla \cdot \mathbf{u}^h\|_0^2 \quad (2.19)$$

implemented using the equal order space (2.13). This modification has been proposed in [10] as a way to improve mass conservation in least-squares methods. By increasing μ we increase the relative importance of the residual of the continuity equation, thereby promoting mass conservation. In our study we use $\mu = 1$, $\mu = 10$ and $\mu = 20$.

Our results are summarized in Figure 2.3. We see that for $\mu = 1$ the least-squares solution of the backward-facing step problem experiences severe mass loss in excess of 50% of the total mass near the reentrant corner. Increasing μ does improve conservation, however, mass

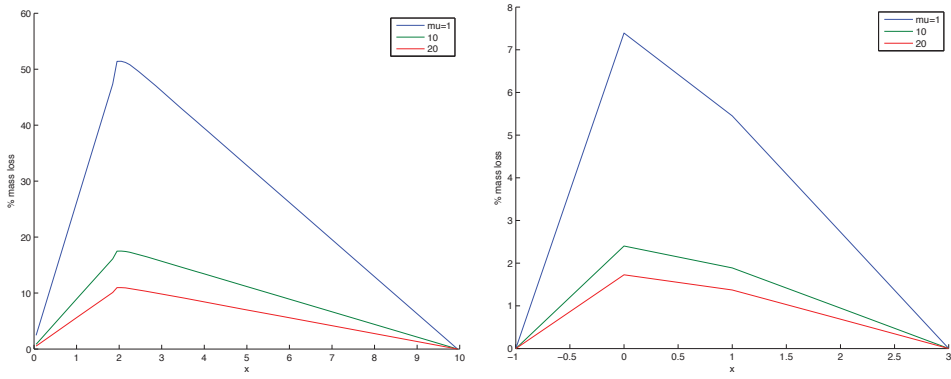


FIG. 2.3. Percent mass loss of (2.19) for the backward-facing step (left panel) and the flow past a cylinder (right panel) test problems.

loss remains unacceptably high even for $\mu = 20$. We note that significant increase of μ is not recommended as this will reduce the accuracy of the other terms in the functional and compromise, e.g., conservation of momentum. Indeed, by increasing the weight of a single term in the least squares functional, it is in effect *decreasing* the weight of the other terms. Thus, by choosing a large weight for μ to promote mass conservation, we are effectively demoting conservation of momentum. The mass loss in the second test problem is not as severe but still significant at 7%. In this case, setting $\mu = 20$ helps to bring down the loss of mass across the narrowings to about 2%.

REMARK 1. *Exact element-wise mass conservation with C^0 elements has been achieved in the so-called restricted least-squares method [9]. In the restricted LSFEM mass conservation on each element is added as an explicit constraint leading to the following constrained minimization problem:*

$$\begin{aligned} \min_{X^h} \quad & J_{-1}^h(\mathbf{u}^h, \omega^h, p^h; \mathbf{f}) \\ \text{subject to} \quad & \int_K (\nabla \cdot \mathbf{u}^h) dK = 0, \quad \forall K \in \mathcal{K} \end{aligned} \quad (2.20)$$

Although (2.20) returns a solution with exact element-wise mass conservation, the system is typically solved using Lagrange multipliers and results in a saddle-point system which negates the advantages of using least-squares in the first place. The constrained optimization problem can also be solved by a penalty approach, in which case one is led back to a formulation similar to (2.19) with a very large μ . Because the penalty must be strong enough to enforce the constraint accurately, the penalty formulation of (2.20) suffers from the same disadvantages as (2.19).

In the next section we explore an alternative approach to improve mass conservation in least-squares methods based on allowing discontinuous velocity spaces in the formulation.

3. Discontinuous velocity least-squares finite element method. Numerical results in the last section show that least-squares methods with C^0 elements can suffer from severe mass loss which in some cases may exceed 50% of the total mass. Furthermore, the remedies available to counter this loss are not satisfactory: weighting strongly the continuity equation residual as in (2.19) reduces conservation of momentum, while using the restricted formulation (2.20) leads to a saddle-point problem and negates the advantages of least-squares.

The option of using div-conforming elements to achieve exact mass conservation in least-squares methods has been explored in [4]. However, the resulting mimetic LSFEM requires

non-standard boundary conditions for the Stokes equations, and its extension to the practically important velocity boundary condition is not clear.

Consequently, in order to improve mass conservation in LSFEMs for the Stokes equation with the velocity boundary condition we propose to employ a *discontinuous* finite element approximation of the velocity, while retaining C^0 elements for the rest of the variables. In so doing we achieve two objectives. First, we keep the growth of the degrees of freedom to a minimum, compared to a fully discontinuous formulation. Second, relaxation of the interelement continuity of the velocity space allows a greater flexibility in the choice of the local finite element approximation of that variable. In particular, it becomes possible to consider locally divergence-free spaces which would have been impractical if the global velocity space had to be H^1 -conforming.

Following these ideas we develop a *discontinuous* velocity least-squares finite element method based on the well-posed formulation (2.11) in two stages. At the first stage we allow discontinuous finite elements for the velocity in (2.11), i.e., we set

$$\tilde{X}_r^h = ([R]_r)^2 \times R_{r-1} \times R_{r-1}. \quad (3.1)$$

This necessitates some changes in the least-squares functional, namely, the last two terms have to be broken into element sums to deal with the loss of conformity in the velocity space:

$$\begin{aligned} \tilde{J}_{-1}^h(\mathbf{u}^h, \omega^h, p^h; \mathbf{f}^h) = \\ h^2 \|\nabla \times \omega^h + \nabla p^h - \mathbf{f}^h\|_0^2 + \sum_{K \in \mathcal{K}} \left(\|\nabla \times \mathbf{u}^h - \omega^h\|_{0,K}^2 + \|\nabla \cdot \mathbf{u}^h\|_{0,K}^2 \right) \end{aligned} \quad (3.2)$$

Furthermore, to obtain a well-posed formulation with a unique solution, we need to recover some of the H^1 -conformity qualities of the velocity. Therefore, constraints on the jumps in normal and tangential components of the velocity are introduced.

Recall that $\mathcal{E}(\Omega)$ is the set of all interior edges in the mesh. It is easy to see that the weighted L^2 least-squares method (2.11) is equivalent to the following constrained minimization problem

$$\begin{aligned} \min_{\tilde{X}_r^h} \tilde{J}_{-1}^h(\mathbf{u}^h, \omega^h, p^h; \mathbf{f}^h) \\ \text{subject to } \int_{e_i} [\mathbf{u}^h \cdot \mathbf{n}_i] d\ell = 0 \quad \text{and} \quad \int_{e_i} [\mathbf{u}^h \times \mathbf{n}_i] d\ell = 0 \quad \forall e_i \in \mathcal{E}(\Omega) \end{aligned} \quad (3.3)$$

The constrained system can be solved by Lagrange multipliers in which case the resulting minimization problem becomes

$$\min_{\tilde{X}_r^h} \max_{\mathbb{R}^{|\mathcal{E}|}} \tilde{J}_{-1}^h(\mathbf{u}^h, \omega^h, p^h; \mathbf{f}^h) - \sum_{e_i \in \mathcal{E}(\Omega)} \lambda_1^i \int_{e_i} [\mathbf{u}_i \cdot \mathbf{n}_i] d\ell - \sum_{e_i \in \mathcal{E}} \lambda_2^i \int_{e_i} [\mathbf{u}_i \times \mathbf{n}_i] d\ell \quad (3.4)$$

Of course, similar to (2.20), this formulation is a saddle-point system that gives rise to an indefinite algebraic system.

Instead of using Lagrange multipliers we will encourage H^1 conformity by a penalty approach—by adding residuals of the interelement jumps to the least-squares functional. This gives rise to the following *discontinuous* velocity functional:

$$\begin{aligned} \tilde{J}_{-1}^h(\mathbf{u}^h, \omega^h, p^h; \mathbf{f}^h) = \\ h^2 \|\nabla \times \omega^h + \nabla p^h - \mathbf{f}^h\|_0^2 + \sum_{K \in \mathcal{K}} \left(\|\nabla \times \mathbf{u}^h - \omega^h\|_{0,K}^2 + \|\nabla \cdot \mathbf{u}^h\|_{0,K}^2 \right) \\ + h^{-1} \sum_{e_i \in \mathcal{E}(\Omega)} \left(\alpha_1 \int_{e_i} [\mathbf{u} \cdot \mathbf{n}_i]^2 d\ell + \alpha_2 \int_{e_i} [\mathbf{u} \times \mathbf{n}_i]^2 d\ell \right) \end{aligned} \quad (3.5)$$

where $\alpha_1, \alpha_2 > 0$ are penalty parameters. The values of these constants can be used to adjust the relative importance of normal vs. tangential continuity.

Based on analogies with div-conforming elements, one could argue that strengthening the normal continuity of the velocity field should lead to improved mass conservation in the finite element solution of (3.5). If this were the case, then the discontinuous velocity formulation (3.5) with $\alpha_1 \gg \alpha_2$ should be able to take care of the mass losses in our two test problems. To test this hypothesis we implement (3.5) using the equal-order discontinuous velocity, continuous vorticity and pressure finite element space

$$[\overline{X}_2]^h = ([R_2])^2 \times R_2 \times R_2 \quad (3.6)$$

and solve the two test problems with two different choices for α_1 and α_2 . The first choice is to set $\alpha_1 = \alpha_2 = 100$, in which case we expect² to see mass losses comparable to that in the C^0 formulation. The second set of weights is $\alpha_1 = 100$, $\alpha_2 = 0.01$ emphasizes normal over tangential continuity. If our hypotheses were correct, this set of weights would lead to a much improved mass conservation.

Unfortunately, the results shown in Fig. 3.1 refute our seemingly logical hypothesis. The

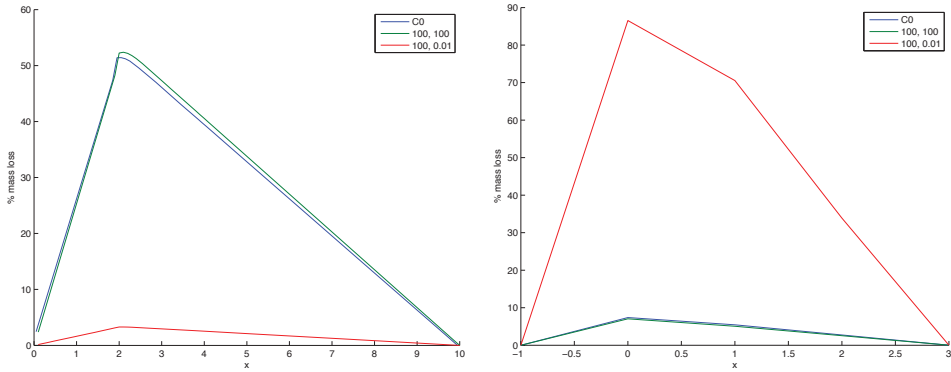


FIG. 3.1. Percent mass loss in the discontinuous velocity least-squares method (3.5) for the backward-facing step (left panel) and the flow past a cylinder (right panel) test problems. Green line corresponds to $\alpha_1 = \alpha_2 = 100$, red line corresponds to $\alpha_1 = 100$, $\alpha_2 = 0.01$ and the blue line gives the reference mass loss by the C^0 least-squares method (2.11). The legend values are read as α_1, α_2 with (2.11) as reference labeled (C_0).

left panel in the figure shows that for the backward-facing step problem the second weight combination does lead to a *significant improvement* in the mass conservation by reducing the mass loss from over 50% to just a bit over 3%. However, for the flow past a cylinder the situation is completely reversed. Now the choice $\alpha_1 = 100$, $\alpha_2 = 0.01$ leads to a *significant deterioration* of the mass conservation and increases mass loss from 7% in the C^0 formulation to *nearly 90%*! These results clearly indicate that the discontinuous velocity formulation (3.5) cannot be reliably counted on to always reduce the mass loss with the same choice of weights, i.e., its mass conservation properties are problem dependent. This is an undesirable feature that we shall deal with at the second stage of the formulation of our new method.

To motivate this stage we note that while discontinuous velocity does allow for improvements in mass conservation, the least-squares formulation (3.5) does not enforce exact mass conservation on each element. At the same time, considering that the velocity space is not

²This is because in the limit as $\alpha_1 \rightarrow \infty$ and $\alpha_2 \rightarrow \infty$, (3.5) recovers the C^0 solution of the weighted L^2 LSFEM method.

subject to any interelement continuity, it is obvious that we have a greater flexibility in choosing the velocity representation on each element than in the C^0 setting. In particular, we can take advantage of this flexibility by choosing the velocity to be *pointwise* divergence free on each element by setting

$$\mathbf{u}^h|_K = \nabla \times \psi^h|_K \quad \forall K \in \mathcal{K}, \quad (3.7)$$

where $\psi^h \in [R]_r$ is a discontinuous *stream function*. Therefore, at the second stage we replace the velocity field in (3.5) with the field defined in (3.7). Note that when defining \mathbf{u}^h in this way, $\nabla \cdot \mathbf{u}^h = 0$ is automatically satisfied and hence the residual of the continuity equation can be dropped from the least-squares functional. However, a term that penalizes the jump of the stream function must be added to the functional. Furthermore, because velocity is eliminated, the velocity boundary condition must be implemented through the stream function. It is easy to see that $\mathbf{n} \cdot \nabla \times \psi$ involves only tangential derivatives of ψ . Therefore, a Dirichlet boundary condition on the stream-function specifies the normal component of the velocity. We specify the tangential component of the velocity weakly by adding another least-squares term to our functional. As a result, we arrive at the following discontinuous stream function-vorticity-pressure (SVP) least-squares functional:

$$\begin{aligned} \widehat{J}_{-1}^h(\psi^h, \omega^h, p^h; \mathbf{f}^h) = & h^2 \|\nabla \times \omega^h + \nabla p^h - \mathbf{f}^h\|_0^2 + \sum_{K \in \mathcal{K}} \|\nabla \times \nabla \times \psi^h - \omega^h\|_{0,K}^2 \\ & + h^{-1} \sum_{e_i \in \mathcal{E}(\Omega)} \left(\alpha_1 \int_{e_i} [(\nabla \times \psi^h) \cdot \mathbf{n}_i]^2 d\ell + \alpha_2 \int_{e_i} [(\nabla \times \psi^h) \times \mathbf{n}_i]^2 d\ell \right) \\ & + h^{-1} \sum_{e_i \in \mathcal{E}(\Gamma)} |(\nabla \times \psi^h) \times \mathbf{n}_i|^2 d\ell + h^{-3} \sum_{e_i \in \mathcal{E}(\Omega)} \int_{e_i} [\psi^h]^2 d\ell \end{aligned} \quad (3.8)$$

The weight for the last term of (3.8) is determined by a scaling argument assuming that $\psi \in H^2$ and hence its trace is in $H^{3/2}$. The jump of the stream-function is necessary for elements not adjacent to the boundary since constraining only $[\mathbf{n} \cdot \nabla \times \psi]$ and $[\mathbf{n} \times \nabla \times \psi]$ specifies ψ only up to a constant. Once (3.8) is solved, the velocity is recovered through formula (3.7), i.e., on each element

$$\mathbf{u}^h|_K = \nabla \times \psi^h|_K. \quad (3.9)$$

We can view the discontinuous SVP formulation (3.8) as a special case of the discontinuous velocity formulation (3.5) with a specific choice of a *divergence-free* basis. We choose to define this basis through a stream function as in (3.7) primarily because of the simplicity of this choice; however, it should be clear that our approach can easily accommodate any choice of a divergence-free velocity basis.

It is worth pointing out that the discrete least-squares method for the Darcy flow in two-dimensions [7] uses a discontinuous finite element space for the flux defined in a similar manner by

$$\mathbf{V}^h = \nabla(S^h) \oplus \nabla \times (S^h), \quad (3.10)$$

where S_D^h and S_N^h are standard C^0 finite element spaces constrained by zero on the Dirichlet and Neumann portions of the boundary. The key difference is that our approach deals with the discontinuity of the approximating space by including appropriate jump terms and retaining the original differential operators, whereas [7] retains the global inner products but switches to weak discrete differential operators defined using integration by parts.

The use of stream functions is not a novel idea, and has been applied to the Stokes equations [11], however, most research on the SVP formulation is done using finite differences because of the presence of the second derivative. However, in the discontinuous framework, it is not necessary to construct a global $H^2(\Omega)$ -conforming finite element space as $\nabla \times \nabla \times \psi$ is only needed locally.

4. Implementation. All of the above methods are implemented using Intrepid [5] and solved using the KLU solver of Amesos [17], both packages of Trilinos [12]. Intrepid is a local framework that implements basis functions for H^1 , $H(curl)$, and $H(div)$. Since our formulations are discontinuous, it suffices to choose basis functions to be H^1 on each element and implement the jump terms.

It is easy to convert least squares functionals to an implementable weak form by setting the first directional derivative to zero. For example, the weak form that minimizes (3.5) is to find $(\mathbf{u}^h, \omega^h, p^h) \in X_r^h$, such that

$$\begin{aligned} & (\nabla \times \omega^h + \nabla p^h, \nabla \times s^h + \nabla q^h) + \sum_{K \in \mathcal{K}} (\nabla \times \mathbf{u}^h - \omega^h, \nabla \times \mathbf{v}^h - s^h)_{0,K} \\ & + \sum_{e_i \in \mathcal{E}(\Omega)} \left(\int_{e_i} [\mathbf{u}^h \cdot \mathbf{n}] [\mathbf{v}^h \cdot \mathbf{n}] d\ell + \int_{e_i} [\mathbf{u}^h \times \mathbf{n}] [\mathbf{v}^h \times \mathbf{n}] d\ell \right) = (f, \nabla \times s^h + \nabla q^h)_0 \end{aligned} \quad (4.1)$$

for all $(\mathbf{v}, s, q) \in [H_0^1(\Omega)]^2 \times H^1(\Omega) \times L_0^2(\Omega)$. The weak form for all other least squares functionals can be obtained in a similar way.

Since $\mathbf{u} = (u_1^h, u_2^h)$ is a vector valued function, each component has separate degrees of freedom and in the cases where \mathbf{u}^h is discontinuous, each element has its own set of degrees of freedom for u_1^h and u_2^h .

4.1. Transformations. All basis functions are defined on the reference element, thus, it is necessary to use the correct transformation. Intrepid provides implementations for the most common transformations; however, non standard transformations such as $\nabla \times \omega$ (curl of a scalar field in two-dimensions) and (for the SVP formulation) $\nabla \times \nabla \times \psi$ are not straightforward. $\nabla \times \omega$ is the curl of a scalar function and hence is an element of $H(div)$. Thus we use `HDIVtransformVALUE` for curls of scalar functions. In two dimensions, using the identity,

$$\nabla \times \nabla \times \psi = -\psi_{xx} - \psi_{yy} \quad (4.2)$$

it follows that on a reference element, we can compute $\nabla \times \nabla \times \psi$ by using `OPERATOR_D2` which computes all second derivatives of ψ . Once $\nabla \times \nabla \times \psi$ is computed for the reference element, it is necessary to transform to the physical element. This is done by noting that $\nabla \times \psi \in H(div)$, and $\nabla \times \mathbf{v}$, for a vector valued function \mathbf{v} , is a *rotated divergence* and hence `HDIVtransformDIV` transformation is used.

4.2. Boundary conditions. In all of our tests, we use the velocity boundary condition where $\mathbf{u}|_{\partial\Omega} = \mathbf{u}_D$ is specified on the entire boundary. We set the pressure to 0 at a single point on the boundary. Since the basis in Intrepid is interpolatory, the boundary conditions are set strongly by specifying

$$\begin{aligned} & \mathbf{u}(x_i) \mathbf{u}_D(x_i) \quad \forall x_i \in \partial\Omega \\ & p(x_0) = 0 \end{aligned} \quad (4.3)$$

This is done by defining a vector u_0 that is zero for all degrees of freedom corresponding to interior points and equal to \mathbf{u}_D at the boundary degrees of freedom. We then set

$$\mathbf{b} \leftarrow \mathbf{b} - A\mathbf{u}_0 \quad (4.4)$$

Each row and column of A corresponding to a boundary degree of freedom is set to zero and the diagonal is set to 1.

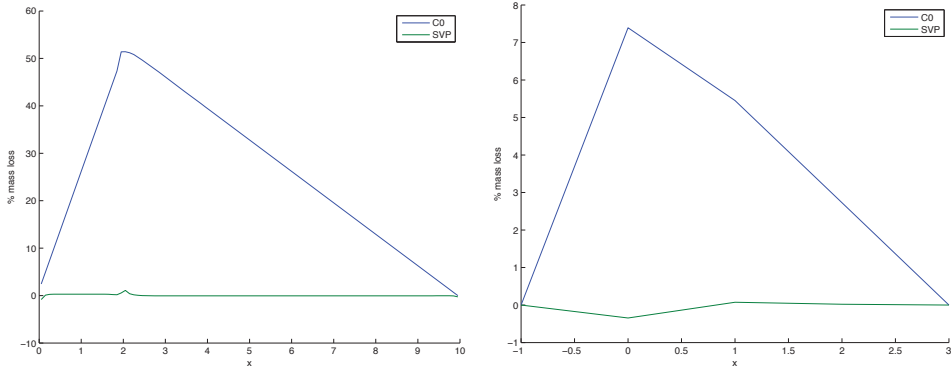


FIG. 5.1. Comparison of mass loss of for the backward-facing step (left panel) and the flow past a cylinder (right panel) test problems. Blue line represents weighted L^2 formulation (2.19), green line is SVP formulation (3.8).

5. Numerical examples. Because the discontinuous SVP formulation (3.8) does not include the velocity some care must be exercised in setting the velocity boundary condition for our two test problems. In the case of the backward step, recall that the boundary condition is given by (2.14). Because on Γ_{in} and Γ_{out} velocity is only a function of y , the \mathbf{u}_1 component is integrated to obtain an equivalent boundary condition on the stream function:

$$\psi_{in} = -\frac{8}{3}y^3 + 6y^2 - 4y + C_1, \quad \psi_{out} = \frac{y^2}{2} - \frac{y^3}{3} + C_2 \quad (5.1)$$

The constants C_1 and C_2 are chosen so that $\mathbf{u}_{in}(0.5) = \mathbf{u}_{out}(0)$ and $\mathbf{u}_{in}(1) = \mathbf{u}_{out}(1)$. The top and bottom walls are then chosen to be constants equal to $\mathbf{u}_{in}(1)$ and $\mathbf{u}_{in}(0.5)$ respectively. Likewise, the equivalent stream function boundary conditions for the second domain, Figure 2.2, with velocity boundary conditions (2.15) are

$$\psi_{in} = \psi_{out} = \psi_{wall} = y - \frac{y^3}{3}. \quad (5.2)$$

However, setting the boundary conditions in this way enforces only the normal component of the velocity. In our test cases, the tangential velocity vanishes on all boundaries. We set the tangential velocity weakly by including

$$\|\mathbf{n} \times \nabla \times \psi\|_{1/2,\Gamma}^2 \approx h^{-1} \|\mathbf{n} \times \nabla \times \psi\|_{0,\Gamma}^2 = h^{-1} \sum_{e_i \in \mathcal{E}(\Gamma)} \int_{e_i} |\mathbf{n} \times \nabla \times \psi|^2 d\ell \quad (5.3)$$

in the least squares functional (3.8).

The resulting mass loss for (3.8) is summarized in Figure 5.1 and it is seen that mass conservation is significantly improved. Indeed, for the backward facing step, the maximum mass loss is less than 1.09% with most of the mass loss centralized at the reentrant corner. On the rest of the domain, the solution is basically conserved over any closed subdomain. This is a dramatic improvement compared to (2.19). For the channel flow with cylinder cutout, the mass conservation is also improved with a slight mass *gain* of 0.34% at the opening of the cylinder. Compared with (3.5), the stream function formulation is able to achieve better mass conservation than (2.19)—recall that no matter how α_1 and α_2 were chosen, the mass conservation could not improve past the weighted L^2 formulation.

The velocity fields of each formulation are visualized in Figures 5.2 and 5.5 and in the case of the backward facing step, the mass loss for (3.5) is clearly visible. For the SVP

formulation, the propagation of the parabolic profile of the inflow is clearly seen throughout the domain. In the case of the weighted L^2 formulation, the parabolic profile diminishes—symbolic of the 50% mass loss. From the additional figures (5.3-5.8), it can also be seen that the pressure and vorticity are more accurately captured with the stream function formulation.

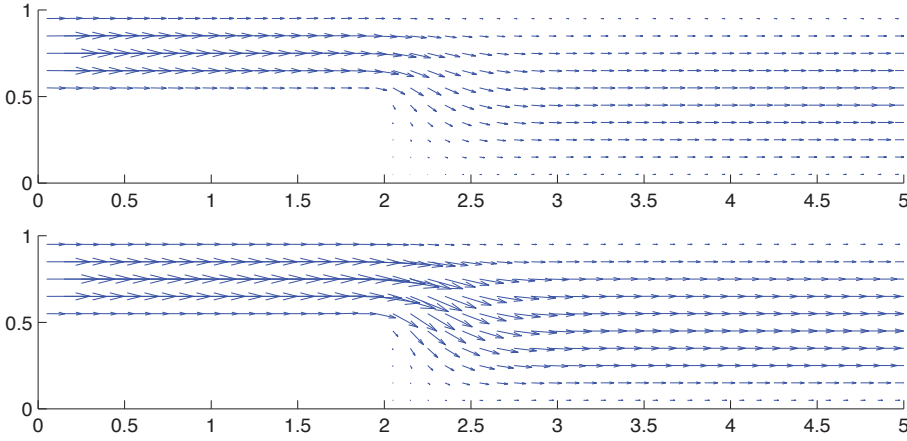


FIG. 5.2. Velocity plot of C^0 (2.19) (top) and SVP (3.8) (bottom) for the backward-facing step.

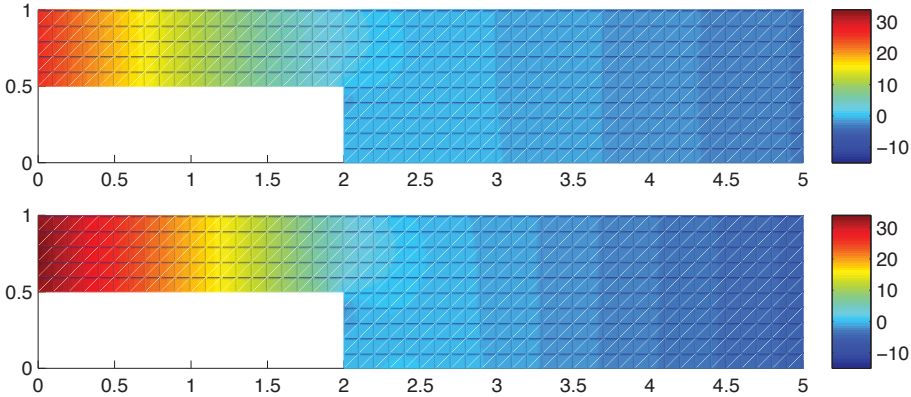


FIG. 5.3. Pressure plot of C^0 (2.19) (top) and SVP (3.8) (bottom) for the backward-facing step.

We considered using divergence free bases on each element, because of the necessity to choose the *correct* weights α_1 and α_2 in (3.5). However, it was not possible to choose a single set of weights that is optimal for all test cases. In the case of (3.8), only one set of weights is used and proved to be effective.

6. Conclusion. In this report we have formulated new discontinuous velocity LSFEMs for the Stokes equations as a means to improve mass conservation. These new methods were compared with a provably optimal norm equivalent weighted L^2 least squares formulation. The immediate discontinuous velocity formulation was found to not be robust as depending on the problem, different weights were required. As a result, a local divergence free basis for the velocity was introduced with the use of a stream function. The stream function approach

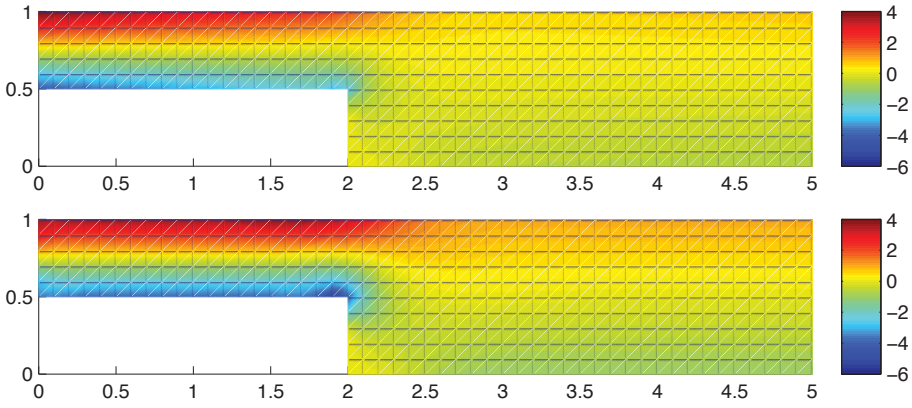


FIG. 5.4. Vorticity plot of C^0 (2.19) (top) and SVP (3.8) (bottom) for the backward-facing step.

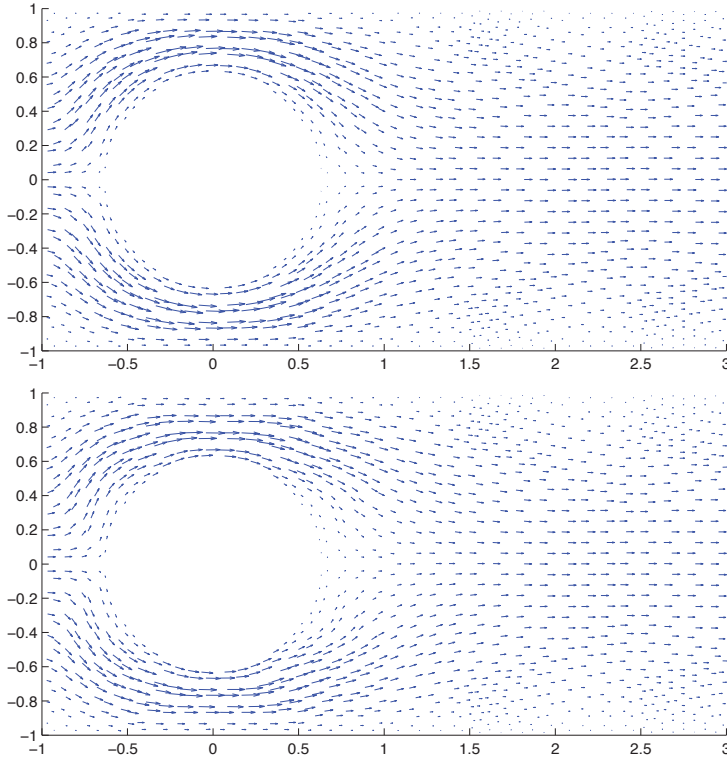


FIG. 5.5. Velocity plot of C^0 (2.19) (top) and SVP (3.8) (bottom) for the cylinder channel.

proved to be robust as only one set of weights, derived from Sobolev theory, allowed the resulting solution to be almost entirely mass conservative.

The proposed approach is very flexible and can be easily applied to other LSFEMs based on the VVP or other first-order Stokes systems. For example, it is trivial to extend (3.8) to a discrete negative-norm method, or to a method which uses velocity gradient, velocity and pressure as dependent variables.

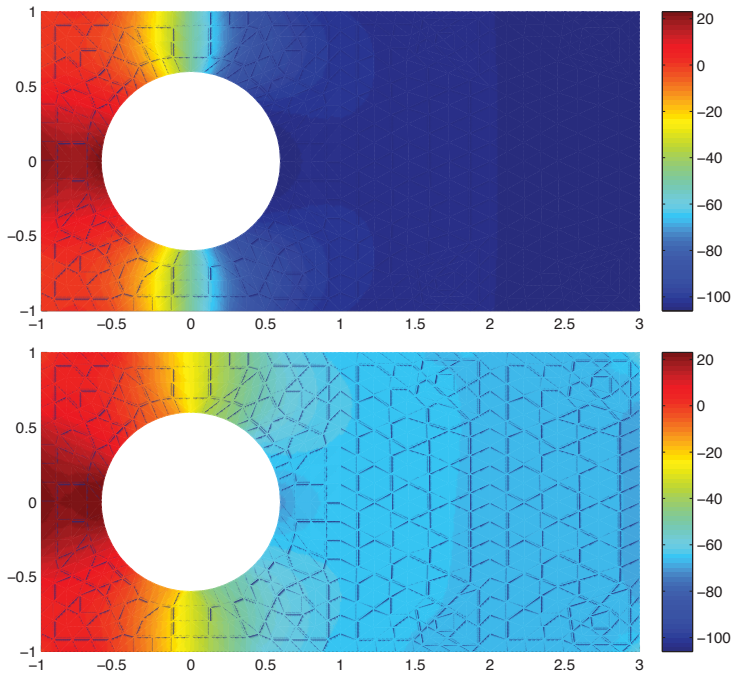


FIG. 5.6. Pressure plot of C^0 (2.19) (top) and SVP (3.8) (bottom) for the cylinder channel.

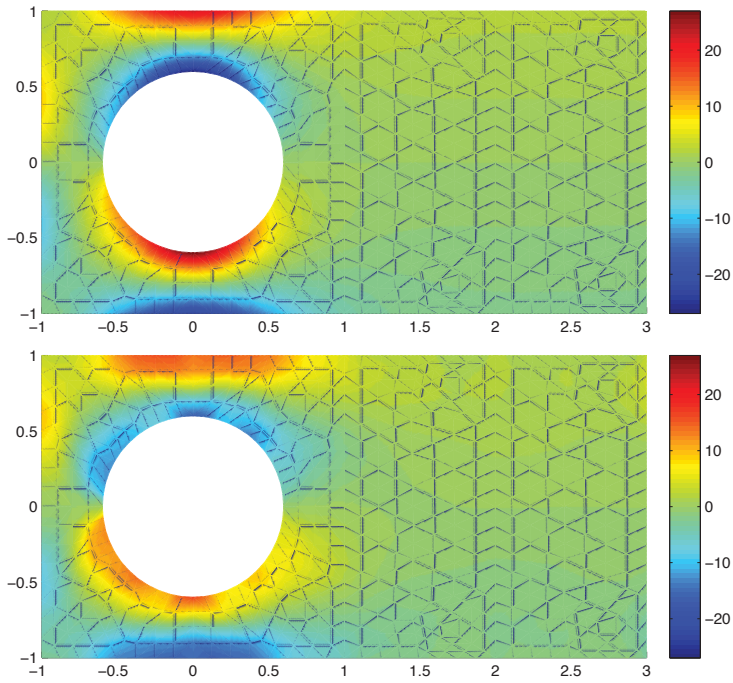


FIG. 5.7. Vorticity plot of C^0 (2.19) (top) and SVP (3.8) (bottom) for the cylinder channel.

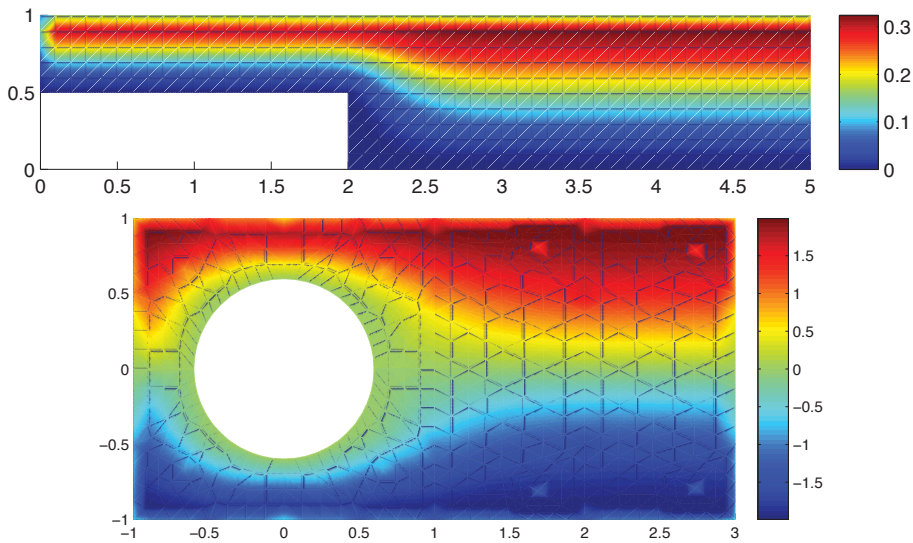


FIG. 5.8. Stream function for backward step (top) and cylinder channel (bottom).

Future work in the area includes theoretical studies of the well-posedness of discontinuous formulations and an implementation of (3.8) using cubic elements for the stream function. This allows the velocity, the curl of the stream function, to be quadratic—satisfying the minimal degree requirement of the parent C^0 least-squares formulation.

REFERENCES

- [1] P. BOCHEV AND D. DAY, *Analysis and computation of a least-squares method for consistent mesh tying*, J. Comp. Appl. Math, 218 (2008), pp. 21–33.
- [2] P. BOCHEV AND M. GUNZBURGER, *Analysis of least-squares finite element methods for the Stokes equations*, Math. Comp., 63 (1994), pp. 479–506.
- [3] ———, *Least-squares finite element methods*, Springer, 2009.
- [4] ———, *A locally conservative mimetic least-squares finite element method for the Stokes equations*, in Proceedings of LSSC 2009, I. Lirkov, S. Margenov, and J. Wasniewski, eds., vol. 5910 of Springer Lecture Notes in Computer Science, 2009, pp. 637–644.
- [5] P. BOCHEV, D. RIDZAL, AND K. PETERSON, *Intrepid: Interoperable Tools For Compatible Discretizations*. <http://trilinos.sandia.gov/packages/intrepid/>, 2010.
- [6] P. BOLTON AND R. THATCHER, *On mass conservation in least-squares methods*, J. Comput. Phys., 203 (2005), pp. 287–304.
- [7] Z. CAI AND B. SHIN, *The discrete first-order system least squares: the second-order elliptic boundary value problem*, SIAM J. Numer. Anal., (2002), pp. 307–318.
- [8] Y. CAO AND M. GUNZBURGER, *Least-squares finite element approximations to solutions of interface problems*, SIAM J. Numer. Anal., 35 (1998), pp. 393–405.
- [9] C. CHANG AND J. NELSON, *Least-squares finite element method for the Stokes problem with zero residual of mass conservation*, SIAM J. Numer. Anal., 34 (1997), pp. 480–489.
- [10] J. DEANG AND M. GUNZBURGER, *Issues related to least-squares finite element methods for the Stokes problem*, SIAM J. Numer. Anal., 35 (1998), pp. 878–906.
- [11] U. GHIA, K. GHIA, AND C. SHIN, *High-re solutions to incompressible flow using the Navier-Stokes equations and a multigrid method*, J. Comput. Phys., 48 (1982), pp. 387–411.
- [12] M. HEROUX, R. BARTLETT, V. HOWLE, R. HOEKSTRA, J. HU, T. KOLDA, R. LEHOUCQ, K. LONG, R. PAWLOWSKI, E. PHIPPS, A. SALINGER, H. THORNIQST, R. TUMINARO, J. WILLENBRING, AND A. WILLIAMS, *An Overview of Trilinos*, Tech. Rep. SAND2003-2927, Sandia National Laboratories, 2003.
- [13] J. HEYS, E. LEE, T. MANTEUFFEL, AND S. MCCORMICK, *On mass-conserving least-squares methods*, SIAM J. Sci. Comput, 28 (2006), pp. 1675–1693.

- [14] ———, *An alternative least-squares formulation of the Navier-Stokes equations with improved mass conservation*, J. Comput. Phys., 226 (2008), pp. 994–1006.
- [15] J. HEYS, E. LEE, T. MANTEUFFEL, S. MCCORMICK, AND J. RUGE, *Enhanced mass conservation in least-squares methods for Navier-Stokes equations*, SIAM J. Sci. Comput., 31 (2009), pp. 2303–2321.
- [16] J. PONTAZA AND J. REDDY, *Space-time coupled spectral/hp least-squares finite element formulations for the incompressible Navier-Stokes equations*, J. Comput. Phys., 197 (2004), pp. 418–459.
- [17] M. SALA, K. STANLEY, AND M. HEROUX, *Amesos: A set of general interfaces to sparse direct solver libraries*, in Proceedings of PARA'06 Conference, Umea, Sweden, 2006.

APPLICATION OF A DISCONTINUOUS PETROV-GALERKIN METHOD TO THE STOKES EQUATIONS

NATHAN V. ROBERTS[‡], DENIS RIDZAL[§], PAVEL B. BOCHEV[¶], LESZEK D. DEMKOWICZ^{||}, KARA J. PETERSON^{**}, AND CHRISTOPHER M. SIEFERT^{††}

Abstract. The discontinuous Petrov-Galerkin finite element method proposed by L. Demkowicz and J. Gopalakrishnan [5, 6] guarantees the optimality of the solution in what they call the *energy norm*. An important choice that must be made in the application of the method is the definition of the inner product on the test space. In this paper, we apply the DPG method to the Stokes problem in two dimensions, analyzing it to determine appropriate inner products, and perform a series of numerical experiments.

1. Introduction. Recently, L. Demkowicz and J. Gopalakrishnan have proposed a new class of discontinuous Petrov-Galerkin (DPG) methods [5, 6, 7, 10, 3], which compute test functions that are adapted to the problem of interest to produce stable discretization schemes. An important choice that must be made in the application of the method is the definition of the inner product on the test space. In this paper, we apply the method to the Stokes problem in two dimensions, analyzing it to determine appropriate inner products, and perform numerical experiments to test these inner products.

Whereas traditional Galerkin methods use the same space for test and trial spaces, Petrov-Galerkin methods allow the test and trial spaces to differ. The DPG approach computes test functions that are *optimal*, in a sense that we make precise in Section 2. One consequence of this choice of test functions is that the stiffness matrix for a continuous, weakly coercive variational formulation is symmetric (hermitian, for complex-valued problems) and positive definite. Of course, the determination of test functions is an extra step compared with traditional methods; it is important that these can be determined cheaply. By using discontinuous Galerkin (DG) formulations, DPG achieves this, reducing the computation of the test functions to a local problem. Our method bears some resemblance to the MDG method [9] in that a local problem is solved on each element. The key difference with that paper is that in MDG the local problem is restriction of the original equations whereas in DPG the local problem is implied by the selected test space inner product. Furthermore, in MDG the local problem is used to express DG degrees of freedom in terms of continuous degrees of freedom, i.e., to effect static condensation on the element.

Our primary goal is the application of the method to the Stokes problem in two dimensions. The strong form of the problem is

$$-2\mu\nabla \cdot \underline{\epsilon} + \nabla p = \mathbf{f} \quad \text{in } \Omega, \quad (1.1)$$

$$\nabla \cdot \mathbf{u} = 0 \quad \text{in } \Omega, \quad (1.2)$$

$$\mathbf{u} = \mathbf{g}_D \quad \text{on } \partial\Omega, \quad (1.3)$$

where $\Omega \subset \mathbb{R}^2$, μ is viscosity, $\underline{\epsilon} = \nabla^{\text{sym}} \mathbf{u}$ is strain, p is pressure, \mathbf{u} velocity, and \mathbf{f} a vector forcing function.

The paper is structured as follows. In Section 2, we give an introduction to the basic features of the DPG method. In Section 3, we derive the weak formulation of the problem.

[‡]The University of Texas at Austin, nroberts@ices.utexas.edu

[§]Sandia National Laboratories, dridzal@sandia.gov

[¶]Sandia National Laboratories, pbboche@sandia.gov

^{||}The University of Texas at Austin, leszek@ices.utexas.edu

^{**}Sandia National Laboratories, kjpeter@sandia.gov

^{††}Sandia National Laboratories, csiefer@sandia.gov

In Section 4, we motivate the choice of inner product on the test space with reference to an argument for the continuity of the bilinear form. In Section 5, we present the numerical results. We conclude in Section 6.

2. DPG Method. Here, we sketch some of the main features of the DPG method. For details, we refer the reader to a series of papers by Demkowicz et al., in particular the second ICES Report [6], from which most of this section is derived. We begin with theoretical definitions and results, and then describe the approach to practical realization. Consider the abstract variational boundary-value problem:

$$\text{Find } u \in U : b(u, v) = l(v) \quad \forall v \in V. \quad (2.1)$$

We take U and V to be real Hilbert spaces. We assume $b(\cdot, \cdot)$ is continuous, i.e.

$$|b(u, v)| \leq M \|u\|_U \|v\|_V, \quad (2.2)$$

for some real M . We assume also that $b(\cdot, \cdot)$ is weakly coercive, that is

$$\inf_{\|u\|_U=1} \sup_{\|v\|_V=1} b(u, v) > \gamma, \quad (2.3)$$

for some $\gamma > 0$. If we additionally assume that

$$\{v \in V : b(u, v) = 0 \quad \forall u \in U\} = \{0\}, \quad (2.4)$$

then it is well known that the problem (2.1) has a unique solution provided that $l \in V'$, the dual of V .

2.1. Energy Norm. We define an alternate norm, called the *energy norm*, on the trial space U by

$$\|u\|_E \stackrel{\text{def}}{=} \sup_{\|v\|_V=1} b(u, v). \quad (2.5)$$

This norm is the one in which the optimality is guaranteed by the selection of optimal test functions. It is an equivalent norm to the standard norm on U , i.e.

$$\gamma \|u\|_U \leq \|u\|_E \leq M \|u\|_U \quad \forall u \in U. \quad (2.6)$$

2.2. Optimal Test Functions. We are now prepared to give a definition of the optimal test functions. Define a map $T : U \rightarrow V$ from the trial space to the test space by: For $u \in U$, define Tu , the *optimal test function* corresponding to u , as the unique solution to

$$(Tu, v)_V = b(u, v) \quad \forall v \in V.$$

By the Riesz representation theorem, T is well-defined. Note that

$$\|u\|_E = \sup_{\|v\|_V=1} b(u, v) = \sup_{\|v\|_V=1} (Tu, v)_V = \frac{1}{\|Tu\|_V} (Tu, Tu)_V = \|Tu\|_V.$$

Thus the energy norm is generated by the inner product on V , i.e.

$$(u, u)_E \stackrel{\text{def}}{=} (Tu, Tu)_V. \quad (2.7)$$

In practice, we approximate T by a discrete operator T_n , described in Section 2.4.

2.3. Optimal Test Space for U_n . Take a finite-dimensional trial space $U_n \subset U$. Define the *optimal test space* for U_n as $V_n = \text{span}\{Te_j : j = 1, \dots, n\}$, where the e_j form a basis for U_n .

Solve the discrete problem

$$\text{Find } u_n \in U_n : b(u_n, v) = l(v) \quad \forall v \in V_n. \quad (2.8)$$

Then the error is the best approximation error in the energy norm,

$$\|u - u_n\|_E = \inf_{w_n \in U_n} \|u - w_n\|_E, \quad (2.9)$$

and this is the sense in which the test space is *optimal*.

2.4. Practical Realization. The method involves two steps: first, find the optimal test functions; second, use the optimal test functions to solve the discrete problem 2.8. The optimal test functions are not in general polynomials. In practice, we approximate them with an “enriched” polynomial space — a space of polynomials of slightly higher degree than the trial space. This is done to provide a higher-fidelity approximation to the continuous space of optimal test functions. The best choice for the amount of “enrichment” is determined experimentally for each problem.

In general, we apply the following procedure:

1. Given a boundary value problem, develop mesh-dependent $b(\cdot, \cdot)$ with test space V that allows inter-element discontinuities (hence *Discontinuous* Petrov-Galerkin). We develop this in Section 3.
2. Choose trial space U_n (in particular the norm of interest in U_n), and the inner product on V , which will be motivated by the choice of trial space. We detail this process for the Stokes problem in Section 4.
3. Compute optimal test functions. Approximate T by $T_n : U_n \rightarrow \tilde{V}_n \subset V$. We use an enriched space of piecewise polynomials for \tilde{V}_n . Defining $t_j = T_n e_j$, we solve

$$(t_j, \tilde{e}_i)_V = b(e_j, \tilde{e}_i)$$

for t_j , where the \tilde{e}_i form the basis for \tilde{V}_n .

4. Use the optimal test functions to solve the problem on $U_n \times \tilde{V}_n$. We note that the stiffness matrix here is symmetric positive definite (hermitian, for a complex-valued problem),

$$\begin{aligned} b(e_j, t_i) &= (T_n e_j, t_i)_V = (T_n e_j, T_n e_i)_V = \overline{(T_n e_i, T_n e_j)}_V \\ &= \overline{(T_n e_i, t_j)}_V = \overline{b(e_i, t_j)}. \end{aligned}$$

Also, note that this means that we may compute the stiffness matrix in terms of the inner product on the test space V , without explicit recourse to the bilinear form.

3. Stokes Formulation. Our general approach to variational formulation in DPG is as follows. First, rewrite the strong form of the problem as a system of first-order partial differential equations. Then, multiply by test functions and integrate by parts, moving all derivatives to the test functions. We thus arrive at the *ultra-weak* form of the problem, a formulation in which all solution variables are in L^2 .

Starting with the strong formulation defined in equations (1.1)-(1.3), introduce stress σ and vorticity ω by

$$\begin{aligned} \underline{\sigma} &= 2\mu \underline{\epsilon} - p \underline{I} \\ \underline{\omega} &= \frac{1}{2}(\nabla \mathbf{u} - \nabla \mathbf{u}^T) \end{aligned}$$

so that equation (1.1) becomes simply $-\nabla \cdot \underline{\sigma} = f$. We also have

$$\underline{\epsilon} = \frac{1}{2\mu}(\underline{\sigma} + p\mathbf{I}).$$

Since $\underline{\epsilon} = \nabla^{\text{sym}} \mathbf{u} = \nabla \mathbf{u} - \underline{\omega}$, the entire system is

$$\begin{aligned} \frac{1}{2\mu}(\underline{\sigma} + p\mathbf{I}) - \nabla \mathbf{u} + \underline{\omega} &= \mathbf{0} && \text{in } \Omega, \\ -\nabla \cdot \underline{\sigma} &= f && \text{in } \Omega, \\ \nabla \cdot \mathbf{u} &= 0 && \text{in } \Omega, \\ \mathbf{u} &= g_D && \text{on } \partial\Omega. \end{aligned}$$

Note that the antisymmetric part of the first equation recovers the definition of $\underline{\omega}$, so that it need not enter the system separately. Define scalar $\omega = \omega_{21} = \frac{1}{2}(u_{1,2} - u_{2,1})$. Our strong formulation is

$$\begin{aligned} \frac{1}{2\mu} \begin{pmatrix} \sigma_{11} + p \\ \sigma_{21} \end{pmatrix} - \nabla u_1 + \begin{pmatrix} 0 \\ \omega \end{pmatrix} &= \mathbf{0} && \text{in } \Omega, \\ \frac{1}{2\mu} \begin{pmatrix} \sigma_{12} \\ \sigma_{22} + p \end{pmatrix} - \nabla u_2 - \begin{pmatrix} \omega \\ 0 \end{pmatrix} &= \mathbf{0} && \text{in } \Omega, \\ -\nabla \cdot \begin{pmatrix} \sigma_{11} \\ \sigma_{21} \end{pmatrix} &= f_1 && \text{in } \Omega, \\ -\nabla \cdot \begin{pmatrix} \sigma_{12} \\ \sigma_{22} \end{pmatrix} &= f_2 && \text{in } \Omega, \\ \nabla \cdot \mathbf{u} &= 0 && \text{in } \Omega, \\ \mathbf{u} &= g_D && \text{on } \partial\Omega. \end{aligned}$$

Multiplying the first two equations by vector test functions \mathbf{q}_i and the following three by scalar test functions v_i , and integrating by parts over an element K , we obtain

$$\begin{aligned} \int_K \left(\frac{1}{2\mu} \begin{pmatrix} \sigma_{11} + p \\ \sigma_{21} \end{pmatrix} + \begin{pmatrix} 0 \\ \omega \end{pmatrix} \right) \cdot \mathbf{q}_1 + \int_K u_1 \nabla \cdot \mathbf{q}_1 &- \int_{\partial K} \widehat{u}_1 \mathbf{q}_1 \cdot \boldsymbol{\nu} &= \mathbf{0} \\ \int_K \left(\frac{1}{2\mu} \begin{pmatrix} \sigma_{12} \\ \sigma_{22} + p \end{pmatrix} - \begin{pmatrix} \omega \\ 0 \end{pmatrix} \right) \cdot \mathbf{q}_2 + \int_K u_2 \nabla \cdot \mathbf{q}_2 &- \int_{\partial K} \widehat{u}_2 \mathbf{q}_2 \cdot \boldsymbol{\nu} &= \mathbf{0} \\ \int_K \begin{pmatrix} \sigma_{11} \\ \sigma_{21} \end{pmatrix} \cdot \nabla v_1 &- \int_{\partial K} \widehat{\sigma}_1 v_1 \cdot \boldsymbol{\nu} &= \int_K f_1 v_1 \\ \int_K \begin{pmatrix} \sigma_{12} \\ \sigma_{22} \end{pmatrix} \cdot \nabla v_2 &- \int_{\partial K} \widehat{\sigma}_2 v_2 \cdot \boldsymbol{\nu} &= \int_K f_2 v_2 \\ - \int_K \mathbf{u} \cdot \nabla v_3 &+ \int_{\partial K} \widehat{\mathbf{u}} v_3 \cdot \boldsymbol{\nu} &= 0, \end{aligned}$$

where the “hatted” variables (\widehat{u}_1 , e.g.) are the *fluxes* introduced by relaxing the continuity requirement at element boundaries. These differ from the *numerical fluxes* that appear in other DG methods, in that they are not constructed a priori, but simply enter the variational problem as additional unknowns. We solve for them at the same time as we solve the rest of the unknowns. As in other DG methods, the fluxes will approach the corresponding “unhatted” solution variables as the latter approach the exact solution.

4. Inner Product Determination. As discussed in Section 2, the optimality proof depends on the continuity and weak coercivity of the bilinear form. In this section, we use continuity to motivate a particular choice of inner product on V .

We seek to show that $|b(U, v)| \leq M \|U\|_U \|v\|_V$, for some constant M , for spaces U and V to be specified. The norm on U should be specified in such a way that minimizing the error in this norm will produce the results we want. We define

$$\|U\|_U^2 = \sum_{i=1}^7 \left(\frac{\|u_i\|_{L_2(\Omega)}}{\alpha_i} \right)^2 + \sum_{i=1}^2 \left(\frac{\|\widehat{F}_i\|_{H^{1/2}(\partial\Omega)}}{\widehat{\alpha}_i} \right)^2 + \sum_{i=3}^5 \left(\frac{\|\widehat{F}_i\|_{H^{-1/2}(\partial\Omega)}}{\widehat{\alpha}_i} \right)^2,$$

where u_1 and u_2 are as above, $u_3 = \sigma_{11}$, $u_4 = \sigma_{12} = \sigma_{21}$, $u_5 = \sigma_{22}$, $u_6 = \omega$, and $u_7 = p$, \widehat{F}_i is the flux corresponding to the i th equation (that is, $\widehat{F}_1 = \widehat{u}_1$, $\widehat{F}_2 = \widehat{u}_2$, $\widehat{F}_3 = \widehat{\sigma}_1 \cdot \nu$, $\widehat{F}_4 = \widehat{\sigma}_2 \cdot \nu$, and $\widehat{F}_5 = \widehat{u} \cdot \nu$), and the α_i and $\widehat{\alpha}_i$ are positive weights that allow us to emphasize specific components. The reason we use the $H^{1/2}$ norm on the fluxes corresponding to $H(\text{div})$ test functions is that $\mathbf{q} \in H(\text{div}) \implies \text{tr}(\mathbf{q}) \in H^{-1/2}$. Thus for $\int_{\partial\Omega} \widehat{F}_i \mathbf{q}_i \cdot \nu$ to make sense mathematically, we require $\widehat{F}_i \in H^{1/2}$. A similar argument establishes that the fluxes corresponding to H^1 test functions should lie in $H^{-1/2}$. Let us consider the first equation of our bilinear form,

$$\begin{aligned} b_1(U, v) &= \int_{\Omega} \left(\frac{\sigma_{11} + p}{2\mu} \right) \cdot \mathbf{q}_1 + \int_{\Omega} u_1 \nabla \cdot \mathbf{q}_1 - \int_{\partial\Omega} \widehat{u}_1 \mathbf{q}_1 \cdot \nu \\ &= \left(\frac{\sigma_{11} + p}{2\mu}, q_{11} \right)_{\Omega} + \left(\frac{\sigma_{21}}{2\mu} + \omega, q_{12} \right)_{\Omega} + (u_1, \nabla \cdot \mathbf{q}_1)_{\Omega} - (\widehat{u}_1, \mathbf{q}_1 \cdot \nu)_{\partial\Omega} \end{aligned}$$

Now, by the Cauchy-Schwarz inequality, we have

$$\begin{aligned} |b_1(U, v)| &\leq \frac{1}{2\mu} (\|\sigma_{11}\|_0 + \|p\|_0) \|q_{11}\|_0 + \left(\frac{1}{2\mu} \|\sigma_{21}\|_0 + \|\omega\|_0 \right) \|q_{12}\|_0 \\ &\quad + \|u_1\|_0 \|\nabla \cdot \mathbf{q}_1\|_0 + \|\widehat{u}_1\|_{H^{1/2}(\partial\Omega)} \|\mathbf{q}_1 \cdot \nu\|_{H^{-1/2}(\partial\Omega)} \end{aligned} \quad (4.1)$$

Applying the finite-dimensional Cauchy-Schwarz inequality, we have

$$\begin{aligned} |b_1(U, v)| &\leq \left(\left(\frac{\|\sigma_{11}\|_0}{2\mu} \right)^2 + \left(\frac{\|p\|_0}{2\mu} \right)^2 + \left(\frac{\|\sigma_{21}\|_0}{2\mu} \right)^2 + \|\omega\|_0^2 + \|u_1\|_0^2 + \|\widehat{u}_1\|_0^2 \right)^{1/2} \\ &\quad \cdot \left(\|q_{11}\|_0^2 + \|q_{12}\|_0^2 + \|\nabla \cdot \mathbf{q}_1\|_0^2 + \|\mathbf{q}_1 \cdot \nu\|_{H^{-1/2}(\partial\Omega)}^2 \right)^{1/2} \end{aligned}$$

Note that for a particular choice of weights, namely $\alpha_3 = \alpha_4 = \alpha_7 = \frac{1}{2\mu}$, $\alpha_6 = \alpha_1 = \widehat{\alpha}_1 = 1$, we then immediately have

$$|b_1(U, v)| \leq \|U\|_U \left(\|q_{11}\|_0^2 + \|q_{12}\|_0^2 + \|\nabla \cdot \mathbf{q}_1\|_0^2 + \|\mathbf{q}_1 \cdot \nu\|_{H^{-1/2}(\partial\Omega)}^2 \right)^{1/2}$$

motivating a norm

$$\|\mathbf{q}_1\|_{V_1} = \left(\|q_{11}\|_0^2 + \|q_{12}\|_0^2 + \|\nabla \cdot \mathbf{q}_1\|_0^2 + \|\mathbf{q}_1 \cdot \nu\|_{H^{-1/2}(\partial\Omega)}^2 \right)^{1/2}.$$

However, one of our purposes in defining a weighted norm $\|U\|$ was to gain some control over scale equivalence in computation of the test space inner product, and the argument above in providing a tight bound has separated the weights from the test space terms. Instead, let us

return to the inequality (4.1), and note that by the definition of $\|\cdot\|_U$, $\|u_i\|_{L_2(\Omega)} \leq \alpha_i \|U\|_U$; similarly, $\|\widehat{F}_i\|_{H^{1/2}(\Omega)} \leq \widehat{\alpha}_i \|U\|_U$ for $i = 1, 2$ and $\|\widehat{F}_i\|_{H^{-1/2}(\Omega)} \leq \widehat{\alpha}_i \|U\|_U$ for $i = 3, 4, 5$. Thus we have

$$\begin{aligned} |b_1(U, v)| &\leq \left(\frac{\alpha_3 + \alpha_7}{2\mu} \|q_{11}\|_0 + \left(\frac{\alpha_4}{2\mu} + \alpha_6 \right) \|q_{11}\|_0 \right. \\ &\quad \left. + \alpha_1 \|\nabla \cdot \mathbf{q}_1\|_0 + \widehat{\alpha}_1 \|\mathbf{q}_1 \cdot \nu\|_{H^{1/2}(\partial\Omega)} \right) \|U\|_U, \end{aligned}$$

motivating the norm

$$\|\mathbf{q}_1\|_{V_1} = \frac{\alpha_3 + \alpha_7}{2\mu} \|q_{11}\|_0 + \left(\frac{\alpha_4}{2\mu} + \alpha_6 \right) \|q_{12}\|_0 + \alpha_1 \|\nabla \cdot \mathbf{q}_1\|_0 + \widehat{\alpha}_1 \|\mathbf{q}_1 \cdot \nu\|_{H^{1/2}(\partial\Omega)}.$$

For convenience, we implement a similar norm given by

$$\begin{aligned} \|\mathbf{q}_1\|_{V_1}^2 &\stackrel{\text{def}}{=} \left(\frac{\alpha_3 + \alpha_7}{2\mu} \right)^2 \|q_{11}\|_0^2 + \left(\frac{\alpha_4}{2\mu} + \alpha_6 \right)^2 \|q_{12}\|_0^2 + \alpha_1^2 \|\nabla \cdot \mathbf{q}_1\|_0^2 \\ &\quad + \widehat{\alpha}_1^2 \|\mathbf{q}_1 \cdot \nu\|_{L_2(\Omega)}^2. \end{aligned}$$

Similarly, for \mathbf{q}_2 we implement

$$\begin{aligned} \|\mathbf{q}_2\|_{V_2}^2 &\stackrel{\text{def}}{=} \left(\frac{\alpha_4}{2\mu} + \alpha_6 \right)^2 \|q_{21}\|_0^2 + \left(\frac{\alpha_5}{2\mu} + \alpha_7 \right)^2 \|q_{22}\|_0^2 + \alpha_2^2 \|\nabla \cdot \mathbf{q}_2\|_0^2 \\ &\quad + \widehat{\alpha}_2^2 \|\mathbf{q}_2 \cdot \nu\|_{L_2(\Omega)}^2. \end{aligned}$$

For $b_3(U, v)$, we have

$$\begin{aligned} b_3(U, v) &= \int_{\Omega} \boldsymbol{\sigma}_1 \cdot \nabla v_1 - \int_{\partial\Omega} \widehat{\boldsymbol{\sigma}}_1 v_1 \cdot \nu \\ &= \int_{\Omega} \begin{pmatrix} u_3 \\ u_4 \end{pmatrix} \cdot \nabla v_1 - \int_{\partial\Omega} \widehat{F}_3 v_1. \end{aligned}$$

Thus

$$|b_3(U, v)| \leq \|U\| \left(\left\| \begin{pmatrix} \alpha_3 \\ \alpha_4 \end{pmatrix} \cdot \nabla v_1 \right\|_0 + \widehat{\alpha}_3 \|v_1\|_{H^{1/2}(\partial\Omega)} \right),$$

and the norm we implement for v_1 is

$$\|v_1\|_{V_3}^2 \stackrel{\text{def}}{=} \alpha_3^2 \left\| \frac{\partial v_1}{\partial x} \right\|_0^2 + \alpha_4^2 \left\| \frac{\partial v_1}{\partial y} \right\|_0^2 + \widehat{\alpha}_3^2 \|v_1\|_0^2.$$

Similarly,

$$\|v_2\|_{V_4}^2 \stackrel{\text{def}}{=} \alpha_4^2 \left\| \frac{\partial v_2}{\partial x} \right\|_0^2 + \alpha_5^2 \left\| \frac{\partial v_2}{\partial y} \right\|_0^2 + \widehat{\alpha}_4^2 \|v_2\|_0^2$$

and

$$\|v_3\|_{V_5}^2 \stackrel{\text{def}}{=} \alpha_1^2 \left\| \frac{\partial v_3}{\partial x} \right\|_0^2 + \alpha_2^2 \left\| \frac{\partial v_3}{\partial y} \right\|_0^2 + \widehat{\alpha}_5^2 \|v_3\|_0^2.$$

Now, we can define a general norm on the test space by

$$\begin{aligned}
 \|(\mathbf{q}_1, \dots, \mathbf{q}_L; v_1, \dots, v_M)\|^2 = & \int_{\Omega} \left(\sum_{i=1}^L (a_i \nabla \cdot \mathbf{q}_i)^2 + \sum_{i=1}^L ((b_{i1} q_{i1})^2 + (b_{i2} q_{i2})^2) \right. \\
 & + \sum_{i=1}^M \left(\left(c_{i1} \frac{\partial v_i}{\partial x} \right)^2 + \left(c_{i2} \frac{\partial v_i}{\partial y} \right)^2 \right) + \sum_{i=1}^M (d_i v_i)^2 \Big) \\
 & + \int_{\partial\Omega} \left(\sum_{i=1}^L (e_i \mathbf{q}_i \cdot \nu)^2 + \sum_{i=1}^M (f_i v_i)^2 \right). \tag{4.2}
 \end{aligned}$$

Based on the analysis above, we require

$$\begin{aligned}
 a_1 &= \alpha_1, & a_2 &= \alpha_2 \\
 b_{11} &= \frac{\alpha_3 + \alpha_7}{2\mu}, & b_{12} &= \frac{\alpha_4}{2\mu} + \alpha_6 \\
 b_{21} &= \frac{\alpha_4}{2\mu} + \alpha_6, & b_{22} &= \frac{\alpha_5 + \alpha_7}{2\mu} \\
 c_{11} &= \alpha_3, & c_{12} &= \alpha_4 \\
 c_{21} &= \alpha_4, & c_{22} &= \alpha_5 \\
 c_{31} &= \alpha_1, & c_{32} &= \alpha_2 \\
 d_i &= 0 \\
 e_1 &= \widehat{\alpha}_1, e_2 = \widehat{\alpha}_2 & f_1 &= \widehat{\alpha}_3, f_2 = \widehat{\alpha}_4, f_3 = \widehat{\alpha}_5.
 \end{aligned} \tag{4.3}$$

4.1. Choice of α values. How do we determine appropriate weights α_i and $\widehat{\alpha}_i$ for the norm of U ? Our choice is motivated by considerations of *norm equivalence* arising, for instance, in the least-squares finite element literature, see [1, Sec. 4.5]. For simplicity, we apply a similar guideline, which we call *scale equivalence*. Let us consider a mesh with elements of size h . In a least-squares method, one would motivate the choice of weights by examining the factors of h entering the stiffness matrix through derivatives in the bilinear form. One would then select weights so that each term of the bilinear form had the same h -factor, thereby ensuring that no single term dominates the least-squares functional as $h \rightarrow 0$.

Recall that in DPG the optimality is expressed in terms of the energy norm in equation (2.9), which in turn is defined by the inner product on V in equation (2.7). As in least-squares methods, there is an underlying optimization principle (equations (2.9) and (2.5)), and thus it makes sense to have all terms in the test space inner product equally weighted in the discrete setting. In this section, therefore, we aim to determine weights α_i and $\widehat{\alpha}_i$ that will allow this.

Computing the optimal test functions involves the solution of a problem of the form

$$(t_j, \tilde{e}_i)_V = b(e_j, \tilde{e}_i),$$

where the \tilde{e}_i form the basis for the enriched polynomial space \widetilde{V}_n used to represent the test functions, and t_j is the optimal test function corresponding to $e_j \in U$. Thus the matrix for determining the optimal test functions is generated by computing inner products $(\tilde{e}_k, \tilde{e}_i)_V$. The goal is to keep the summands entering this matrix of the same order of magnitude in h .

We assume a partition of Ω into quadrilateral elements. Since the various components (e.g. \mathbf{q}_1 and \mathbf{q}_2) of the test function do not interact, we can examine each separately. Suppose

that the element has dimensions (h_1, h_2) and $\mathbf{q}_1 = \begin{pmatrix} xy \\ xy \end{pmatrix}$. Then

$$\begin{aligned} (\mathbf{q}_1, \mathbf{q}_1)_V &= \int_K \left(a_1^2 (y^2 + 2xy + x^2) + (b_{11}^2 + b_{12}^2) x^2 y^2 \right) + \int_{\partial K} e_1^2 \left(\begin{pmatrix} xy \\ xy \end{pmatrix} \cdot \nu \right)^2 \\ &= O\left(a_1^2 (h_1 h_2^3 + h_1^2 h_2^2 + h_1^3 h_2) + (b_{11}^2 + b_{12}^2) h_1^3 h_2^3 + e_1^2 (h_1^2 h_2^3 + h_1^3 h_2^2)\right) \end{aligned}$$

Clearly, no choice of weights will make all summands of the same order in both h_1 and h_2 ; the best we can do is to make the h_1 and h_2 orders of each summand differ by no more than 2, and make the sum of the h_1 and h_2 orders the same across all summands. This can be accomplished by setting $a_1^2 = h_1 h_2$, $b_{11}^2 = b_{12}^2 = 1$, and $e_1^2 = \sqrt{h_1 h_2}$.

The computation with \mathbf{q}_2 is identical. Now, consider $v_1 = xy$. We have

$$\begin{aligned} (v_1, v_1)_V &= \int_K \left(c_{11}^2 y^2 + c_{12}^2 x^2 + d_1^2 x^2 y^2 \right) + \int_{\partial K} f_1^2 x^2 y^2 \\ &= O\left(c_{11}^2 h_1 h_2^3 + c_{12}^2 h_1^3 h_2 + (b_{11}^2 + d_1^2) h_1^3 h_2^3 + f_1^2 (h_1^2 h_2^3 + h_1^3 h_2^2)\right) \end{aligned}$$

Again, we cannot choose the coefficients to get each summand to have the same order in both h_1 and h_2 , but here at least the c_{11} and c_{12} coefficients can be chosen so that their respective terms match precisely. We would like to have $c_{11}^2 = h_1^2$, $c_{12}^2 = h_2^2$, $d_1^2 = 1$, and $f_1^2 = \sqrt{h_1 h_2}$, and similarly for v_2 , we'd like $c_{21}^2 = h_1^2$, etc. This cannot be fully achieved because of the way the α_i enter the inner product; specifically, $\alpha_4 = c_{12} = c_{21}$. Instead, we arrive at the following weights:

$$\alpha_1 = \alpha_2 = \sqrt{h_1 h_2} \quad (4.4)$$

$$\alpha_3 = h_1 \quad (4.5)$$

$$\alpha_4 = \sqrt{h_1 h_2} \quad (4.6)$$

$$\alpha_5 = h_2 \quad (4.7)$$

$$\alpha_6 = \alpha_7 = 1 \quad (4.8)$$

$$\widehat{\alpha}_i = \sqrt{\sqrt{h_1 h_2}}. \quad (4.9)$$

We detail numerical results for this inner product in Section 5.3. In Section 5.4, we present a version where $\alpha_3 = \alpha_4 = \alpha_5 = 1$, with very similar results.

5. Numerical Results. We solve the Stokes problem on the domain $(-1, 1) \times (-1, 1)$, with $\mu = 1$. We follow the choice of manufactured solution employed in a paper by Cockburn et al. [4], in which they apply the LDG method to Stokes. We compare our convergence rates to theirs; the L^2 error measurements are not strictly comparable because they employ a triangular mesh, whereas we use a quadrilateral mesh. As stated in Section 2.4, the space for V is an “enriched” polynomial space. The numerical results presented below were produced with test functions of degree one higher than that of the trial space.

Although the differences between our meshes and those in Cockburn et al. mean that our error measurements are not strictly comparable, we still would expect to attain similar *rates* of convergence, and for the L^2 error values in each component to be within an order of magnitude or so. The rates of convergence we would expect in a velocity-stress-pressure (VSP) least-squares context would be $k + 1$ for the velocity components u_1 and u_2 , and k for the pressure p , where k denotes the polynomial degree of the trial space [1, p. 269]. We have yet to carry out the convergence analysis for DPG.

Following Cockburn et al., we use

$$\begin{aligned} u_1 &= -e^x(y \cos y + \sin y) \\ u_2 &= e^x y \sin y \\ p &= 2\mu e^x \sin y \end{aligned}$$

as our manufactured solution. We impose the constraint $p(0, 0) = p_0$ in order to establish the uniqueness of the solution.

We try four inner products, the first two as baselines, and the latter two as suggested by our analysis. As expected, the choice of the inner product makes a great deal of difference to the rate of convergence.

5.1. Generic Inner Product. As a baseline to show the importance of a good inner product on the test function space, the results in this section are produced using a test space inner product unrelated to our analysis. In the general form of the norm specified in equation (4.2), let $a_i = b_{ij} = c_{ij} = d_i = e_i = f_i = 1$.

As can be seen in Table A.1, although our convergence rates generally start out near the asymptotic rates predicted, they fall off quickly. The rate for pressure with quadratic elements is particularly poor. The L^2 error values in \mathbf{u} are perhaps not too bad, within an order of magnitude or so of the LDG results. However, the pressure error values are extremely poor, off by up to three orders of magnitude.

As an experiment, we tried enriching the fluxes, using polynomials of degree $k+1$ to represent the solution fluxes; at the same time, we enriched the test function space further, using polynomials of degree $k+2$. As shown in Table A.2, this uniformly reduces the error, particularly in the pressure, and improves the convergence rate observed in the pressure for quadratic elements. Although cubic elements also saw uniformly reduced error, the convergence rates observed were somewhat worse.

5.2. “All Ones” Inner Product. In Section 4.1, we derived weights for the inner product so as to weight all terms in the determination of the optimal test functions equally. To see the impact of our choice of those weights in relief, we try an inner product in which $\alpha_i = \widehat{\alpha}_j = 1$. Compared with the generic inner product employed in the previous section, this inner product takes account of the continuity argument.

As can be seen in Table B.1, with this choice of inner product, DPG performs slightly better than with the generic inner product, but the rates of convergence in p are quite poor, especially for quadratic elements. For the 64×64 mesh, we even see regression in the p error compared with the 32×32 mesh, suggesting that some terms in the inner product dominate as $h \rightarrow 0$, preventing convergence.

As in Section 5.1, we tried enriching the fluxes, using polynomials of degree $k+1$ to represent the solution fluxes; at the same time, we enriched the test function space further, using polynomials of degree $k+2$. As shown in Table B.2, this uniformly reduces the error, particularly in the pressure, and improves the convergence rate observed in the pressure for quadratic elements. Although cubic elements also saw uniformly reduced error, the convergence rates observed for pressure were somewhat worse.

5.3. Mesh-Dependent Inner Product. In this inner product, we choose the α_i values as derived in Section 4.1 and specified in equations (4.4)-(4.9). There, we aimed to achieve scale equivalence in the determination of the optimal test functions while selecting an inner product that allowed our argument for the continuity of $b(\cdot, \cdot)$ to remain intact.

As can be seen in Table C.1, with this inner product, we have far superior convergence compared with either of the previous two inner products we have considered. Here, the

convergence rates for both velocity and pressure are very close to those predicted by theory, and the L^2 error for \mathbf{u} is within a factor of 3 of the LDG results. However, our L^2 error in pressure remains more than an order of magnitude worse than that LDG was able to achieve.

We again tried enriching the flux space; the results are in Table C.2. Here, however, the results are merely comparable to those in the enriched flux space experiments using the previous two inner products. With cubic elements, enriching the fluxes made for slightly worse results, perhaps due to round-off errors. All told, it appears that whatever is lost to scale inequivalence in the previous two inner products is regained through higher-fidelity flux approximation.

5.4. Mesh-Dependent Inner Product, Least-Squares Compromise $\alpha_3 = \alpha_4 = \alpha_5 = 1$. Finally, we tried an inner product with weights just as in Section 5.3, except $\alpha_3 = \alpha_4 = \alpha_5 = 1$. The rationale was that, in the norm of U , these weights are applied to the tensor σ , which is a derivative, so the natural norm for U in a least-squares approach (arising from concern for scale equivalence of terms within the form $b(\cdot, \cdot)$) would have an extra factor of h on u_1 and u_2 compared to the components of σ .

As can be seen in Table D.1, the results are almost identical to those reported in the previous section. The only exception is the error in the pressure on a cubic, 64×64 mesh, for which the present inner product produced an error about half that produced by the previous inner product.

We again tried enriching the flux space; the results are in Table D.2. The enriched fluxes again give us lower error, but as we refine, the advantage this gives us appears to become less significant; for cubic elements, the error values for the 32×32 mesh are nearly identical to those we attained for this inner product without enriching the fluxes.

6. Conclusions and Future Work. A robust application of the DPG method requires a test space inner product that simultaneously allows a proof of coercivity and continuity of the variational form and achieves scale equivalence in both the inner product matrix used to compute the optimal test functions, and in the stiffness matrix used to compute the solution. In this paper, we have applied the DPG method to the Stokes problem, comparing several inner product choices. The two inner products that did not account for scale equivalence within the inner product matrix both demonstrated substantially poorer performance; while those that did account for it achieved optimal convergence rates.

The fact that our L^2 errors in pressure were substantially worse than those for the LDG method suggests that there may be a better choice of inner product; it may be that an examination of coercivity (here absent) would suggest a better choice. The strategy we intend to employ in the future is to use test norms motivated by examining the *optimal test norm* studied in previous DPG efforts (see [10, Sec. 2]); this is a norm on the *global* test space for which $\|\cdot\|_E = \|\cdot\|_U$.

Enriching the flux space erased most of the distinctions between our various inner products, and greatly reduced the errors observed in the pressure. It appears that the benefit of using a better inner product is that we can save the computational cost associated with the enriched flux space!

In the future, we plan to investigate the *hp*-adaptive solution of Stokes equations using DPG, which offers stability independent of discretization parameters. We also plan to use DPG to solve Stokes on polygons and polyhedra.

The work presented here was completed using L. Demkowicz's *hp*-adaptive code; we are presently implementing a DPG framework using Intrepid [2] and Trilinos [8].

REFERENCES

- [1] P. B. BOCHEV AND M. D. GUNZBERGER, *Least-Squares Finite Element Methods*, vol. 166 of Applied Mathematical Sciences, Springer, 2009, pp. 114–128.
- [2] P. B. BOCHEV, R. C. KIRBY, K. J. PETERSON, AND D. RIDZAL, *Intrepid project*, <http://trilinos.sandia.gov/packages/intrepid/>.
- [3] J. CHAN, L. DEMKOWICZ, R. MOSER, AND N. ROBERTS, *A new discontinuous Petrov-Galerkin method with optimal test functions. Part V: Solution of 1D Burgers' and Navier-Stokes equations*, ICES Technical Report, (2010).
- [4] B. COCKBURN, G. KANSCHAT, D. SCHÖTZAU, AND C. SCHWAB, *Local discontinuous Galerkin methods for the Stokes system*, SIAM Journal on Numerical Analysis, 40 (2003), pp. 319–343.
- [5] L. DEMKOWICZ AND J. GOPALAKRISHNAN, *A class of discontinuous Petrov-Galerkin methods. Part I: The transport equation*, Computer Methods in Applied Mechanics and Engineering, 199 (2010), pp. 1558 – 1572.
- [6] L. D. DEMKOWICZ AND J. GOPALAKRISHNAN, *A class of discontinuous Petrov-Galerkin methods. Part II: Optimal test functions.*, ICES Technical Report, (2009).
- [7] L. D. DEMKOWICZ, J. GOPALAKRISHNAN, AND A. NIEMI, *A class of discontinuous Petrov-Galerkin methods. Part III: Adaptivity*, (2010).
- [8] M. A. HEROUX, R. A. BARTLETT, V. E. HOWLE, R. J. HOEKSTRA, J. J. HU, T. G. KOLDA, R. B. LEHOUCQ, K. R. LONG, R. P. PAWLOWSKI, E. T. PHIPPS, A. G. SALINGER, H. K. THORNQUIST, R. S. TUMINARO, J. M. WILLENBRING, A. WILLIAMS, AND K. S. STANLEY, *An overview of the Trilinos project*, ACM Trans. Math. Softw., 31 (2005), pp. 397–423.
- [9] T. J. HUGHES, G. SCOVAZZI, P. B. BOCHEV, AND A. BUFFA, *A multiscale discontinuous Galerkin method with the computational structure of a continuous Galerkin method*, Computer Methods in Applied Mechanics and Engineering, 195 (2006), pp. 2761 – 2787.
- [10] J. ZITELLI, I. MUGA, L. DEMKOWICZ, J. GOPALAKRISHNAN, D. PARDO, AND V. CALO, *A class of discontinuous Petrov-Galerkin methods. Part IV: Wave propagation*, (2010).

Appendix

A. Numerical Results: Generic Inner Product.

TABLE A.1

L^2 error and h -convergence rates for generic inner product selected without reference to continuity argument, as defined in Section 5.1: comparison with LDG [4].

Quadratic Elements								
	DPG Error		LDG Error		DPG Error		LDG Error	
Mesh Size	u	rate	u	rate	p	rate	p	rate
2×2	3.6e-2	-	-	-	5.9e-1	-	-	-
4×4	5.1e-3	2.82	-	-	2.0e-1	1.58	-	-
8×8	8.1e-4	2.73	2.0e-4	-	1.7e-1	0.92	5.1e-4	-
16×16	1.6e-4	2.59	2.4e-5	3.06	9.6e-2	0.81	1.2e-4	2.09
32×32	3.9e-5	2.46	2.9e-6	3.05	5.1e-2	0.81	3.0e-5	2.04
64×64	9.8e-6	2.36	-	-	2.6e-2	0.83	-	-
Cubic Elements								
	DPG Error		LDG Error		DPG Error		LDG Error	
Mesh Size	u	rate	u	rate	p	rate	p	rate
2×2	2.7e-3	-	-	-	2.7e-1	-	-	-
4×4	1.7e-4	3.97	5.8e-5	-	2.8e-2	3.27	2.4e-4	-
8×8	1.2e-5	3.88	3.6e-6	4.01	3.8e-3	3.08	3.9e-5	2.62
16×16	1.0e-6	3.79	2.2e-7	4.02	5.1e-4	3.01	5.3e-6	2.75
32×32	1.0e-7	3.68	-	-	7.1e-5	2.96	-	-
64×64	1.3e-8	3.55	-	-	3.6e-5	2.67	-	-

TABLE A.2

L^2 error and h -convergence rates for generic inner product selected without reference to continuity argument with enriched fluxes ($k_{\text{flux}} = k + 1, k_{\text{test}} = k + 2$), as defined in Section 5.1: comparison with LDG [4].

Quadratic Elements - Enriched Fluxes								
	DPG Error		LDG Error		DPG Error		LDG Error	
Mesh Size	u	rate	u	rate	p	rate	p	rate
2×2	3.5e-2	-	-	-	7.6e-1	-	-	-
4×4	4.4e-3	3.01	-	-	1.4e-1	2.48	-	-
8×8	5.4e-4	3.02	2.0e-4	-	1.9e-2	2.67	5.1e-4	-
16×16	6.6e-5	3.03	2.4e-5	3.06	3.5e-3	2.61	1.2e-4	2.09
32×32	8.2e-6	3.02	2.9e-6	3.05	7.0e-4	2.54	3.0e-5	2.04
Cubic Elements - Enriched Fluxes								
	DPG Error		LDG Error		DPG Error		LDG Error	
Mesh Size	u	rate	u	rate	p	rate	p	rate
2×2	2.6e-3	-	-	-	3.2e-2	-	-	-
4×4	1.6e-4	3.96	5.8e-5	-	9.4e-3	1.78	2.4e-4	-
8×8	1.0e-5	4.00	3.6e-6	4.01	1.2e-3	2.35	3.9e-5	2.62
16×16	6.0e-7	4.02	2.2e-7	4.02	1.6e-4	2.58	5.3e-6	2.75
32×32	3.7e-8	4.02	-	-	3.3e-5	2.57	-	-

B. Numerical Results: “All Ones” Inner Product.

TABLE B.1

L^2 error and h -convergence rates for an inner product for which $\alpha_i = \widehat{\alpha}_j = 1$ (i.e. with weights selected without concern for scale equivalence) as defined in Section 5.2: comparison with LDG [4].

Quadratic Elements								
	DPG Error		LDG Error		DPG Error		LDG Error	
Mesh Size	\mathbf{u}	rate	\mathbf{u}	rate	p	rate	p	rate
2×2	3.6e-2	-	-	-	5.6e-1	-	-	-
4×4	5.0e-3	2.84	-	-	1.7e-1	1.70	-	-
8×8	7.7e-4	2.77	2.0e-4	-	1.4e-1	1.00	5.1e-4	-
16×16	1.5e-4	2.64	2.4e-5	3.06	8.2e-2	0.87	1.2e-4	2.09
32×32	3.4e-5	2.51	2.9e-6	3.05	4.3e-2	0.85	3.0e-5	2.04
64×64	8.5e-6	2.40	-	-	2.2e-2	0.86	-	-
Cubic Elements								
	DPG Error		LDG Error		DPG Error		LDG Error	
Mesh Size	\mathbf{u}	rate	\mathbf{u}	rate	p	rate	p	rate
2×2	2.7e-3	-	-	-	2.7e-1	-	-	-
4×4	1.7e-4	3.99	5.8e-5	-	2.7e-2	3.30	2.4e-4	-
8×8	1.2e-5	3.92	3.6e-6	4.01	3.7e-3	3.09	3.9e-5	2.62
16×16	9.2e-7	3.84	2.2e-7	4.02	4.8e-4	3.03	5.3e-6	2.75
32×32	9.0e-8	3.73	-	-	6.3e-5	2.99	-	-
64×64	1.4e-8	3.55	-	-	2.8e-4	2.25	-	-

TABLE B.2

L^2 error and h -convergence rates for an inner product for which $\alpha_i = \widehat{\alpha}_j = 1$ (i.e. with weights selected without concern for scale equivalence) with enriched fluxes ($k_{\text{flux}} = k + 1, k_{\text{test}} = k + 2$), as defined in Section 5.2: comparison with LDG [4].

Quadratic Elements - Enriched Fluxes								
	DPG Error		LDG Error		DPG Error		LDG Error	
Mesh Size	\mathbf{u}	rate	\mathbf{u}	rate	p	rate	p	rate
2×2	3.5e-2	-	-	-	7.6e-1	-	-	-
4×4	4.4e-3	3.00	-	-	1.3e-1	2.50	-	-
8×8	5.4e-4	3.02	2.0e-4	-	1.9e-2	2.66	5.1e-4	-
16×16	6.6e-5	3.02	2.4e-5	3.06	3.6e-3	2.60	1.2e-4	2.09
32×32	8.2e-6	3.02	2.9e-6	3.05	7.1e-4	2.53	3.0e-5	2.04
Cubic Elements - Enriched Fluxes								
	DPG Error		LDG Error		DPG Error		LDG Error	
Mesh Size	\mathbf{u}	rate	\mathbf{u}	rate	p	rate	p	rate
2×2	2.5e-3	-	-	-	3.5e-2	-	-	-
4×4	1.6e-4	3.95	5.8e-5	-	8.6e-3	2.02	2.4e-4	-
8×8	1.0e-5	3.98	3.6e-6	4.01	1.2e-3	2.46	3.9e-5	2.62
16×16	6.1e-7	4.01	2.2e-7	4.02	1.5e-4	2.65	5.3e-6	2.75
32×32	3.7e-8	4.02	-	-	3.3e-5	2.59	-	-

C. Numerical Results: Mesh-Dependent Inner Product.

TABLE C.1

L^2 error and h -convergence rates for a mesh-dependent inner product with weights as specified in equations (4.4)-(4.9) and discussed in Section 5.3, an inner product that represents our best compromise between the continuity argument and concerns for scale equivalence in the determination of the optimal test functions.: comparison with LDG [4].

Quadratic Elements								
	DPG Error		LDG Error		DPG Error		LDG Error	
Mesh Size	u	rate	u	rate	p	rate	p	rate
2×2	3.6e-2	-	-	-	5.6e-1	-	-	-
4×4	4.7e-3	2.91	-	-	1.1e-1	2.39	-	-
8×8	6.0e-4	2.94	2.0e-4	-	2.8e-2	2.15	5.1e-4	-
16×16	7.6e-5	2.96	2.4e-5	3.06	7.3e-3	2.07	1.2e-4	2.09
32×32	9.5e-6	2.97	2.9e-6	3.05	1.8e-3	2.05	3.0e-5	2.04
64×64	1.2e-6	2.98	-	-	4.3e-4	2.04	-	-
Cubic Elements								
	DPG Error		LDG Error		DPG Error		LDG Error	
Mesh Size	u	rate	u	rate	p	rate	p	rate
2×2	2.7e-3	-	-	-	2.7e-1	-	-	-
4×4	1.6e-4	4.06	5.8e-5	-	2.4e-2	3.47	2.4e-4	-
8×8	9.9e-6	4.04	3.6e-6	4.01	3.1e-3	3.22	3.9e-5	2.62
16×16	6.1e-7	4.03	2.2e-7	4.02	3.9e-4	3.13	5.3e-6	2.75
32×32	3.8e-8	4.02	-	-	4.9e-5	3.08	-	-
64×64	2.4e-9	4.02	-	-	4.3e-6	3.13	-	-

TABLE C.2

L^2 error and h -convergence rates for a mesh-dependent inner product with weights as specified in equations (4.4)-(4.9) and discussed in Section 5.3, with enriched fluxes ($k_{flux} = k + 1, k_{test} = k + 2$): comparison with LDG [4].

Quadratic Elements - Enriched Fluxes								
	DPG Error		LDG Error		DPG Error		LDG Error	
Mesh Size	u	rate	u	rate	p	rate	p	rate
2×2	3.5e-2	-	-	-	7.6e-1	-	-	-
4×4	4.4e-3	3.00	-	-	1.2e-1	2.62	-	-
8×8	5.4e-4	3.01	2.0e-4	-	1.9e-2	2.64	5.1e-4	-
16×16	6.7e-5	3.01	2.4e-5	3.06	4.7e-3	2.46	1.2e-4	2.09
32×32	8.4e-6	3.01	2.9e-6	3.05	1.1e-3	2.35	3.0e-5	2.04
64×64	1.0e-6	3.01	-	-	2.8e-4	2.27	-	-
Cubic Elements - Enriched Fluxes								
	DPG Error		LDG Error		DPG Error		LDG Error	
Mesh Size	u	rate	u	rate	p	rate	p	rate
2×2	2.5e-3	-	-	-	3.5e-2	-	-	-
4×4	1.6e-4	3.99	5.8e-5	-	5.4e-3	2.70	2.4e-4	-
8×8	1.0e-5	3.99	3.6e-6	4.01	6.1e-4	2.92	3.9e-5	2.62
16×16	6.2e-7	4.00	2.2e-7	4.02	1.3e-4	2.74	5.3e-6	2.75
32×32	3.8e-8	4.00	-	-	2.5e-5	2.63	-	-

D. Numerical Results: Mesh-Dependent Inner Product, Least-Squares Compromise.

TABLE D.1

L^2 error and h -convergence rates for an inner product with weights as described in Section 5.4, an inner product that brings concern for scale equivalence in the stiffness matrix into our compromise between the continuity argument and concerns for scale equivalence in the determination of optimal test functions: comparison with LDG [4].

Quadratic Elements								
	DPG Error		LDG Error		DPG Error		LDG Error	
Mesh Size	\mathbf{u}	rate	\mathbf{u}	rate	p	rate	p	rate
2×2	3.6e-2	-	-	-	5.6e-1	-	-	-
4×4	4.6e-3	2.97	-	-	1.2e-1	2.23	-	-
8×8	5.8e-4	2.97	2.0e-4	-	2.7e-2	2.19	5.1e-4	-
16×16	7.4e-5	2.97	2.4e-5	3.06	6.6e-3	2.14	1.2e-4	2.09
32×32	9.4e-6	2.97	2.9e-6	3.05	1.7e-3	2.10	3.0e-5	2.04
64×64	1.2e-6	2.98	-	-	4.1e-4	2.07	-	-
Cubic Elements								
	DPG Error		LDG Error		DPG Error		LDG Error	
Mesh Size	\mathbf{u}	rate	\mathbf{u}	rate	p	rate	p	rate
2×2	2.7e-3	-	-	-	2.7e-1	-	-	-
4×4	1.6e-4	4.10	5.8e-5	-	2.4e-2	3.47	2.4e-4	-
8×8	9.7e-6	4.05	3.6e-6	4.01	3.1e-3	3.22	3.9e-5	2.62
16×16	6.1e-7	4.03	2.2e-7	4.02	3.9e-4	3.12	5.3e-6	2.75
32×32	3.8e-8	4.02	-	-	4.9e-5	3.08	-	-
64×64	2.4e-9	4.02	-	-	2.3e-6	3.26	-	-

TABLE D.2

L^2 error and h -convergence rates for an inner product with weights as described in Section 5.4, an inner product that brings concern for scale equivalence in the stiffness matrix into our compromise between the continuity argument and concerns for scale equivalence in the determination of optimal test functions, with enriched fluxes ($k_{\text{flux}} = k + 1, k_{\text{test}} = k + 2$): comparison with LDG [4].

Quadratic Elements - Enriched Fluxes								
	DPG Error		LDG Error		DPG Error		LDG Error	
Mesh Size	\mathbf{u}	rate	\mathbf{u}	rate	p	rate	p	rate
2×2	3.5e-2	-	-	-	7.6e-1	-	-	-
4×4	4.2e-3	3.08	-	-	4.2e-2	4.18	-	-
8×8	5.2e-4	3.04	2.0e-4	-	1.0e-2	3.11	5.1e-4	-
16×16	6.5e-5	3.02	2.4e-5	3.06	2.7e-3	2.64	1.2e-4	2.09
32×32	8.2e-6	3.02	2.9e-6	3.05	7.3e-4	2.40	3.0e-5	2.04
Cubic Elements - Enriched Fluxes								
	DPG Error		LDG Error		DPG Error		LDG Error	
Mesh Size	\mathbf{u}	rate	\mathbf{u}	rate	p	rate	p	rate
2×2	2.5e-3	-	-	-	3.5e-2	-	-	-
4×4	1.5e-4	4.06	5.8e-5	-	4.5e-3	2.96	2.4e-4	-
8×8	9.5e-6	4.04	3.6e-6	4.01	2.0e-3	2.07	3.9e-5	2.62
16×16	5.9e-7	4.02	2.2e-7	4.02	3.6e-4	2.10	5.3e-6	2.75
32×32	3.7e-8	4.02	-	-	4.9e-5	2.26	-	-

AN INVESTIGATION OF BLOCK PRECONDITIONERS FOR UNSTEADY NAVIER-STOKES

EDWARD G. PHILLIPS*, ERIC C. CYR†, AND JOHN N. SHADID‡

Abstract. In this paper we investigate block upper triangular preconditioners for the saddle point system generated by discretizing the unsteady Navier-Stokes equations. We focus on Schur complement approximations used within the block structure of these preconditioners. We consider Schur complements generated by Neumann series approximations based on various approximations of the convection-diffusion operator. We also consider improvements upon these approximations using a sparse approximate inverse (SPAI) algorithm and a structured probing algorithm. The preconditioners are compared based on numerical results.

1. Introduction. In this paper, we consider the numerical solution of the unsteady Navier-Stokes problem for the flow of viscous Newtonian fluids: Given an open bounded domain $\Omega \subset \mathbf{R}^d$ with boundary $\partial\Omega$, time interval $[0, T]$, and data f , find a velocity field $\mathbf{u} = \mathbf{u}(\mathbf{x}, t)$ and a pressure field $p = p(\mathbf{x}, t)$ satisfying

$$\frac{\partial \mathbf{u}}{\partial t} - \nu \Delta \mathbf{u} + (\mathbf{u} \cdot \nabla) \mathbf{u} + \nabla p = \mathbf{f} \quad \text{on } \Omega \times [0, T], \quad (1.1)$$

$$\nabla \cdot \mathbf{u} = 0 \quad \text{on } \Omega \times [0, T], \quad (1.2)$$

subject to initial and boundary conditions, where ν is the kinematic viscosity, Δ is the Laplacian, and ∇ is the gradient.

Time discretization is applied to this system along with a Newton or Picard linearization. Then spatial discretization using finite differences or finite elements results in large, sparse saddle-point systems of the form

$$\begin{pmatrix} F & B^T \\ B & -C \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix}, \quad (1.3)$$

or

$$\mathcal{A}x = b, \quad (1.4)$$

where u and p are the discrete velocity and pressure, F is the discrete transient convection-diffusion operator for velocity, B^T is the discrete gradient, B is discrete divergence, C is a stabilization matrix, and f and g account for forcing and boundary conditions. If the discretization is LBB stable (see [5] Chapter 5), then no stabilization is required, and $C = 0$. For a detailed description of the linearization and discretization of (1.1) and (1.2) see [5] Chapter 7.

In order to efficiently solve equation (1.3), a preconditioned Krylov method is often used. Many preconditioners employ the block LU factorization

$$\mathcal{A} = \begin{pmatrix} I & 0 \\ BF^{-1} & I \end{pmatrix} \begin{pmatrix} F & B^T \\ 0 & -S \end{pmatrix} \quad (1.5)$$

where

$$S = C + BF^{-1}B^T \quad (1.6)$$

*Department of Applied Mathematics and Scientific Computation, University of Maryland, eg-phillips@math.umd.edu

†Sandia National Laboratories, eccyr@sandia.gov

‡Sandia National Laboratories, jnshadi@sandia.gov

is the pressure Schur complement. For fast convergence of a Krylov method right-preconditioned by \mathcal{P} , the operator $\mathcal{A}\mathcal{P}^{-1}$ should have few distinct eigenvalues. As noted in [9], if we let \mathcal{P} be the upper block triangular factor, then $\mathcal{A}\mathcal{P}^{-1}$ is the lower block triangular factor and has a single eigenvalue of 1. Building on this idea, we let \mathcal{P} be an approximation to this block factor

$$\mathcal{P} = \begin{pmatrix} \hat{F} & B^T \\ 0 & -\hat{S} \end{pmatrix}, \quad (1.7)$$

where \hat{F} and \hat{S} are approximations for F and S respectively (see [4] for a more general discussion of approximate block factorization preconditioners). To apply \mathcal{P}^{-1} , only the action of the inverses of \hat{F} and \hat{S} is required. F^{-1} is often well approximated by a multi-grid preconditioner as argued in [5] Chapter 8 and [14], so we set $\hat{F} = F$ and focus on approximations for the Schur complement.

The aim of this paper is to investigate the quality of various Schur complement approximations. The immediate goal is not an efficient block preconditioner but an increased understanding of how Schur complement approximations impact preconditioning. We consider how each Schur complement approximation affects the convergence of a Krylov method applied to the exact Schur complement and how this compares to the effect when the approximation is used in the block preconditioner (1.7) for the saddle point system (1.3). For Schur complement approximations which are based on approximations of F^{-1} we also want to compare the effect of the F^{-1} approximation as a preconditioner for F . We are particularly interested in how the performance of these preconditioners scales with CFL and Reynolds number as finding good, inexpensive approximations to S is difficult for large CFL or Reynolds number [1].

The remainder of this paper is structured as follows. In Section 2 we introduce several Schur complement approximations based on Neumann series approximations of F^{-1} . In Sections 3-4 we consider two ways to improve upon these approximations: the sparse approximate inverse algorithm and structured probing. Section 5 describes another approximate block factorization preconditioner, the least-squares commutator, for comparison. Section 6 contains some computational results for assessing the various preconditioners. Finally, conclusions are drawn in Section 7.

2. Neumann series approximations. In considering approximations for the Schur complement, we first investigate those produced by approximating F^{-1} by some \tilde{F}^{-1} . The Schur complement is then approximated through the explicit product

$$\hat{S} = C + B\tilde{F}^{-1}B^T. \quad (2.1)$$

The Neumann series is a simple polynomial approximation for the inverse of a matrix (see [10] Section 12.3.1 for a brief discussion). Given a nonsingular matrix F and a preconditioner M , if $\rho(I - FM^{-1}) < 1$, the following expansion holds

$$F^{-1} = M^{-1} \sum_{i=0}^{\infty} (I - FM^{-1})^i. \quad (2.2)$$

Truncating this series gives the approximation

$$\tilde{F}^{-1} = M^{-1} \sum_{i=0}^{K-1} (I - FM^{-1})^i, \quad (2.3)$$

the K term Neumann series preconditioned by M . Since we require $\rho(I - FM^{-1}) < 1$, a good choice for M is an easily invertible approximation for F . Note that the 1 term Neumann series is equal to the preconditioner M^{-1} itself.

The choice of M is then very important. A good choice of M will lead the Neumann series to converge and will produce a good approximation for the Schur complement. An important issue for efficient computing is the sparsity of \hat{S} . Since we do not alter S beyond the approximation of F^{-1} , it is important that \tilde{F}^{-1} is constructed to be sparse. Consequently M should be chosen to be sparse, and since the density of \tilde{F}^{-1} increases as the number of Neumann terms increases, K should be kept small. For M we consider the following sparse approximations to F or F^{-1} .

SIMPLE

SIMPLE approximates F by its diagonal.

SIMPLEC

SIMPLEC approximates F by a diagonal matrix, where the i th diagonal entry M_{ii} is the sum of the absolute values of the row F_{i*} . This expands on the idea of SIMPLE by attempting to incorporate more data from F into the diagonal.

Block(k)

Given a whole number k , the block Jacobi approximation preserves the block diagonal of F composed of $k \times k$ blocks. This expands on SIMPLE by preserving data off of the diagonal. Naturally, k should be taken modestly so M is sparse. We have considered $k = 2, 3$, and 4 , and $k = 1$ gives SIMPLE. We consider systems where the x and y velocities are split so $u^T = (u_x^T, u_y^T)$. In this case, the blocking accounts for the influence of closely indexed spatial nodes on each other. The effect of this blocking approximation will then be largely contingent on index ordering and flow direction.

ILU(0)

ILU(0) is the incomplete LU factorization with no fill-in. It computes the LU decomposition of F only filling in according to the sparsity pattern of F . M^{-1} is then given by $U^{-1}L^{-1}$. See [10] Section 10.3.2 for an algorithmic description of ILU(0). This approximation is used mostly as a benchmark since it produces denser approximations than desired.

ILUT(τ)

ILUT is the incomplete LU factorization with threshold. Given a threshold τ , ILUT computes the LU decomposition, but does not fill in for any element of F whose magnitude is less than τ times the norm of its row. See [10] Section 10.4.1 for an algorithmic description.

SPAI

SPAI is a sparse approximate inverse. M^{-1} is computed to minimize the norm $\|F(M^{-1})_{*i} - e_i\|_2$ within a tolerance for each column of M^{-1} , subject to a constraint on the number of nonzeros allowed per column. SPAI begins minimization for each column in the direction of the corresponding identity column. Nonzeros are iteratively added to the search direction based on the residual until either the tolerance or the maximum number of nonzeros is reached. For precise algorithmic details, SPAI is described as the Approximate Inverse Algorithm in [2]. We used the MATLAB SPAI implementation SPAI2 [7].

We will refer to a K term Neumann series preconditioned by any of these choices of M as (preconditioner_name). K (i.e. a 3 term Neumann series with a 2×2 block Jacobi preconditioner will be referred to as Block(2).3).

3. Other uses for SPAI. In addition to approximating F^{-1} , SPAI can be used in other ways to construct an approximate Schur complement. First, it can be used to approximate the product $F^{-1}B^T$ by a matrix X . This can be accomplished by minimizing $\|FX_{*i} - B_{i*}^T\|_2$ for each column of B^T with the SPAI algorithm. Once X is computed, S can be approximated by $\hat{S} = BX$. Because X has fewer columns than M this requires less work than the application of SPAI defined above. This second use of SPAI will be referred to as SPAIb. Note that SPAIb cannot act as a preconditioner for a Neumann series approximating F^{-1} .

Another use of SPAI is to apply it to the entire Schur complement approximation. Note that in practice we do not explicitly require the matrix \hat{S} for preconditioning. We merely need the action of \hat{S}^{-1} . If we approximate S and end up with a matrix which is denser than we would like, we can obtain a sparse approximation of \hat{S}^{-1} by applying SPAI.

4. Structured probing. Another method which can be used to produce a sparse approximation of the Schur complement is structured probing, as defined in [13]. Structured probing approximates a matrix $A \in \mathbf{R}^{n \times n}$ according to a chosen sparsity pattern represented in a matrix $H \in \{0, 1\}^{n \times n}$. A matrix of p probing vectors $X \in \{0, 1\}^{n \times p}$ is then constructed via graph coloring techniques such that if A were already of the desired sparsity pattern, the entries of A would be preserved in the matrix AX . The entries of AX are then mapped into an approximation of A according to the sparsity pattern of H . We use the MATLAB version of the Structured Probing Toolkit [12] to implement probing.

We regard the sparsity of a 1 term Neumann series with a diagonal preconditioner as a benchmark. In this case, if $C = 0$, then \hat{S} has the same sparsity pattern as BB^T . To measure the sparsity of a Schur complement approximation, we appeal to the ratio of nonzeros in \hat{S} to nonzeros in BB^T . If this ratio is much greater than 1 for a good \hat{S} , then we can attempt to preserve the convergence properties of \hat{S} and reduce its density by probing it to the sparsity structure of BB^T . When probing is applied to the Schur complement approximation in this way, we refer to the resulting method as (original_method_name)_P (i.e. a 3 term Neumann series with a 2×2 block Jacobi preconditioner which has been probed will be referred to as Block(2)_3_P).

5. The least-squares commutator. The least-squares commutator (LSC) preconditioner, closely related to the BFBt preconditioner [6], is a popular preconditioner developed for Navier-Stokes systems. It is fast and does not require any data beyond what is needed to construct the problem. Consequently, it makes a good benchmark with which to compare the preconditioners studied here. The idea of LSC, as described in [5] and [3], is to commute the discrete velocity convection-diffusion operator F with the discrete gradient operator B^T so that

$$BF^{-1}B^T \approx BM_v^{-1}B^TF_p^{-1}M_p \quad (5.1)$$

where M_v is the velocity mass matrix, M_p is the pressure mass matrix, and F_p is the discrete pressure convection-diffusion operator. So that F_p need not be constructed explicitly, it is obtained by solving the least squares problem

$$\min ||[M_v^{-1}FM_v^{-1}B^T]_j - M_v^{-1}B^TM_p^{-1}[F_p]_j||_{M_v} \quad (5.2)$$

for each column j of F_p . This defines

$$F_p = M_p(BM_v^{-1}B^T)^{-1}(BM_v^{-1}FM_v^{-1}B^T) \quad (5.3)$$

so we let

$$\hat{S} = BM_v^{-1}B^T(BM_v^{-1}FM_v^{-1}B^T)^{-1}(BM_v^{-1}B^T). \quad (5.4)$$

This approximation requires only the velocity mass matrix which is required to build F . M_v^{-1} is commonly replaced by its diagonal. Then the $BM_v^{-1}B^T$ terms are scaled discrete Laplacian operators for which there exist efficient solvers. The action of \hat{S}^{-1} is then easy to compute.

6. Computational results. The test problem considered is a regularized lid driven cavity on a 32×32 grid, using $Q2 - P1$ stable elements and a Picard linearization. The problem is generated with IFISS [15] using its standard time-stepping procedure (TR-AB2, described in [8]) with $\nu = 1/100$ to a time of 0.4. The time-step is then adjusted to produce a given CFL. The domain is $[-1, 1] \times [-1, 1]$ so $Re = 2/\nu$. The next matrix system is used to test the preconditioners. The Krylov method used is GMRES [11] with a stopping tolerance of 10^{-9} and zero initial guess. This very tight criteria may favor the more robust preconditioners. We assess the effect of the preconditioners on the entire saddle point system, \mathcal{A} , by solving (1.3) and the effect on S and F by solving the explicitly generated equations

$$-Sp = g - BF^{-1}f, \quad (6.1)$$

and

$$Fu = f - B^T p \quad (6.2)$$

respectively, as these equations produce the same solution as (1.3). For (6.1) we use \hat{S} as preconditioner and for (6.2) we use \tilde{F} . We will use GMRES iteration counts and total computation time as measures of performance. We note that the MATLAB codes used may not be highly optimized, but computation time can give an indication of the relative performance of the preconditioners studied. The computation times for probing and SPAI may be particularly inflated by their MATLAB implementations.

We begin by considering the performance of Schur complement approximations based on single term Neumann series approximations for F^{-1} : SIMPLE_1, SIMPLEC_1, Block(2)_1, ILU(0)_1, ILUT(0.25)_1, and SPAI_1. We use Block(2) because Block(3) and Block(4) give very similar results with slightly higher density. We found that the choice of $\tau = 0.25$ for ILUT gives a good balance between density and efficiency. We also compare SPAIb and LSC for which Neumann series on F do not apply. For both SPAI and SPAIb we use the default stopping tolerance of 0.4 and a maximum of 50 non-zeros per column. Initially our results will be based on setting $Re = 200$. Iteration counts and computation time are compared as functions of CFL for \mathcal{A} in Figure 6.1, for S in Figure 6.2, and for F in Figure 6.3. The first thing to notice is that iteration count for every preconditioner has the same relationship to CFL: it is moderate and approximately constant for CFL up to about 5, then begins to increase for CFL greater than 5. Because of this dependence on CFL, performance of preconditioners will be judged largely on the right-hand tail in these figures.

In general, the growth rate of iteration count is very similar between \mathcal{A} , S , and F , but we see an interesting effect with Block(2)_1. Notice that for F , Block(2)_1 performs worse than SIMPLEC_1 at the largest CFL, with iteration count as large as that of SIMPLE_1. But for S and \mathcal{A} , Block(2)_1 has a lower iteration count than SIMPLEC_1 at this CFL. We can also see that the difference in iteration count between SIMPLE_1 and SIMPLEC_1 is dramatically greater for S and \mathcal{A} than for F . These observations show that the performance of a Schur complement approximation using \tilde{F}^{-1} is not entirely contingent on the performance of \tilde{F}^{-1} on F . On the other hand, it seems that the performance of a block preconditioner using Schur complement approximation \hat{S} relies heavily on the performance of \hat{S} preconditioning S . Similar trends are seen in computation time. Note that computation time is large for SPAI_1 because of the extra work involved during minimization. An investigation of the large computation time for SPAIb at low CFL is deferred until density information in Table 6.2 is presented.

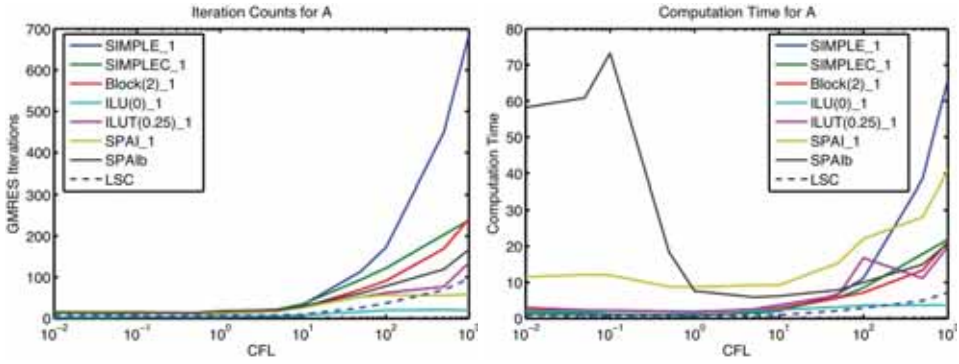


FIG. 6.1. Iteration count and computation time for various single Neumann term preconditioners applied to \mathcal{A} .

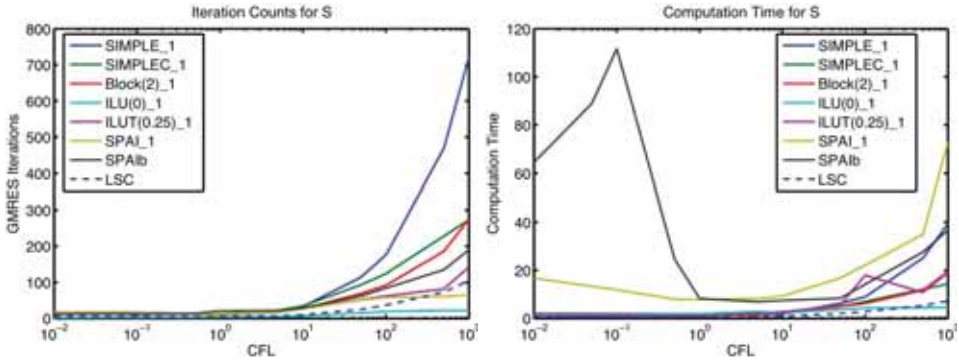
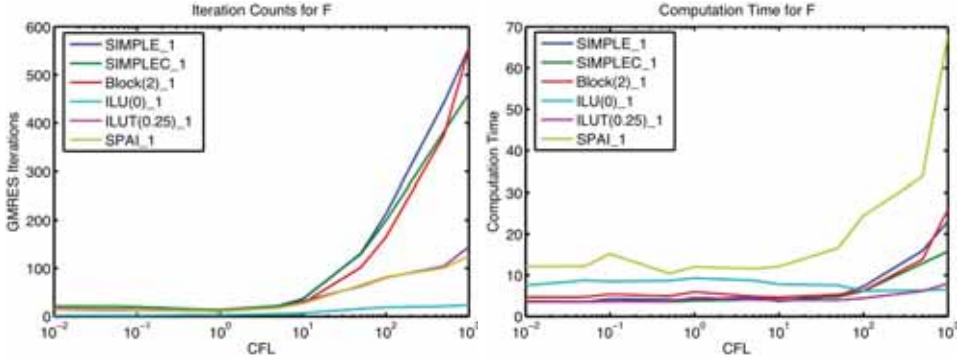
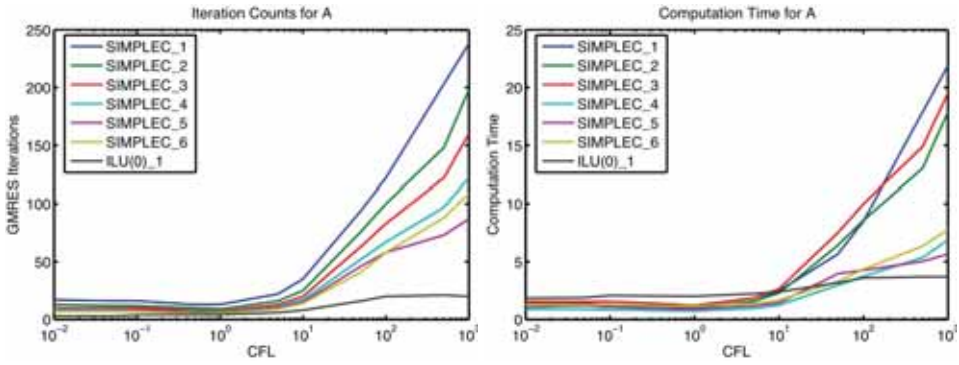
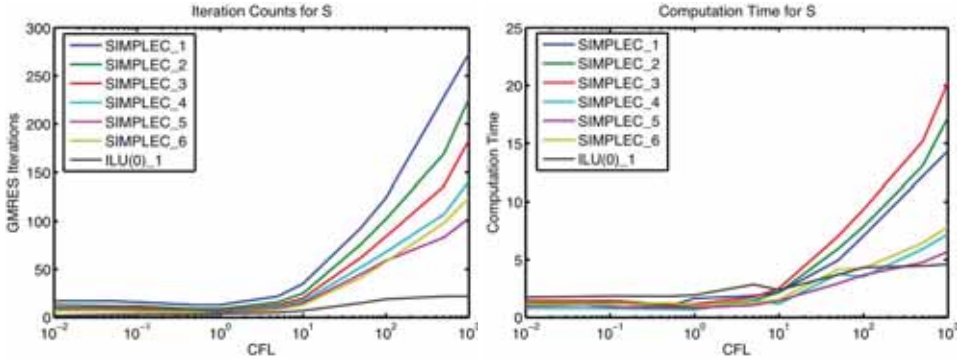


FIG. 6.2. Iteration count and computation time for various single Neumann term preconditioners applied to S .

We now turn to higher order Neumann series approximations for F^{-1} . To avoid excessive density in the Schur complement approximation, we add more Neumann terms only to the sparsest preconditioners: SIMPLE, SIMPLEC, and Block(2). We observed that iteration count improved for each preconditioner when a second Neumann term was added, but SIMPLEC was the only preconditioner for which three or more Neumann terms improved iteration count for all CFL. This effect is well explained by considering the spectral radius $\rho(I - FM^{-1})$, as shown in Table 6.1. For SIMPLE and Block(2), the spectral radius is much greater than 1 for large CFL. As a result, iteration count degrades for large CFL when Neumann terms are added. The spectral radius for SIMPLEC is close to 1 for all CFL considered. Although it slightly exceeds 1 for larger CFL, this is good enough to see iteration count improvement with up to 6 Neumann terms for F and up to 5 for S and \mathcal{A} , as shown in Figures 6.4 – 6.6. The slope of iteration count decreases for each Neumann term added in each of the 3 systems. The same sort of decrease is apparent in computation time, with the exception of the third Neumann term, until SIMPLEC_5 is comparable to ILU(0)_1.

CFL	0.01	0.05	0.1	0.5	1	5	10	50	100	500	1000
SIMPLE	0.7	0.8	0.7	0.6	0.5	0.7	1.0	3.0	5.5	25.3	50.2
SIMPLEC	0.9	0.9	0.9	0.8	0.7	0.7	0.9	1.0	1.1	1.3	1.3
Block(2)	0.7	0.7	0.7	0.5	0.4	0.5	0.7	1.2	2.1	3.4	6.6

TABLE 6.1
Values of $\rho(I - FM^{-1})$ for 3 choices of M .

FIG. 6.3. Iteration count and computation time for various single Neumann term preconditioners applied to F .FIG. 6.4. Iteration count and computation time for Neumann series preconditioned by SIMPLEC applied to A .FIG. 6.5. Iteration count and computation time for Neumann series preconditioned by SIMPLEC applied to S .

Although adding Neumann terms improves iteration count and computation time, it also increases the density of \hat{S} . This increase in density can be seen in Table 6.2. The ratio of non-zeros in \hat{S} to non-zeros in BB^T is constant over CFL for the Neumann series preconditioned by SIMPLEC, increasing with each term. The ratio is also constant over CFL for ILU(0)_1, but is much greater. Having comparable computation time and lower density makes SIMPLEC_5 more favorable than ILU(0)_1 for larger problems. ILUT(0.25)_1 and SPAI_1 grow in density

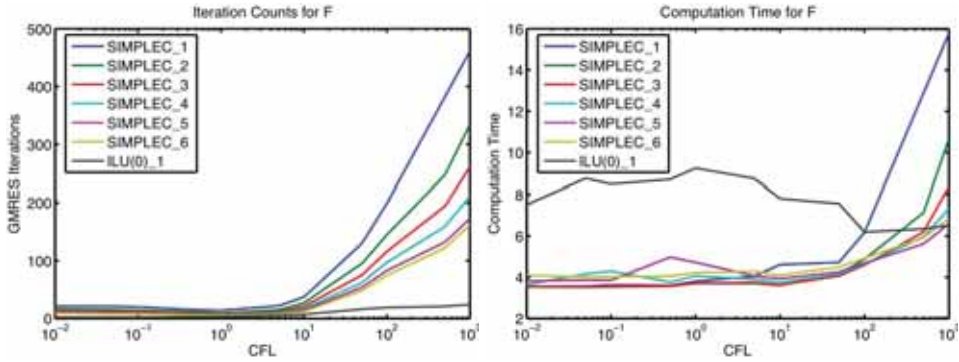


FIG. 6.6. Iteration count and computation time for Neumann series preconditioned by SIMPLEC applied to F .

as CFL increases, while SPAIb, in contrast, decreases in density for larger CFL. This explains the large computation times seen with SPAIb for low CFL in Figures 6.1 and 6.2. In this case, the SPAI algorithm ran longer for low CFL, adding more non-zeros, without reaching its tolerance.

CFL	0.01	0.05	0.1	0.5	1	5	10	50	100	500	1000
SIMPLEC.1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
SIMPLEC.2	2.6	2.6	2.6	2.6	2.6	2.6	2.6	2.6	2.6	2.6	2.6
SIMPLEC.3	4.8	4.8	4.8	4.8	4.8	4.8	4.8	4.8	4.8	4.8	4.8
SIMPLEC.4	7.4	7.4	7.4	7.4	7.4	7.4	7.4	7.4	7.4	7.4	7.4
SIMPLEC.5	10.3	10.3	10.3	10.3	10.3	10.3	10.3	10.3	10.3	10.3	10.3
SIMPLEC.6	13.3	13.3	13.3	13.3	13.3	13.3	13.3	13.3	13.3	13.3	13.3
ILU(0).1	31.6	31.6	31.6	31.6	31.6	31.6	31.6	31.6	31.6	31.6	31.6
ILUT(0.25).1	1.0	1.0	1.0	1.0	1.0	1.0	1.1	3.5	7.2	21.1	24.9
SPAI.1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.4	1.6	2.0	2.6
SPAIb	2.5	2.5	2.5	1.3	1.0	1.0	1.0	1.0	1.0	1.0	1.0

TABLE 6.2
Ratio of non-zeros in \hat{S} to non-zeros in BB^T .

To assess the performance of probing, we apply it to the Schur complement approximations defined by ILU(0).1 and the SIMPLEC Neumann series with up to 5 terms. For this problem, using the sparsity pattern of BB^T , 27 probing vectors are required. Iteration counts and computation time are compared with the original non-probed preconditioners in Figures 6.7 and 6.8. For the SIMPLEC preconditioners, probing only slightly increases iteration count. Computation time is much greater for probing in this size problem due to the expensive graph coloring process, but grows slower with CFL. ILU(0).1 sees a much greater deterioration in iteration count when probing is applied. This can be attributed to the higher density of ILU(0).1 as compared to the SIMPLEC preconditioners. More information is lost from ILU(0).1 when probed to the sparsity pattern of BB^T .

Given what we have seen, the preconditioners of most interest are ILU(0).1, ILUT(0.25).1, SPAIb, SIMPLEC.5, and SIMPLEC.5.P. Although dense, ILU(0).1 is consistently the best in terms of both iteration count and computation time. ILUT(0.25).1 grows in density as CFL increases but is competitive in iteration count and computation time. SPAIb may have larger iteration counts than SPAI, but the decrease in its density with growing CFL and the need for fewer minimization procedures make it a much faster preconditioner for large CFL. SIMPLEC.5 is the best of the higher order Neumann series preconditioners considered, and SIMPLEC.5.P performs almost as well in iteration count while being 10.3 times sparser.

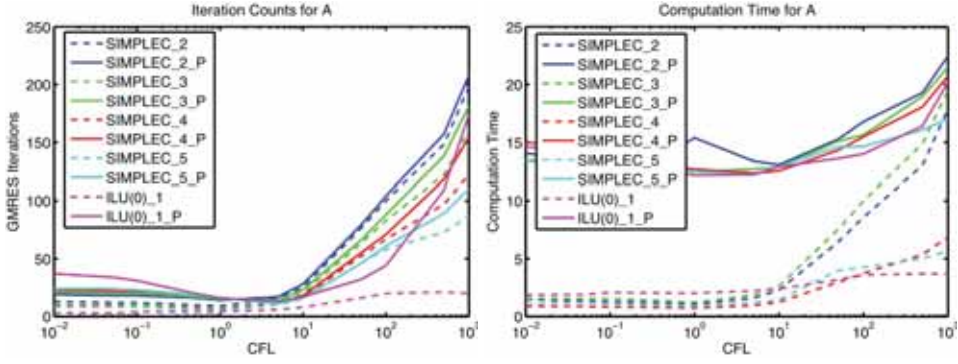


Fig. 6.7. Iteration count and computation time for probed preconditioners applied to \mathcal{A} .

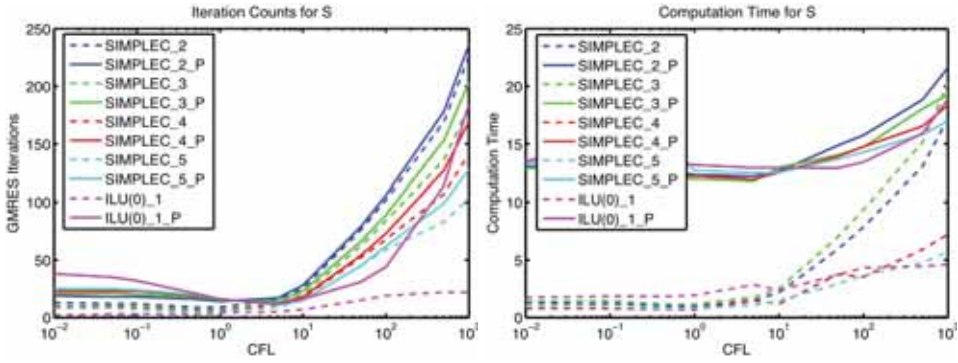


Fig. 6.8. Iteration count and computation time for probed preconditioners applied to S .

We assess these preconditioners further by showing how their performance scales with Reynolds number and mesh size. The Reynolds number is modified by changing the value of ν . The scaling of iteration count with Reynolds number is plotted in Figure 6.9. Observe that ILU(0)_1 and ILUT(0.25)_1 degrade with an increase in Reynolds number, as they do not converge for very high Reynolds number when coupled with large CFL. The other 3 preconditioners scale similarly with Reynolds number and are comparable to LSC for $Re = 20000$.

We compared the performance of the 5 best preconditioners with 2 finer meshes: 64×64 and 128×128 . Iteration counts and computation time for these preconditioners on the three meshes with $Re = 200$ are plotted in Figures 6.10 – 6.12. The detriment of ILU(0)_1's density is clear on the 128×128 grid as MATLAB, running on 1 gb of RAM, ran out of memory. ILUT(0.25)_1 was also too dense to run at $CFL = 1000$. Keep in mind that the Schur complement approximation is explicitly computed. In practice, matrix free multi-grid methods could be used to avoid constructing it on the finest grid. Otherwise, we do not see much change in iteration count. But as far as computation time, it appears that the probing graph coloring algorithm as implemented gets more expensive with mesh refinement, making SIMPLEC_5_P very slow on the 128×128 mesh. Ultimately, SPAIb fares the best with mesh refinement obtaining the lowest computation times for large CFL on the finest mesh, owing to its sparsity and accurate approximation of S .

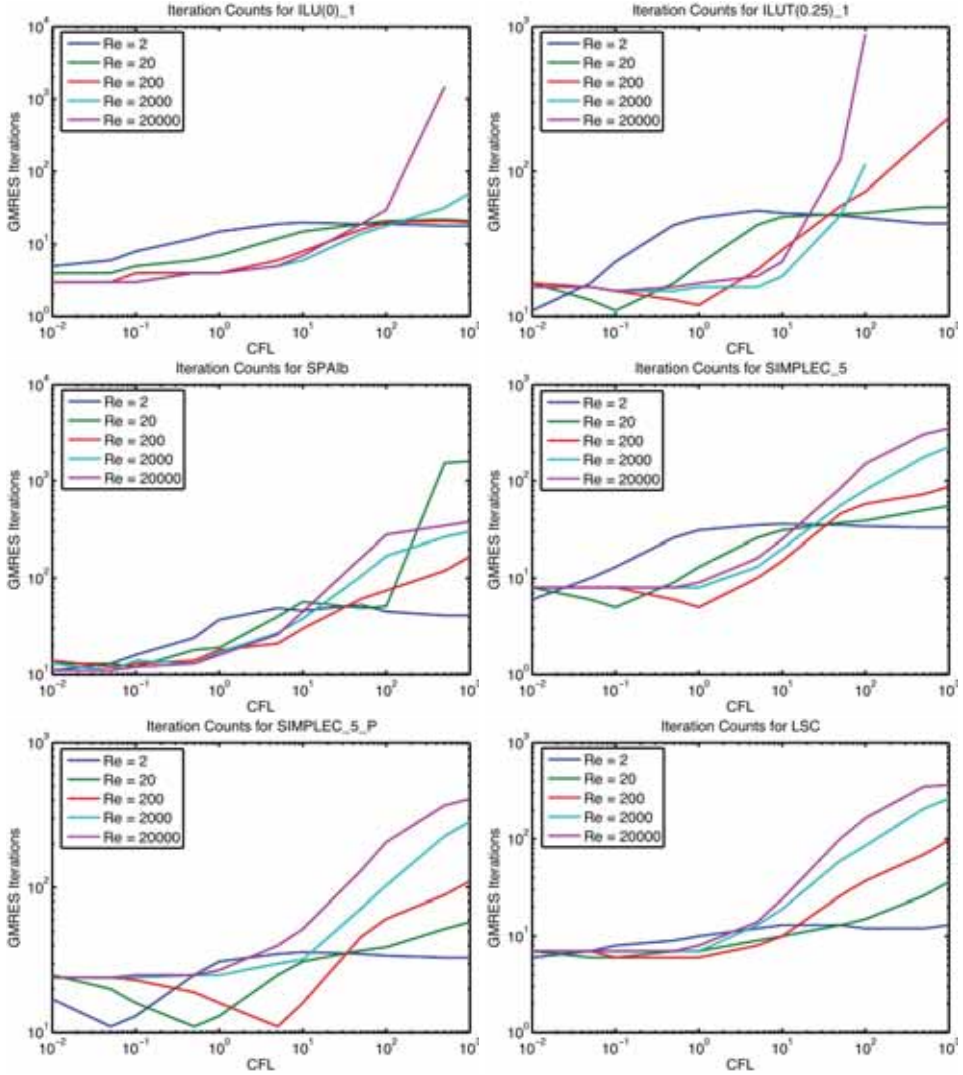


FIG. 6.9. Iteration count for the best preconditioners as it scales with Reynolds number on \mathcal{A} . Note that $ILU(0)_1$ did not converge for $Re = 20000, CFL = 1000$. $ILUT(0.25)_1$ did not converge for $Re = 2000, 20000, CFL = 500, 1000$.

7. Conclusion. This paper presented a number of preconditioners for saddle point problems arising from discretizations of the unsteady Navier-Stokes equations. The approach is based on an approximate block triangular factorization, focusing on approximating the Schur complement. We considered Schur complement approximations based on Neumann series approximations of F^{-1} . These rely heavily upon the spectral radius $\rho(I - FM^{-1})$. But the efficiency of an approximation \tilde{F}^{-1} as a preconditioner for F is not directly analogous to the efficiency of a block preconditioner using \tilde{F}^{-1} to approximate S . Structured probing can be employed to reduce the density of a Schur complement approximation without increasing iteration count much, but the graph coloring procedure involved proves very expensive. A competitive Schur complement approximation for larger problems is developed by adapting a sparse approximate inverse algorithm to approximate $F^{-1}B^T$. It will be valuable to study

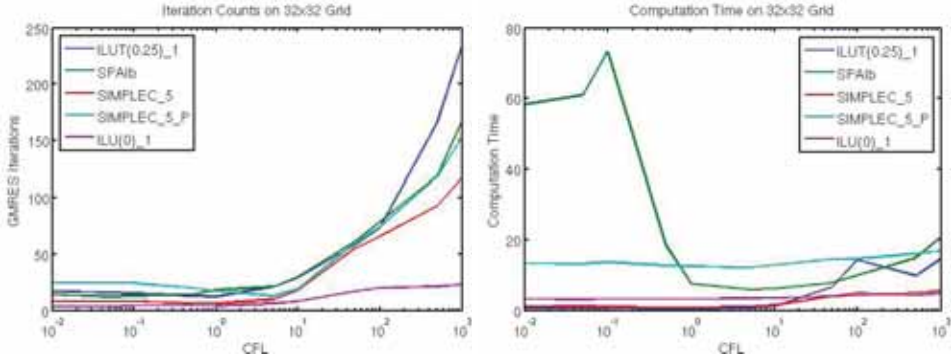


FIG. 6.10. Iteration count and computation time for the best preconditioners on a 32×32 grid for \mathcal{A} .

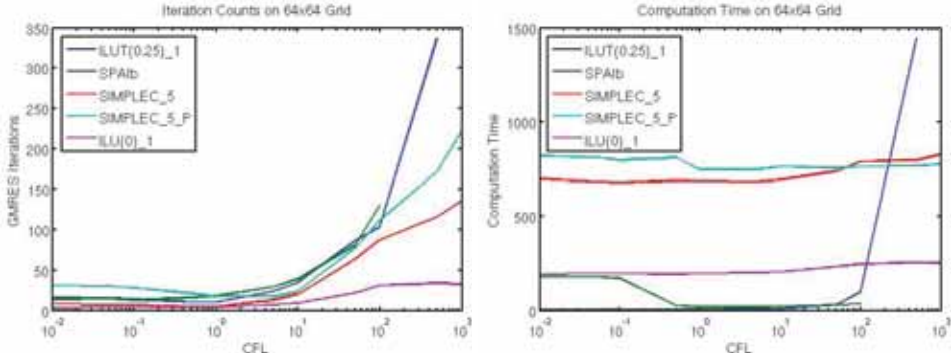


FIG. 6.11. Iteration count and computation time for the best preconditioners on a 64×64 grid for \mathcal{A} . Note that $ILUT(0.25)_1$ did not converge for $CFL = 1000$ and $SPAib$ did not converge for $CFL = 500, 1000$.

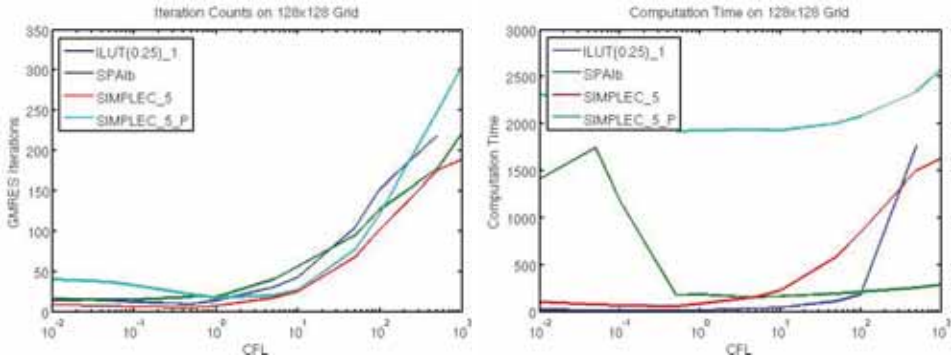


FIG. 6.12. Iteration count and computation time for the best preconditioners on a 128×128 grid for \mathcal{A} . $ILU(0)_1$ consumed too much memory to be run. $ILUT(0.25)_1$ also ran out of memory for $CFL = 1000$.

the robustness of the preconditioners presented here by applying them to other domains and using stabilized finite elements. It is also important to note that in practice a direct solver will not be used to invert the Schur complement approximation. Keeping this in mind, it would be of interest to study how these Schur complement approximations interact with multi-grid methods.

REFERENCES

- [1] M. BENZI AND M. A. OLSHANSKII, *An augmented lagrangian-based approach to the oseen problem*, SIAM Journal on Scientific Computing, 28 (2006), pp. 2095–2113.
- [2] E. CHOW AND Y. SAAD, *Approximate inverse techniques for block-partitioned matrices*, SIAM Journal on Scientific Computing, 18 (1995), pp. 1657–1675.
- [3] H. ELMAN, V. E. HOWLE, J. SHADID, R. SHUTTLEWORTH, AND R. TUMINARO, *Block preconditioners based on approximate commutators*, SIAM Journal on Scientific Computing, 27 (2006), pp. 1651–1668.
- [4] H. ELMAN, V. E. HOWLE, J. SHADID, R. SHUTTLEWORTH, AND R. TUMINARO, *A taxonomy and comparison of parallel block multi-level preconditioners for the incompressible navier-stokes equations*, Journal of Computational Physics, 227 (2008), pp. 1790–1808.
- [5] H. ELMAN, D. SILVESTER, AND A. WATHEN, *Finite Elements and Fast Iterative Solvers: With Applications in Incompressible Fluid Dynamics*, Oxford University Press, Oxford, 2005.
- [6] H. C. ELMAN, *Preconditioning for the steady-state navier-stokes equations with low viscosity*, SIAM Journal on Scientific Computing, 20 (1996), pp. 1299–1316.
- [7] M. D. HUGHES AND K. CHEN, *SPAI2*.
- [8] D. A. KAY, P. M. GRESHO, D. F. GRIFFITHS, AND D. J. SILVESTER, *Adaptive time-stepping for incompressible flow part ii: Navier-stokes equations*, SIAM Journal on Scientific Computing, 32 (2010), pp. 111–128.
- [9] M. F. MURPHY, G. H. GOLUB, AND A. J. WATHEN, *A note on preconditioning for indefinite linear systems*, SIAM Journal on Scientific Computing, 21 (2000), pp. 1969–1972.
- [10] Y. SAAD, *Iterative Methods for Sparse Linear Systems*, SIAM, Philadelphia, 2 ed., 2003.
- [11] Y. SAAD AND M. H. SCHULTZ, *Gmres: A generalized minimum residual algorithm for solving nonsymmetric linear systems*, SIAM Journal on Scientific and Statistical Computing, 7 (1986), pp. 856–869.
- [12] C. SIEFERT, *Structured Probing Toolkit*.
- [13] C. SIEFERT AND E. DE STURLER, *Probing methods for saddle-point problems*, Electronic Transactions on Numerical Analysis, 22 (2006), pp. 163–183.
- [14] D. SILVESTER, H. ELMAN, D. KAY, AND A. WATHEN, *Efficient preconditioning of the linearized navier-stokes equations*, Journal of Computational and Applied Mathematics, 128 (1999), pp. 261–279.
- [15] D. SILVESTER, H. ELMAN, AND A. RAMAGE, *IFISS: Incompressible Flow Iterative Solution Software*.

EFFICIENTLY COMPUTING TENSOR EIGENVALUES ON A GPU

GREY BALLARD*, TAMARA KOLDA[†], AND TODD PLANTENGA[‡]

Abstract. The tensor eigenproblem has many important applications, and both mathematical and application-specific communities have taken recent interest in the properties of tensor eigenpairs as well as methods for computing them. In particular, Kolda and Mayo [3] present a generalization of the matrix power method for symmetric tensors. We focus in this work on efficient implementation of their algorithm, known as the shifted symmetric higher-order power method, and on how a GPU can be used to accelerate the computation up to 70× over a sequential implementation for an application involving many small tensor eigenproblems.

1. Introduction. The tensor eigenproblem has many important applications, and both mathematical and application-specific communities have taken recent interest in the properties of tensor eigenpairs as well as methods for computing them. In particular, Kolda and Mayo [3] present a generalization of the matrix power method for symmetric tensors. We focus in this work on efficient implementation of their algorithm, known as the shifted symmetric higher-order power method (SS-HOPM).

The main motivating application for this work involves detection of nerve fibers in the brain from diffusion-weighted magnetic resonance imaging data. In this application, data is gathered for millions of cubic millimeter sized voxels. Determining the number and directions of nerve fiber bundles within each voxel requires solving a small tensor eigenvalue problem. Because each voxel can be resolved independently, the computations are amenable to parallelism, and we focused our implementation on a graphics processing unit (GPU) using the Compute Unified Device Architecture (CUDA) programming framework.

We review the definition of the tensor eigenproblem as well as the SS-HOPM algorithm from [3] in Section 2. All of the tensors discussed here are symmetric, and exploiting symmetry is the foremost sequential optimization we use to gain performance. Symmetric matrices can be stored in half the space and symmetric matrix computations often require only half the flops of their nonsymmetric counterparts; exploiting symmetry in tensors can save storage and computation by much larger factors. In Section 3 we discuss a symmetric tensor storage format and how this compressed format is used in the main computational kernels of SS-HOPM.

Instead of attempting to write an algorithm that offers high parallel performance for computing eigenpairs of tensors of general order and dimension, we focus the GPU implementation on small tensors, as in our motivating application. Because of the inherent parallelism in the problem, we can run many independent threads concurrently on the hardware, and we facilitate efficiency of each thread with careful memory management. We offer an overview of GPU computing in Section 4, describe the motivating application in Section 5, and give the details and results of our implementation in Section 6.

The main contributions of this work are (1) the introduction of a symmetric storage format and means of exploiting symmetry to avoid redundant computation, and (2) a parallel implementation of SS-HOPM. While the implementation is tailored to a specific application, we believe it will be widely applicable to high performance computations with symmetric tensors.

2. Symmetric Tensors and Tensor Eigenpairs. We formally introduce the notion of a symmetric tensor which is invariant under any permutation of its indices. Let $\mathbb{R}^{[m,n]}$ be the set

*UC Berkeley, ballard@cs.berkeley.edu

[†]Sandia National Laboratories, tgkolda@sandia.gov

[‡]Sandia National Laboratories, tplante@sandia.gov

of real-valued order- m tensors where each mode has dimension n .

DEFINITION 2.1 (Symmetric tensor [1]). A tensor $\mathcal{A} \in \mathbb{R}^{[m,n]}$ is symmetric if

$$a_{i_{\pi(1)} \dots i_{\pi(m)}} = a_{i_1 \dots i_m} \quad \text{for all } i_1, \dots, i_m \in \{1, \dots, n\} \text{ and } \pi \in \Pi_m$$

where Π_m is the set of permutations of the set $\{1, \dots, m\}$.

The main computational kernels in the shifted symmetric higher-order power method will be instances of the following definition of symmetric tensor-vector multiply.

DEFINITION 2.2 (Symmetric tensor-vector multiply [3]). Let $\mathcal{A} \in \mathbb{R}^{[m,n]}$ be symmetric and $\mathbf{x} \in \mathbb{R}^n$. Then for $0 \leq p \leq m-1$, the $(m-p)$ -times product of the tensor \mathcal{A} with the vector \mathbf{x} is denoted by $\mathcal{A}\mathbf{x}^{m-p} \in \mathbb{R}^{[p,n]}$ and defined by

$$(\mathcal{A}\mathbf{x}^{m-p})_{i_1 \dots i_p} = \sum_{i_{p+1}, \dots, i_m} a_{i_1 \dots i_m} x_{i_{p+1}} \dots x_{i_m} \quad \text{for all } 1 \leq i_1, \dots, i_p \leq n. \quad (2.1)$$

Note that there is ambiguity in defining a tensor times the same vector in some subset of modes, but due to symmetry the choice of indexing below yields the same result as any other valid definition. Also note that the result of a symmetric tensor-vector multiply is also a symmetric tensor, because any permutation of the indices of the result tensor $(i_1 \dots i_p)$ on the left hand side of Equation 2.1 corresponds to a permutation of the first p indices of the symmetric input tensor entries in the summation on the right hand side which remains invariant.

We recall the definition of a tensor eigenpair used in [3]. There are other definitions of eigenvalues and eigenvectors in the literature, but the relationships between the definitions and the many interesting properties of tensor eigenvalues are beyond the scope of this work.

DEFINITION 2.3 (Symmetric tensor eigenpair [3]). Assume that \mathcal{A} is a symmetric m^{th} -order n -dimensional real-valued tensor. Then $\lambda \in \mathbb{C}$ is an eigenvalue of \mathcal{A} if there exists $\mathbf{x} \in \mathbb{C}^n$ such that

$$\mathcal{A}\mathbf{x}^{m-1} = \lambda\mathbf{x} \quad \text{and} \quad \|\mathbf{x}\|_2 = 1. \quad (2.2)$$

The vector \mathbf{x} is the corresponding eigenvector, and (λ, \mathbf{x}) is called an eigenpair.

Finally, we present the shifted symmetric higher-order power method (SS-HOPM) from [3] as Algorithm 1. This algorithm is a generalization of the matrix power method where the operation $\mathcal{A}\mathbf{x}^{m-1}$ generalizes the matrix-vector product and $\mathcal{A}\mathbf{x}^m$ generalizes the Rayleigh quotient for a unit vector. Algorithm 1 includes the shift parameter α which is chosen to force the underlying function to be convex ($\alpha \geq 0$) or concave ($\alpha < 0$).

The symmetric higher-order power method (with no shift) was introduced in [2, 4], and convergence of the method was proved for certain types of tensors. While the symmetric higher-order power method does not converge in general, choosing a sufficiently large (in absolute value) shift α guarantees convergence of SS-HOPM. The convergence properties of a given eigenpair are characterized in [3], but there are still many open problems regarding choice of starting vector, choice of shift, and finding eigenpairs with certain properties.

3. Exploiting Symmetry.

3.1. Symmetric Tensor Storage. Let $\mathcal{A} \in \mathbb{R}^{[m,n]}$ be a symmetric tensor. In general, \mathcal{A} has n^m entries, but since it is symmetric, many of the entry values are repeated and need not be stored redundantly. We define an *index* as a number $i \in \{1, \dots, n\}$, we define a *tensor index* as an array of m indices corresponding to one entry of the tensor, and we define an *index class* as a set of tensor indices such that the corresponding tensor entries all share a value due to symmetry. For example, for $m = 3$ and $n = 2$, the possible indices are 1 and 2, and the tensor indices $[1, 1, 2]$ and $[1, 2, 1]$ are in the same index class since $a_{112} = a_{121}$.

Algorithm 1 Shifted Symmetric Higher-Order Power Method (SS-HOPM) [3]

Given a tensor $\mathcal{A} \in \mathbb{R}^{[m,n]}$.

Require: $\alpha \in \mathbb{R}$, $\mathbf{x}_0 \in \mathbb{R}^n$ with $\|\mathbf{x}_0\| = 1$. Let $\lambda_0 = \mathcal{A}\mathbf{x}_0^m$.

```

1: for  $k = 0, 1, \dots$  do
2:   if  $\alpha \geq 0$  then
3:      $\hat{\mathbf{x}}_{k+1} \leftarrow \mathcal{A}\mathbf{x}_k^{m-1} + \alpha\mathbf{x}_k$ 
4:   else
5:      $\hat{\mathbf{x}}_{k+1} \leftarrow -(\mathcal{A}\mathbf{x}_k^{m-1} + \alpha\mathbf{x}_k)$ 
6:   end if
7:    $\mathbf{x}_{k+1} \leftarrow \hat{\mathbf{x}}_{k+1} / \|\hat{\mathbf{x}}_{k+1}\|$ 
8:    $\lambda_{k+1} \leftarrow \mathcal{A}\mathbf{x}_{k+1}^m$ 
9: end for
```

We can find a unique representative of an index class by choosing the tensor index whose indices are in nondecreasing order. We define this nondecreasing tensor index as the *index representation* of the index class.

The index classes of \mathcal{A} can also be characterized by the number of occurrences of each index $i \in \{1, \dots, n\}$ in the tensor indices of the index class. Thus, we can define the *monomial representation* of an index class as an array of n integers where the i^{th} entry in the array corresponds to the number of occurrences of the index i in the index class. Following the example given above, the index class that includes $[1, 1, 2]$ and $[1, 2, 1]$ has monomial representation $[2, 1]$ since there are two 1's and one 2 in every tensor index in the class.

In order to avoid redundant storage, we store only the unique values of the tensor (i.e. one value per index class). The following property gives the number of unique values of a dense symmetric tensor.

PROPERTY 3.1. *The number of unique values of a symmetric tensor $\mathcal{A} \in \mathbb{R}^{[m,n]}$ is given by the binomial coefficient*

$$\binom{m+n-1}{m} = \frac{n^m}{m!} + O(n^{m-1}).$$

Proof. Each index class corresponds to a unique value. Counting the number of possible monomial representations of length m with n possible values is equivalent to counting the number of ways to distribute m indistinguishable balls into n distinguishable buckets, where the balls correspond to the indices of the tensor index and the buckets correspond to the possible index values. By a “stars and bars” argument,¹ this number is

$$\binom{m+n-1}{m} = \frac{(n+m-1) \cdots (n+1)n}{m!} = \frac{n^m}{m!} + O(n^{m-1})$$

as claimed. \square

Assuming \mathcal{A} is dense, we can impose an ordering on the unique entries and avoid storing any index information. We choose to use a lexicographic order of the index classes, increasing with respect to the index representation and decreasing with respect to the monomial representation. That is, the index class with index representation $[i_1, i_2, \dots, i_m]$ is listed before $[j_1, j_2, \dots, j_m]$ if $i_1 < j_1$ or if $i_1 = j_1$ and $i_2 < j_2$, and so on. Equivalently, the index class with monomial representation $[k_1, k_2, \dots, k_n]$ is listed before $[l_1, l_2, \dots, l_n]$ if $k_1 > l_1$ or

¹See Theorem 2 in Section 4.6 of [9], for example.

	index			monomial			
1	1	1	1	3	0	0	0
2	1	1	2	2	1	0	0
3	1	1	3	2	0	1	0
4	1	1	4	2	0	0	1
5	1	2	2	1	2	0	0
6	1	2	3	1	1	1	0
7	1	2	4	1	1	0	1
8	1	3	3	1	0	2	0
9	1	3	4	1	0	1	1
10	1	4	4	1	0	0	2
11	2	2	2	0	3	0	0
12	2	2	3	0	2	1	0
13	2	2	4	0	2	0	1
14	2	3	3	0	1	2	0
15	2	3	4	0	1	1	1
16	2	4	4	0	1	0	2
17	3	3	3	0	0	3	0
18	3	3	4	0	0	2	1
19	3	4	4	0	0	1	2
20	4	4	4	0	0	0	3

TABLE 3.1
Set of index classes $\mathcal{J}^{[3,4]}$ in lexicographic order.

if $k_1 = l_1$ and $k_2 > l_2$, and so on. This corresponds to an ordering on monomials in a given polynomial ring (the origin of the terminology). In this case, the index classes correspond to monomials which all have total degree m . See Table 3.1 for an example of lexicographic ordering for both representations in the case $m = 3$ and $n = 4$.

While the lexicographic ordering makes storing index information for every unique value unnecessary, it will be important to compute index information during computations. Since the index representation requires m integers and the monomial representation requires n integers and we expect $n \gg m$ for most problems, we store the index representation and compute monomial representation values implicitly. Note that while the monomial representation will be sparse when $n \gg m$, even a compressed format would require at least m integers.

3.2. Computational Kernels. The two most computationally intensive kernels in Algorithm 1 are computing the scalar $\mathcal{A}\mathbf{x}^m$ and the vector $\mathcal{A}\mathbf{x}^{m-1}$, where $\mathcal{A} \in \mathbb{R}^{[m,n]}$ is symmetric and $\mathbf{x} \in \mathbb{R}^n$. Both of these are instances of the symmetric tensor-vector multiply given in Definition 2.2, with $p = 0$ and $p = 1$, respectively.

3.2.1. Tensor times same vector in all modes. Consider the case $p = 0$:

$$\mathcal{A}\mathbf{x}^m = \sum_{i_1=1}^n \cdots \sum_{i_m=1}^n a_{i_1 \dots i_m} x_{i_1} \cdots x_{i_m} \quad (3.1)$$

For a nonsymmetric tensor, this summation requires at least one multiplication for each term (corresponding to each entry of \mathcal{A}), yielding at least n^m flops. However, we can exploit symmetry to reduce the computational complexity. Note that the tensor index matches the indices of the \mathbf{x} vector entries for each term in the summation. Since the product of a set of numbers is also invariant under permutation, all of the terms in the summation corresponding to the same index class will have the same value.

For example, for $m = 3$ and $n = 2$, the term in the summation corresponding to the tensor index $[1, 1, 2]$ is given by $a_{112} \cdot x_1 \cdot x_1 \cdot x_2 = a_{112} x_1^2 x_2$, and the term in the summation

corresponding to the tensor index $[1, 2, 1]$ is given by $a_{121} \cdot x_1 \cdot x_2 \cdot x_1 = a_{121} x_1^2 x_2$. Any tensor index with monomial representation $[2, 1]$ will yield this value.

We can avoid recomputing the redundant value by instead computing the number of times each unique term appears in the summation, which is given by the following property.

PROPERTY 3.2. *The number of tensor indices of a symmetric tensor $\mathcal{A} \in \mathbb{R}^{[m,n]}$ in the index class with monomial representation $[k_1, k_2, \dots, k_n]$ is given by the multinomial coefficient*

$$\binom{m}{k_1, k_2, \dots, k_n} = \frac{m!}{k_1! k_2! \dots k_n!}.$$

Proof. Consider the monomial representation $[k_1, k_2, \dots, k_n]$. Counting the number of tensor indices in this class is equivalent to counting the number of ways one can distribute m distinct balls into n distinct bins such that the i^{th} bin has k_i balls. Here the balls correspond to the (ordered) indices of the tensor index and the bins correspond to the possible index values. One way to solve this problem is to count the number of ways of filling the first bin (given by the binomial coefficient $\binom{m}{k_1}$), followed by the number of ways of filling the second bin (given by $\binom{m-k_1}{k_2}$), and so on. Using the product rule and after much cancellation, we have

$$\binom{m}{k_1} \cdot \binom{m-k_1}{k_2} \dots \binom{m-(k_1+k_2+\dots+k_{n-1})}{k_n} = \frac{m!}{k_1! k_2! \dots k_n!}$$

as claimed. \square

We can thus rewrite Equation 3.1 as

$$\mathcal{A}\mathbf{x}^m = \sum_{I \in \mathcal{J}^{[m,n]}} \binom{m}{k_1, k_2, \dots, k_n} a_{i_1 \dots i_m} x_1^{k_1} \dots x_n^{k_n}, \quad (3.2)$$

where $\mathcal{J}^{[m,n]}$ is the set of index classes for a symmetric tensor in $\mathbb{R}^{[m,n]}$, and $[k_1, \dots, k_n]$ and $[i_1, \dots, i_m]$ are the monomial and index representations of the index class I , respectively. Equation 3.2 yields Algorithm 2, which assumes the unique values of \mathcal{A} are stored in lexicographic order. For each unique value, the algorithm computes the monomial coefficient and index array associated with the tensor entry and adds the contribution of that term to the accumulating result.

3.2.2. Tensor times same vector in all modes but one. Now consider computing the vector $\mathcal{A}\mathbf{x}^{m-1}$, the case $p = 1$ in Definition 2.2:

$$(\mathcal{A}\mathbf{x}^{m-1})_{i_1} = \sum_{i_2=1}^n \dots \sum_{i_m=1}^n a_{i_1 \dots i_m} x_{i_2} \dots x_{i_m} \quad (3.3)$$

Note that the j^{th} component of $\mathcal{A}\mathbf{x}^{m-1}$ does not depend on every tensor entry, only those tensor entries whose index representation starts with index j . Because of symmetry, Equation 3.3 can be rewritten with i_1 appearing as any index in the tensor index of the tensor value.

As in the case of computing $\mathcal{A}\mathbf{x}^m$, we can exploit symmetry to avoid performing the more than n^m multiplications required to compute all entries of the output vector if we followed Equation 3.3. As before, if a tensor value contributes to the summation for index k of the output vector, its symmetric counterparts will contribute the same value to the sum. Following the example given before, where $m = 3$ and $n = 2$, both a_{112} and a_{121} will contribute to the computation of $(\mathcal{A}\mathbf{x}^{m-1})_1$, and each will contribute the value $a_{112} \cdot x_1 \cdot x_2$. Note that a_{211} will not contribute to the summation for $(\mathcal{A}\mathbf{x}^{m-1})_1$, because its first index is not 1.

Algorithm 2 Compute $y = \mathcal{A}\mathbf{x}^m$ via Equation 3.2, where $\mathcal{A} \in \mathbb{R}^{[m,n]}$ is symmetric, $\mathbf{x} \in \mathbb{R}^n$, and $y \in \mathbb{R}$

Require: A stores the unique entries of \mathcal{A} in lexicographic order

```

1: function  $y = \text{SYMMETRICTENSORVECTORMULTIPLY0}(A, \mathbf{x})$ 
2:    $y = 0$ 
3:    $I = [1, \dots, 1]$  ▷ use index representation (length  $m$ )
4:   for  $j = 1$  to  $\binom{m+n-1}{m}$  do ▷ iterate over unique entries
5:      $\hat{x} = x_{I_1} \cdot x_{I_2} \cdots x_{I_m}$  ▷ compute monomial value
6:      $C = \text{NUMOCC0}(I)$  ▷ compute number of occurrences
7:      $y = y + C \cdot A_j \cdot \hat{x}$  ▷ accumulate sum
8:      $I = \text{UPDATEINDEX}(I)$  ▷ See Algorithm 4
9:   end for
10: end function

```

Require: I has length m with entries in nondecreasing order

```

11: function  $C = \text{NUMOCC0}(I)$ 
12:    $\text{div} = 1$  ▷ divisor of  $\binom{m}{k_1, \dots, k_n}$ 
13:    $\text{curr} = -1$  ▷ current index value
14:    $\text{mult} = -1$  ▷ multiplicity of current index value
15:   for  $j = 1$  to  $m$  do
16:     if  $I_j \neq \text{curr}$  then
17:        $\text{mult} = 1$ 
18:        $\text{curr} = I_j$ 
19:     else ▷ repeated index
20:        $\text{mult} = \text{mult} + 1$ 
21:        $\text{div} = \text{div} \cdot \text{mult}$  ▷ only update divisor if  $\text{mult} > 1$ 
22:     end if
23:   end for
24:    $C = m! / \text{div}$  ▷ set  $C = \binom{m}{k_1, \dots, k_n}$ 
25: end function

```

Computing the number of tensor indices in an index class that will contribute to a given entry of the output vector is a variation on Property 3.2. Consider an index class that contributes to the j^{th} entry of the output vector (i.e., an index class whose index representation includes an index j). Let $[k_1, k_2, \dots, k_n]$ be the monomial representation, so that $k_j > 0$. In the context of assigning m balls to n bins with appropriate multiplicities, we can assign the first ball to the j^{th} bin (enforcing that the tensor index starts with j). Then we have $m - 1$ more balls to assign to the n bins, but only $k_j - 1$ more will be assigned to the j^{th} bin. Using the approach given in the proof of Property 3.2, we see that the number of tensor indices that will contribute the same value to the j^{th} element is given by the multinomial coefficient

$$\binom{m-1}{k_1, \dots, k_j-1, \dots, k_n}.$$

Now we can rewrite Equation 3.3 as

$$(\mathcal{A}\mathbf{x}^{m-1})_j = \sum_{\substack{I \in \mathcal{I}^{[m,n]} \\ k_j > 0}} \binom{m-1}{k_1, \dots, k_j-1, \dots, k_n} a_{i_1 \cdots i_m} x_1^{k_1} \cdots x_j^{k_j-1} \cdots x_n^{k_n} \quad (3.4)$$

where $\mathcal{J}^{[m,n]}$ is the set of index classes for a symmetric tensor in $\mathbb{R}^{[m,n]}$, and $[k_1, \dots, k_n]$ and $[i_1, \dots, i_m]$ are the monomial and index representations of the index class I , respectively. Equation 3.4 yields Algorithm 3.

Algorithm 3 Compute $\mathbf{y} = \mathcal{A}\mathbf{x}^{m-1}$ via Equation 3.4, where $\mathcal{A} \in \mathbb{R}^{[m,n]}$ is symmetric, and $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$

Require: \mathcal{A} stores the unique entries of symmetric tensor \mathcal{A} in lexicographic order

```

1: function  $\mathbf{y} = \text{SYMMETRICTENSORVECTORMULTIPLY1}(\mathcal{A}, \mathbf{x})$ 
2:    $\mathbf{y} = 0$ 
3:    $I = [1, \dots, 1]$  ▷ use index representation (length  $m$ )
4:   for  $j = 1$  to  $\binom{m+n-1}{m}$  do ▷ iterate over unique tensor entries
5:     for unique  $i \in I$  do ▷ skip repeated indices in  $I$ 
6:        $\hat{\mathbf{x}} = x_{I_1} \cdot x_{I_2} \cdots x_{I_m} / x_i$  ▷ compute monomial value (excluding  $x_i$ )
7:        $C = \text{NUMOCC1}(I, i)$  ▷ compute number of occurrences
8:        $y_i = y_i + C \cdot \mathcal{A}_j \cdot \hat{\mathbf{x}}$  ▷ accumulate sum
9:     end for
10:     $I = \text{UPDATEINDEX}(I)$  ▷ See Algorithm 4
11:  end for
12: end function
```

Require: I has length m with entries in nondecreasing order

```

13: function  $C = \text{NUMOCC1}(I, i)$ 
14:    $\text{div} = 1$  ▷ divisor of  $\binom{m-1}{k_1, \dots, k_{i-1}, \dots, k_n}$ 
15:    $\text{curr} = -1$  ▷ current index value
16:    $\text{mult} = -1$  ▷ multiplicity of current index value
17:   for  $j = 1$  to  $m$  do
18:     if  $j \neq \text{first index of } i \text{ in } I$  then ▷ ignore one occurrence of  $i$ 
19:       if  $I_j \neq \text{curr}$  then
20:          $\text{mult} = 1$ 
21:          $\text{curr} = I_j$ 
22:       else ▷ repeated index
23:          $\text{mult} = \text{mult} + 1$ 
24:          $\text{div} = \text{div} \cdot \text{mult}$  ▷ only update divisor if  $\text{mult} > 1$ 
25:       end if
26:     end if
27:   end for
28:    $C = (m-1)! / \text{div}$  ▷ set  $C = \binom{m-1}{k_1, \dots, k_{i-1}, \dots, k_n}$ 
29: end function
```

3.2.3. Index array calculations. We can compute the index representation of an index class quickly by exploiting the lexicographic ordering and computing each index representation from the previous one. That is, given any index representation we want to compute the next larger index representation in the lexicographic order, under the conditions that the indices within the index representation are nondecreasing and range between 1 and n .

To find the next representation, we seek to increment the least significant possible index (i.e., the rightmost index not equal to n). In the example given in Table 3.1, the successor of $[1, 1, 1]$ is $[1, 1, 2]$ (the last index is incremented). More generally, suppose the k^{th} index is

the least significant index not equal to n , so that the index class is $[i_1, \dots, i_k, n, \dots, n]^2$. Thus, this is the largest representation with prefix $[i_1, \dots, i_k, \dots]$, so the successor must have prefix $[i_1, \dots, i_k + 1, \dots]$. The smallest such representation that satisfies the nondecreasing condition is

$$[i_1, \dots, i_k + 1, i_k + 1, \dots, i_k + 1].$$

For example, again from Table 3.1, the successor of $[2, 4, 4]$ is $[3, 3, 3]$. See Algorithm 4 for the implementation. In this way, we can store index information in an array of m integers, and under the lexicographic ordering, and updating the index information for each term in the summation requires $O(m)$ operations.

Algorithm 4 Update index representation of unique entry in symmetric tensor $\mathcal{A} \in \mathbb{R}^{[m,n]}$

Require: I has length m with entries in nondecreasing order

```

1: function UPDATEINDEX( $I$ )
2:    $j = m$ 
3:   while  $I_j == n$  ▷ find least significant index  $\neq n$ 
4:      $j = j - 1$ 
5:   end while
6:    $I_j = I_j + 1$  ▷ increment least significant index  $\neq n$ 
7:   for  $k = j + 1$  to  $m$  do ▷ update less significant indices
8:      $I_k = I_j$ 
9:   end for
10: end function

```

Ensure: I is the successor in lexicographic ordering (restricted to nondecreasing)

3.2.4. Computing number of occurrences. The number of occurrences of each index class is given by a multinomial coefficient in terms of the monomial representation of the index class. Since we store the index representation and not the monomial representation, we compute the multinomial coefficient implicitly. We can do this by computing the denominator with one pass over the array storing the index representation. The numerator is constant over all index classes and can be precomputed (either $m!$ or $(m - 1)!$ for the two computational kernels).

In the case of computing $\mathcal{A}\mathbf{x}^m$, the task is to compute for each index class the product $k_1! \cdots k_n!$, where $[k_1, \dots, k_n]$ is the monomial representation which is not stored explicitly. Note that k_i is the number of occurrences of index i in the index representation which is stored in memory. Since the index representation is nondecreasing, repeated occurrences of an index will be contiguous. Thus, as we pass over the index array, we can multiply the accumulated product by 1 for the first occurrence of an index, by 2 for the second occurrence, and so on. For example, given the index representation $[1, 2, 2, 5, 5, 5, 5]$, the accumulated product will be $1 \cdot 1 \cdot 2 \cdot 1 \cdot 2 \cdot 3 \cdot 4 = 1! \cdot 2! \cdot 4!$. This approach yields the function NumOcc0 in Algorithm 2.

In the case of computing $\mathcal{A}\mathbf{x}^{m-1}$, we take the same approach to compute the denominator, but we ignore one occurrence of the index corresponding to the entry of the output vector being computed. Following the preceding example, in the case of computing the 5th element of $\mathcal{A}\mathbf{x}^{m-1}$, the index representation $[1, 2, 2, 5, 5, 5, 5]$ would yield to the accumulated product $1 \cdot 1 \cdot 2 \cdot 1 \cdot 2 \cdot 3 = 1! \cdot 2! \cdot 3!$. This approach yields the function NumOcc1 in Algorithm 3.

²Note that there may be no instances of index n in the index class, in which case $k = m$, the index class is $[i_1, \dots, i_k]$, and the successor is $[i_1, \dots, i_k + 1]$.

In order to avoid redundant computation (at the expense of extra storage), we can pre-compute the multinomial coefficient $\binom{m}{k_1, k_2, \dots, k_n}$ for each index class. This is the coefficient used in the computation of $\mathcal{A}\mathbf{x}^m$, and the coefficients needed in the computation of $\mathcal{A}\mathbf{x}^{m-1}$ can be obtained by dividing the stored value by m and multiplying by k_j for appropriate j .

3.2.5. Computational costs. All the computations in the main loop of Algorithm 2 are done in $O(m)$ operations (floating point and otherwise). Thus, the computational complexity of computing $\mathcal{A}\mathbf{x}^m$ is $O\left(m \cdot \frac{n^m}{m!}\right) = O\left(\frac{n^m}{(m-1)!}\right)$.

There are nested loops in Algorithm 3, and the inner loop requires m iterations in the worst case. All the computations in the inner loop are done in $O(m)$ operations (floating point and otherwise), so the computational complexity of computing $\mathcal{A}\mathbf{x}^{m-1}$ is $O\left(m^2 \cdot \frac{n^m}{m!}\right) = O\left(\frac{mn^m}{(m-1)!}\right)$.

4. GPU Computing Overview. Graphics processing units (GPUs) were originally developed and optimized to offload and accelerate graphics rendering computations from the more general purpose microprocessor or “central processing unit” (CPU) on a host computer. Graphics processing consists largely of data parallel computations, and GPU hardware is designed to exploit this data parallelism via single instruction/multiple data (SIMD) instructions. GPUs also exploit instruction level parallelism: instruction streams for several threads of execution are pipelined in order to hide the latency of memory operations for each thread (this requires that the threads be mutually independent).

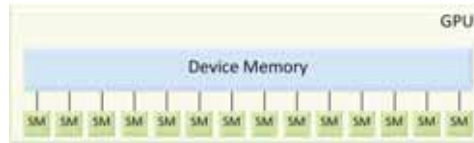
GPU architecture is rapidly developing to meet the demands of new applications and users. Many of these applications require high graphics rendering performance, but a growing number of users are interested in exploiting the computing power of GPUs for many other purposes including scientific computing. To this end, nVidia has invested in the development of Compute Unified Device Architecture (CUDA) which is used for general purpose programming of GPUs. Most programmers use CUDA as an extension of the C language which gives access to a set of virtual instructions for accessing the memory spaces and functional units on a GPU.

Along with making CUDA freely available, nVidia also offers a software development kit including programming guides, example programs, and other documentation for programmers. Much of the information in the following sections is available in more detail in the CUDA documentation, particularly [5, 6].

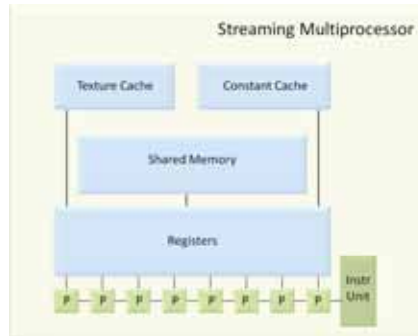
4.1. Physical Hardware Model. Both the computational units and the memory hierarchy on GPUs are fundamentally different from CPU architectures. See Figure 4.1 for a graphical representation of the physical hardware model.

Computational Units. The functional units on a GPU are organized into groups which concurrently execute SIMD instructions. In nVidia terminology, each functional unit is known as a “processor” or “core”, and each group of processors resides on a “streaming multiprocessor.” On the GeForce 9800 GT used in our experiments, there are 14 multiprocessors (see Figure 4.1(a)), each with 8 processors (see Figure 4.1(b)). Thus eight operations can simultaneously execute the same instruction on different data on a multiprocessor. The GPU we used is capable of only single precision floating point operations, but newer models can execute double precision operations.

Memory Hierarchy. GPUs have a complicated memory hierarchy with several different physical and logical memory spaces. Note that the memory hierarchy discussed here is only representative of nVidia GPUs of Compute Capability 1.x; newer architectures of Compute Capability 2.x have fundamental differences. The largest memory is known as “device memory” and is accessible to all multiprocessors on the GPU (see Figure 4.1(a)). It is also



(a) GPU card with device memory and set of streaming multiprocessors (SM). Memory on each SM shown in (b).



(b) Streaming multiprocessor with on-chip memories and SIMD functional units (P). Each SM has access to device memory as shown in (a).

FIG. 4.1. GPU Hardware Model

accessible from the host device (CPU) and is the means through which the CPU and GPU communicate data. Except for “integrated” cards, this memory resides on the graphics card itself. The memory access latency for device memory to one of the GPU’s computational units is two orders of magnitude greater than the latency of the on-chip memory.

There are four types of on-chip memory: registers, shared memory, constant cache, and texture cache (see Figure 4.1(b)). The set of registers, or “register file,” is relatively large but must be divided up among all threads resident on the multiprocessor; it has the smallest memory access latency (one or two cycles). The shared memory is the next fastest memory. It is smaller than the register file but can be shared among threads in a thread block. Shared memory can be dynamically allocated and can be used as a local store (i.e. there is no hardware-managed caching system).

Some of device memory can be statically allocated as “constant” memory, and accesses to constant memory will be cached by the hardware. Constant memory is read-only for a given GPU kernel function but can be written by the host CPU between kernel calls. A “texture” can be bound to an array in device memory such that the result of a texture “fetch” will be cached. The texture caches on a GPU are shared by two or three multiprocessors. The texture caching system is designed to exploit 2D spatial locality, and texture fetches include other features designed to improve the performance of certain relevant graphics operations. See Table 4.1 for the sizes of the on-chip memory for the GeForce 9800 GT card.

4.2. CUDA Programming Model. The simplest CUDA programming model treats the GPU as a coprocessor to the host CPU. That is, a single thread of execution works on the CPU sequentially until it calls a “kernel” function on the GPU which is run by many CUDA threads in parallel, and after the kernel returns, the single CPU thread resumes execution until it calls another kernel or terminates. Multiple CPU threads can be used in order to overlap CPU

Register file	8192 registers
Shared memory	16 KB
Texture cache	6-8 KB
Constant cache	8 KB

TABLE 4.1

On-chip memory sizes per multiprocessor for GeForce 9800 GT (Compute Capability 1.1)

and GPU computation, but we only consider one CPU thread in this work. Kernel functions may call other functions to be run on the GPU (which will also run in parallel); these other functions cannot be called from host code. When a kernel function is launched from the host code, the host specifies the number of thread blocks, the number of threads per block, and optionally the amount of shared memory to allocate to each thread block (all of which can be determined at run time).

Thread blocks are groups of threads which are all run on the same multiprocessor. They have a common memory space residing in the physical shared memory through which the threads can communicate and synchronize. Thread blocks are logical entities and the number of threads per block is unrestricted up to a certain maximum; however, threads are physically grouped into warps (the physical unit of SIMD instructions) during execution, so the number of threads per block should be a multiple of the warp size (typically 32).

The logical memory hierarchy is tightly coupled to the physical memory. Registers are local to threads, shared memory is restricted to threads within a thread block, and global memory (which resides in “device” memory) is accessible by all threads and by the host code. Communication between thread blocks using global memory is possible but rare because thread blocks may be scheduled on any multiprocessor in any order. Textures and constant memory are also globally accessible and are read-only; textures are accessed via special texture fetches. Another memory space known as “local” memory is logically local to each thread, but the name is misleading because local memory physically resides in device memory. In general, local memory is used to handle register spilling.

5. Detecting Nerve Fiber Direction in the Brain. We next discuss an application well-suited for computation on a GPU. It involves many independent problems that can be solved in parallel, and each problem involves an amount of data that is small enough to reside in the on-chip memories of the multiprocessors.

Diffusion-weighted magnetic resonance imaging (DW-MRI) is a tool used to detect nerve fibers in the brain. It is a non-invasive procedure that uses magnetic resonance to measure how quickly water diffuses in a certain direction. Water diffuses more quickly along the longitudinal axis of nerve fiber bundles than in any transverse or axial direction. DW-MRI measurements are taken from many different orientations for a discrete set of voxels in the brain. For each voxel, a diffusion function $D : \Sigma \rightarrow \mathbb{R}$ which maps an orientation to its rate of diffusion (here Σ denotes the unit sphere in \mathbb{R}^3) is approximated using the measurement data. For a unit vector \mathbf{g} , $D(\mathbf{g})$ is known as the “apparent diffusion coefficient” (ADC) [10].

When a voxel includes only one fiber orientation, the longitudinal direction should (globally) maximize D (it will exhibit the largest ADC). When a voxel includes more than one fiber orientation (in the case of crossing fibers), each fiber orientation should correspond to a local maximum of D .

According to [7, 8, 10], a common way to approximate the diffusion function is as a finite sum of spherical harmonic functions (which form a basis for complex functions on the unit sphere). The 2nd order series (with 6 terms) corresponds to a quadratic form

$$D(\mathbf{g}) \approx \mathbf{g}^T \mathbf{M} \mathbf{g}$$

where \mathbf{M} is a symmetric positive definite 3×3 matrix. In this case, at least six measurements are required to determine the unique entries in the matrix \mathbf{M} (or the six coefficients of the first spherical harmonic functions). In the case of a voxel with one principal fiber orientation, this approach is usually sufficient for resolving the correct orientation. However, in the case of fiber crossings or other complications such as bending or fanning fiber bundles, the approximation is often unable to resolve the fiber directions.

In order to handle such cases, more accurate measurements and approximations are necessary. The approach is to use higher order spherical harmonic series approximations which can be represented not as quadratic forms, but more generally as homogeneous forms. The homogeneous forms correspond to higher order tensors:

$$D(\mathbf{g}) \approx \mathcal{A}\mathbf{g}^m$$

for some symmetric tensor $\mathcal{A} \in \mathbb{R}^{[m,3]}$. Note that m must be even since $D(\mathbf{g})$ is a positive physical quantity for all \mathbf{g} (if m is odd then $\mathcal{A}(-\mathbf{g})^m = -\mathcal{A}\mathbf{g}^m$). More DW-MRI measurements are required to determine the greater degrees of freedom in tensors of order $m > 2$, and the higher order polynomial can better approximate the true diffusion function. Orders $m = 4$ and $m = 6$ are most commonly used ($m = 8$ requires 120 measurements). The correspondence between coefficients of spherical harmonic functions with the entries in the associated symmetric tensor are given in [10].

As described in [3], the critical points of the function $f(\mathbf{x}) = \mathcal{A}\mathbf{x}^m$ and their function values are exactly the eigenpairs of the tensor \mathcal{A} (satisfying Equation 2.2). Thus, in order to determine the principal fiber orientations in a given voxel, we can compute the principal eigenvectors of the associated tensor.

Note that specific instances of Properties 3.1 and 3.2 for $n = 3$ appear in the DW-MRI literature. See Equations 17 and 19 in [7], for example.

6. Implementation Details. The computation problem for the nerve fiber data is to take as input a three dimensional array of symmetric tensors and output one or more eigenpairs for each tensor. The three dimensional array corresponds to the set of voxels which discretize the volume of a brain. The entries of each tensor correspond to the coefficients of the homogeneous polynomial which approximates the diffusion function for a given voxel. The eigenpairs which define local maxima of the approximate diffusion function correspond to principal nerve fiber directions within the voxel.

In order to find multiple eigenpairs, Algorithm 1 must be executed with different starting vectors. Because there is not much theory to direct the choice of starting vectors to find all eigenpairs corresponding to local maxima, we use many randomly chosen starting vectors in order to get reasonable coverage of the unit sphere. We choose random vectors by independently selecting each vector entry uniformly from $[-1, 1]$ and then normalizing. Alternatively, one could use a deterministic approach and pick starting vectors evenly spaced about the sphere.

The computational problem consists of executing Algorithm 1 with many different tensors and many different starting vectors each. Since the voxel size for DW-MRI is on the order of one cubic millimeter, the number of voxels in a data set for a human brain can be in the millions. In order to cover the sphere, we use somewhere between 32 and 128 starting vectors for each tensor. With this much inherent parallelism in the problem, we can easily saturate the computational units on a GPU. The main data structures involved in the computation include the unique entries of each tensor, an array of randomly generated starting vectors, an array of output eigenvectors, and an array of output eigenvalues.

6.1. Synthetic Test Set. We experimented with a synthetic test set provided by the Scientific Computing and Imaging Institute at the University of Utah. It consists of 1024 tensors

corresponding to a 2D array of voxels which includes some with one and some with two principal fiber directions. Each tensor is 4th order, so each has 81 total entries with 15 unique values. We chose to use 128 starting vectors for each tensor in the hope of reasonably covering the sphere in \mathbb{R}^3 and also because it is a multiple of 32, the physical warp size on the GPU. We used a shift of $\alpha = 0$ as it yielded correct results for the tensors in this synthetic set. Note that $\alpha = 0$ implies that SS-HOPM is the same algorithm as the one given in [2, 4]. Although the performance of the implementation will not vary much with α , choosing an appropriate shift for real data will balance a tradeoff between guarantees of convergence and time-to-completion. To find local maxima, a nonnegative shift must be used.

6.2. Thread Organization. Because of the number of independent problems, we are able to map the computation to the GPU in a straightforward way with minimal synchronization. We organize the CUDA threads in the following way: assign a thread block to each tensor and assign each thread in a thread block to a different starting vector. Since the number of starting vectors is greater than the warp size, each thread block will utilize all the processors on its multiprocessor. Similarly, as long as the number of tensors is at least 50 or so, all of the multiprocessors will be utilized with three or four thread blocks each (multiple thread blocks are necessary to fill the instruction pipelines).

6.3. Data Structures. Because of the small size of the tensors and vectors in this problem, we can fit all the data for each thread block in the on-chip memory and minimize the accesses to device memory. Let T be the number of tensors, U be the number of unique entries in each tensor, and V be the number of starting vectors. Recall that for this problem, $m = 4$, $n = 3$, $T = 1024$, $U = 15$, and $V = 128$. For real data, we expect T to grow into the millions but the rest of the parameters will remain constant, though V could be varied experimentally. The tensor data is of size $T \cdot U$, the array of starting vectors is $n \times V$, the array of output eigenvectors is $n \times (T \cdot V)$, and the array of output eigenvalues is of size $T \cdot V$. Note that every thread block can use the same set of starting vectors, but each has its own set of output vectors.

In addition to the main data structures, we pre-compute and store the index and multinomial coefficient information required in Algorithms 2 and 3. The index information is stored as an array of size $m \times U$ and can be shared by all threads. We store the multinomial coefficient $\binom{m}{k_1, \dots, k_n}$ for each unique tensor value, where $[k_1, \dots, k_n]$ is the monomial representation of the index class of the unique entry. In this way, finding the number of occurrences of an entry in Algorithm 2 is just a look-up, and computing the related multinomial coefficients used in Algorithm 3, which are of the form $\binom{m-1}{k_1, \dots, k_{i-1}, \dots, k_n}$ for some i , can be done by reading the stored value, multiplying by k_i and dividing by m .³ Thus the array of multinomial coefficients is of size U . All threads can share this information.

6.4. Memory Management. We use both the shared memory and constant cache to minimize the memory accesses to device memory. Because the index array and multinomial coefficients are read only and can be shared by all the threads in the computation, we designate them as constant memory which resides in global (device) memory. However, because that information can fit into the constant cache of each multiprocessor, they will be read from device memory to the cache only once per multiprocessor for the entire computation. Because the tensor entries can be shared by the threads within one thread block, we store them in the shared memory. In this way, the tensor entries are read from device memory to the on-chip shared memory only once per thread block.

³One might consider storing the “coefficient” $\binom{m-1}{k_1, \dots, k_n}$ so that only one multiply is needed to update the stored value for each kernel, but note that this value is not an integer in general.

```

y1 = Avals[0]      * x1 * x1 * x1 + \
      Avals[1] * 3 * x1 * x1 * x2 + \
      Avals[2] * 3 * x1 * x1 * x3 + \
      Avals[3] * 3 * x1 * x2 * x2 + \
      Avals[4] * 6 * x1 * x2 * x3 + \
      Avals[5] * 3 * x1 * x3 * x3 + \
      Avals[6]      * x2 * x2 * x2 + \
      Avals[7] * 3 * x2 * x2 * x3 + \
      Avals[8] * 3 * x2 * x3 * x3 + \
      Avals[9]      * x3 * x3 * x3;

```

FIG. 6.1. *Unrolled computation of the first entry of the vector $\mathcal{A}\mathbf{x}^{m-1}$, for $\mathcal{A} \in \mathbb{R}^{[4,3]}$. The variables $x1$, $x2$, $x3$ are register variables which store the input vector and the `Avals` array is in shared memory.*

Finally, we store the input and output vectors, which are private to each thread, in shared memory. Although this data will not be shared with other threads in the thread block, we use the shared memory because it is the only on-chip memory that can be dynamically allocated and overwritten. There are two main drawbacks from using shared memory this way. First, allocating $2n$ words of shared memory per thread requires a lot of memory per thread block, and since the physical shared memory is shared by all thread blocks on a multiprocessor, fewer thread blocks can be scheduled simultaneously on each multiprocessor. The amount of oversubscription (known as “occupancy” in nVidia’s terminology) allows for pipelining instruction streams and hiding memory latency. Second, the register file is faster to access than shared memory. Since the number of thread blocks per multiprocessor is limited by the shared memory requirements, the size of the register file is not being exploited.

6.5. Loop Unrolling. For a given order and dimension, we can unroll the loops within the two main computational kernels. This enables us to exploit the register file for storing the input and output vectors by statically allocating register variables corresponding to input and output vector entries. Not only does this expose instruction-level parallelism to the compiler, it also removes the indirection in accessing input and output vector entries. This is possible for small problems, but to scale to larger problems we would need a blocked approach. See Figure 6.1 for an example of an unrolled loop in the case $m = 4$ and $n = 3$. The `Avals` array stores the unique tensor entries in lexicographic order, the input vector entries are stored in static variables, and the multinomial coefficients are stored as constants in the instruction stream. In this case, the number of terms in the summation for $\mathcal{A}\mathbf{x}^m$ is 15, and each of the three summations for the entries of the output vector $\mathcal{A}\mathbf{x}^{m-1}$ have 10 terms.

Without unrolling the loops, each access to an input or output vector requires two memory operations. For example, if the index information is stored in an array called `index`, then accessing entries to the input vector \mathbf{x} take the form $\mathbf{x}[\text{index}[\mathbf{k}]]$. This indirection prevents the compiler from pipelining instructions within one thread and degrades performance even if `index` and \mathbf{x} are both on-chip (see Section 6.6).

Note that the arithmetic intensity (ratio of flops to bytes involved in the computations) is high for both kernels. In the case $m = 4$ and $n = 3$, there are 15 unique tensor entries and two vectors each with three entries, so the number of bytes in single precision is 84 while the number of flops in computing $\mathcal{A}\mathbf{x}^{m-1}$ is 140. Another possible optimization would be to use common subexpression elimination on the unrolled summations. For example, the code shown in Figure 6.1 computes x_1^2 three times.

6.6. Results. The processor used for these results is a quad-core Intel Bloomfield (Core i7). The GPU used is an nVidia GeForce 9800 GT which nVidia classifies as Compute Capa-

(a) Flop rates in Gflops/s and speedup of loop unrolling

	General	Unrolled	Unrolled speedup
CPU seq	0.24	1.86	7.86
CPU par	0.92	6.85	7.41
GPU	5.95	131.73	22.15

(b) Relative performance, normalized to sequential implementations

	General	Unrolled
CPU seq	1.00	1.00
CPU par	3.90	3.67
GPU	25.08	70.66

TABLE 6.1

Performance results for six different implementations on all 1024 tensors

bility 1.1. The parallel CPU code was run with four threads using OpenMP. All computations were done in single precision (the only precision available on GPUs of Compute Capability 1.1), and we use 128 starting vectors in all cases.

We report on six different implementations. We benchmarked a completely sequential implementation, using one core on the quad-core CPU; a parallel CPU implementation, using all four cores of the processor; and our GPU implementation. In each case, we benchmarked both the general version of the code and the loop-unrolled version which is specialized to tensors of order 4 and dimension 3. Note that no memory hierarchy optimizations were used in the CPU implementations.

Table 6.1 shows the performance results for all six implementations computing the eigenpairs for all 1024 tensors. Table 6.1(a) shows the absolute performance and gives the speedup observed for each implementation by unrolling the loops. Comparing the unrolled code to the general implementations, we see that unrolling yields over $7\times$ speedup for both CPU implementations and a $22\times$ speedup for the GPU implementation. In Table 6.1(b) the relative performance values are normalized to the sequential CPU implementations to show parallel speedups. We observe that the GPU implementation achieves a speedup of $20\times$ over the parallelized CPU implementation. Although the CPU implementation was not optimized for the memory hierarchy, we believe that because of the large number of independent problems in this application, the GPU implementation will outperform the best multi-core implementation for this test set. Future research will explore which architecture is better suited for computing eigenpairs of larger tensors or tensor applications with less inherent parallelism. In either case, developing high performing code for general orders and dimensions will require an efficient blocking strategy to allow for loop unrolling and the use of register variables.

Figure 6.2 shows performance results for four different implementations for subsets of the 1024 tensors in our test set. Note that the loop unrolling makes a significant difference in the GPU performance for all problem sizes. Because of the independence of the tensor eigenproblems, the parallel CPU implementation requires only a slight modification of the sequential code using OpenMP pragmas and we observe close to perfect parallel scaling for sufficiently large problems.

7. Conclusions. In this paper we present an implementation of SS-HOPM targeted for a GPU. We describe how to save both storage and computation in the two main computational kernels of the algorithm, and for the case of solving many small tensor eigenproblems we show how to map the computation onto a GPU. For our experimental data set, we achieved

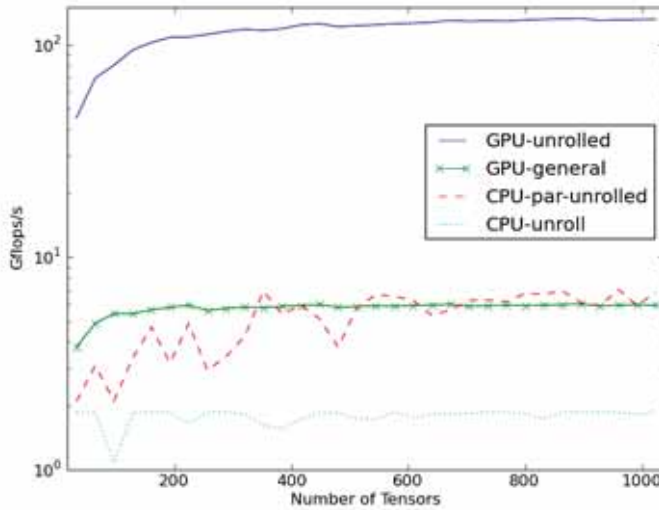


FIG. 6.2. Performance results for running SS-HOPM on sets of 4^{th} order 3-dimensional tensors with 128 starting vectors each. Note the y-axis is a log scale.

parallel speedups of up to $70\times$ over a sequential code using the same low-level optimizations (but no memory hierarchy optimizations).

We believe that the techniques for exploiting symmetry may be extended to other computations involving symmetric tensors, but many open questions remain about how to write sequential or parallel implementations of the computational kernels that scale to higher order and higher dimension tensors. We are also interested in how to map these computations onto different computing platforms, including more recent GPUs which offer fundamentally different hardware features.

Acknowledgments. We would like to thank Fangxiang Jiao, Yaniv Gur, and Chris Johnson of the Scientific Computing and Imaging Institute at the University of Utah for the motivating application and for providing the sample data.

REFERENCES

- [1] P. COMON, G. GOLUB, L.-H. LIM, AND B. MOURRAIN, *Symmetric tensors and symmetric tensor rank*, SCCM Technical Report 06-02, Stanford University, 2006.
- [2] E. KOFIDIS AND P. A. REGALIA, *On the best rank-1 approximation of higher-order supersymmetric tensors*, SIAM Journal on Matrix Analysis and Applications, 23 (2002), pp. 863–884.
- [3] T. G. KOLDA AND J. R. MAYO, *Shifted power method for computing tensor eigenpairs*. arXiv:1007.1267v1 [math.NA], July 2010.
- [4] L. D. LATHAUWER, B. D. MOOR, AND J. VANDEWALLE, *On the best rank-1 and rank- $(r_{\text{sub } 1}, r_{\text{sub } 2}, \dots, r_{\text{sub } n})$ approximation of higher-order tensors*, SIAM Journal on Matrix Analysis and Applications, 21 (2000), pp. 1324–1342.
- [5] NVIDIA, *NVIDIA CUDA programming guide version 3.0*.
- [6] ———, *PTX: Parallel thread execution ISA version 2.0*.
- [7] E. ÖZARSLAN AND T. H. MARECI, *Generalized diffusion tensor imaging and analytical relationships between diffusion tensor imaging and high angular resolution diffusion imaging*, Magnetic Resonance in Medicine, 50 (2003), pp. 955–965.
- [8] ———, *Generalized scalar measures for diffusion mri using trace, variance, and entropy*, Magnetic Resonance in Medicine, 53 (2005), pp. 866–876.

- [9] K. H. ROSEN, *Discrete mathematics and its applications (2nd ed.)*, McGraw-Hill, Inc., New York, NY, USA, 1991.
- [10] T. SCHULTZ AND H.-P. SEIDEL, *Estimating crossing fibers: A tensor decomposition approach*, IEEE Transactions on Visualization and Computer Graphics, 14 (2008), pp. 1635–1642.

Uncertainty Quantification and Sensitivity Analysis

Uncertainty quantification and sensitivity analysis attempt to quantify the effect of variation in model parameters has on a physical model. The algorithms for computing these effects can be computationally intensive and require the development of novel numerical methods for efficient solution. Even given an efficient algorithm, the development and application appropriate methodologies for sensitivity analysis is still an area of active research. The articles in this section touch on both efficient solution methods and methodological application and development.

Tipireddy et al compare a number of preconditioners for stochastic Galerkin methods. The performance of these methods are compared against results for nonintrusive stochastic Galerkin methods. *Miller et al.* apply the stochastic collocation method to the drift-diffusion equations for semiconductor modeling. Using the results of the collocation method, a sensitivity analysis is performed gaining insight into the global sensitivity of the response function to the parameters. *Ahuja et al.* develop Krylov recycling methods that appropriate for rapidly converging linear systems. The performance of these methods is demonstrated on both ice modeling problems and embedded uncertainty quantification methods. *Blass and Romero* develop a method to analyze the stability of a stochastically forced ordinary differential equation. The approach utilizes analytic expressions for the eigenvalues and functions of a second order differential operator to determine the stability. *Proctor et al.* consider a sensitivity analysis for a linear neutronics model for nuclear reactors. This work compares the performance of adjoint-based local and global sensitivity analysis.

E.C. Cyr
S.S. Collis

December 17, 2010

A COMPARISON OF SOLUTION METHODS FOR STOCHASTIC PARTIAL DIFFERENTIAL EQUATIONS

RAMAKRISHNA TIPIREDDY[§], ERIC T. PHIPPS[¶], AND ROGER G. GHANEM^{||}

Abstract. Several solution methods for stochastic Galerkin discretization of partial differential equations (PDEs) with random input data are compared. Less intrusive approaches based on Jacobi and Gauss-Seidel mean iterations are compared with more intrusive Krylov-based approaches. A set of preconditioners for the Krylov-based iterative methods to accelerate convergence are also examined, including mean-based, Gauss-Seidel, approximate Gauss-Seidel and approximate Jacobi mean preconditioners. All of these methods are compared to a non-intrusive stochastic collocation approach applied to a canonical stochastic diffusion problem. For this problem, the Krylov-based approach using approximate Gauss-Seidel and Jacobi preconditioners is found to be most effective. Sandia's Trilinos software is used to implement all the above algorithms.

1. Introduction. Real life physical problems are often modeled as partial differential equations (PDEs) where input data are treated as random to represent uncertainty in this data. Monte Carlo techniques are popular methods to solve these problems as they only require solutions to the PDE for a given set of realizations of the input data. More recently however, the stochastic finite element method [6, 3] has become a popular choice for solving these problems because of its advantages over Monte Carlo methods. These methods compute statistical properties of the solution more efficiently than Monte Carlo methods.

Stochastic finite element methods are either intrusive stochastic Galerkin methods ([13, 11, 4, 8]) or non-intrusive stochastic collocation methods ([15, 18, 10, 2]). Both exploit solution regularity to achieve higher convergence rates than Monte Carlo methods. The first approach translates the stochastic PDE into a coupled set of deterministic PDEs while the second samples the stochastic PDE at a predetermined set of collocation points, resulting in a set of uncoupled deterministic PDEs. The solution at these collocation points is then used to interpolate the solution in the entire random input domain. Extending legacy software to support stochastic collocation methods is simpler than supporting SGMs. Moreover, intrusive SGMs require specialized linear solvers. However, the resulting set of PDEs in the stochastic Galerkin system is much smaller in number than that in the collocation method. For a canonical random diffusion problem, it is shown [9] that SGM using iterative Krylov-based linear solvers and mean-based preconditioning [12] is more efficient than the non-intrusive sparse grid collocation method.

While the stochastic Galerkin method is often considered to be a fully intrusive method, there are in fact a variety of solver approaches for the stochastic Galerkin method that are less intrusive. In this work, less intrusive Gauss-Seidel and Jacobi mean solver methods are compared to more intrusive Krylov-based techniques. We consider these methods to be less intrusive than the Krylov-based methods as they allow reuse of existing deterministic solvers. Moreover preconditioning techniques for Krylov-based methods based on Gauss-Seidel and Jacobi ideas are also explored and compared to traditional mean-based preconditioning. All of these techniques are then compared to the non-intrusive stochastic collocation method applied to a canonical random diffusion problem. These comparisons demonstrate a trade-off in computational cost versus intrusiveness with the Krylov-based methods using an approximate Gauss-Seidel or Jacobi mean preconditioner being the most efficient.

This paper is organized as follows. In section 2, the model random diffusion problem is formulated. Two models of the input random field are developed in section 3 which dictate

[§]Department of civil engineering at University of Southern California, tipiredd@usc.edu

[¶]Sandia National Laboratories, ethipp@sandia.gov

^{||}Department of civil engineering at University of Southern California, ghanem@usc.edu

very different behavior for the stochastic solution methods considered next. Section 4 describes the stochastic Galerkin method, and various solver and preconditioning methods are introduced. The sparse grid collocation method is then reviewed in section 5. In section 6, numerical experiments are carried out to compare the efficiency of the various solver and preconditioning methods that have been introduced. Finally section 7 provides the concluding remarks.

2. Problem Statement. In this work a stochastic steady state elliptic diffusion equation with zero Dirichlet boundary conditions [9] is used as a test problem for various stochastic PDE solution methods. Let D be an open subset of R^n (for this work we assume $n = 2$) and (Ω, Σ, P) be a complete probability space with sample space Ω , σ -algebra Σ and probability measure P . Assume $a(x, \omega) : D \times \Omega \rightarrow \mathbb{R}$ is a random field that is bounded and strictly positive, that is,

$$0 < a_l \leq a(x, \omega) \leq a_u < \infty \quad \text{a.e. in } D \times \Omega. \quad (2.1)$$

We wish to compute a random field $u(x, \omega) : D \times \Omega \rightarrow \mathbb{R}$, $u \in H^1(D) \otimes L_2(\Omega)$ such that the following holds P -almost surely (P -a.s.):

$$-\nabla \cdot (a(x, \omega) \nabla u(x, \omega)) = f(x, \omega) \quad \text{in } D \times \Omega, \quad (2.2)$$

$$u(x, \omega) = 0 \quad \text{on } \partial D \times \Omega. \quad (2.3)$$

Let $H_0^1(D)$ be the subspace of the Sobolev space $H^1(D)$ that vanishes on the boundary ∂D and is equipped with the norm $\|u\|_{H_0^1(D)} = [\int_D |\nabla u|^2 dx]^{\frac{1}{2}}$. Problem 2.2 can then be written in the following equivalent variational form [7]: find $u \in H_0^1(D) \otimes L_2(\Omega)$ such that

$$b(u, v) = l(v), \quad \forall v \in H_0^1(D) \otimes L_2(\Omega), \quad (2.4)$$

where, $b(u, v)$ is the continuous and coercive (from assumption 2.1) bilinear form given by

$$b(u, v) = E \left[\int_D a \nabla u \cdot \nabla v dx \right], \quad \forall u, v \in H_0^1(D) \otimes L_2(\Omega), \quad (2.5)$$

and $l(v)$ is the continuous bounded linear functional given by

$$l(v) = E \left[\int_D f v dx \right], \quad \forall v \in H_0^1(D) \otimes L_2(\Omega). \quad (2.6)$$

Here $E[\cdot]$ denotes mathematical expectation. From the Lax-Milgram lemma, Eq. 2.4 has unique a solution in $H_0^1(D) \otimes L_2(\Omega)$.

3. Input random field model. For computational purposes, the diffusion coefficient $a(x, \omega)$ must be discretized in both the spatial and stochastic domains. To this end, it is often approximated with a truncated series expansion that separates the spatial variable x from the stochastic variable ω resulting in a representation by a finite number of random variables. For this representation, second order information such as the covariance function of the random field is required. In the present problem, two cases of random field models are considered. In the first case, the random field is assumed to be uniformly distributed and is approximated through a truncated Karhunen-Loève expansion. In the second case, the random field is assumed to have a log-normal distribution, that is $a(x, \omega) = \exp(g(x, \omega))$ where $g(x, \omega)$ is a Gaussian random field, and is approximated by a truncated polynomial chaos expansion.

3.1. Karhunen-Loève expansion. Let $C(x_1, x_2) = E[a(x_1, \omega)a(x_2, \omega)]$ be the covariance function of the random field $a(x, \omega)$. Then a can be approximated through its truncated Karhunen-Loève (K-L) expansion [6] given by

$$a(x, \omega) \approx \tilde{a}(x, \xi(\omega)) = a_0(x) + \sum_{i=1}^M \sqrt{\lambda_i} a_i(x) \xi_i(\omega), \quad (3.1)$$

where $a_0(x)$ is the mean of the random field $a(x, \omega)$ and $\{(\lambda_i, a_i(x))\}_{i \geq 1}$ are solutions of the integral eigenvalue problem

$$\int_D C(x_1, x_2) a_i(x_2) dx_2 = \lambda_i a_i(x_1). \quad (3.2)$$

The eigenvalues λ_i are positive and non-increasing, and the eigenfunctions $a_i(x)$ are orthonormal, that is,

$$\int_D a_i(x) a_j(x) dx = \delta_{ij}, \quad (3.3)$$

where δ_{ij} is the Kronecker delta. In Eq. 3.1, $\{\xi_i\}_{i=1}^M$ are uncorrelated random variables with zero mean. As a first test-case, the diffusion coefficient $a(x, \omega)$ is modeled with an exponential covariance function

$$C(x_1, x_2) = \sigma^2 \exp(-\|x_1 - x_2\|_1 / L) \quad (3.4)$$

and uniformly distributed random variables $\xi_i(\omega)$. We further assume the random variables are independent.

3.2. Polynomial chaos expansion. The K-L expansion above approximates a random field by a linear combination of a finite set of random variables. To maintain positivity of the random field, such a representation is only appropriate if the random variables are bounded [16]. For unbounded random variables (e.g., log-normal) a nonlinear polynomial chaos representation is more appropriate. The polynomial chaos expansion [6, 17] is used to approximate a random field in terms of multi-variate orthogonal polynomials. Let $\xi = (\xi_1, \dots, \xi_M)^T$ be the random variables from a truncated K-L expansion of a given random field $g(x, \omega)$, that is

$$g(x, \omega) \approx \tilde{g}(x, \xi(\omega)) = g_0(x) + \sum_{i=1}^M \sqrt{\lambda_i} g_i(x) \xi_i(\omega). \quad (3.5)$$

Assume $a(x, \omega)$ is then given by a nonlinear transformation of $g(x, \omega)$. Then $a(x, \omega)$ can be represented through nonlinear functionals of the random variables $\xi_i(\omega)$. It has been shown in [17, 6] that this functional dependence can be expanded in terms of multi-dimensional orthogonal polynomials, called polynomial chaos, as

$$a(x, \omega) = \hat{a}_0(x) + \sum_{i_1=1}^{\infty} \hat{a}_{i_1}(x) \Gamma_1(\xi_{i_1}(\omega)) + \sum_{i_1=1}^{\infty} \sum_{i_2=1}^{i_1} \hat{a}_{i_1 i_2}(x) \Gamma_2(\xi_{i_1}(\omega), \xi_{i_2}(\omega)) + \dots \quad (3.6)$$

where $\Gamma_n(\xi_{i_1}, \dots, \xi_{i_n})$ is the multi-dimensional polynomial chaos of order n in random variables $(\xi_{i_1}, \dots, \xi_{i_n})$. A one-to-one mapping of polynomials $\{\Gamma_i\}$ to a set of polynomials with

ordered indices $\{\psi_i(\xi)\}$ can be introduced [6]. After substituting $\{\psi_i\}$ in Eq. 3.6 and truncating the series to finite number of terms N_ξ , the random field $a(x, \omega)$ can thus be approximated as

$$a(x, \omega) \approx \tilde{a}(x, \xi(\omega)) = a_0(x) + \sum_{i=1}^{N_\xi} a_i(x) \psi_i(\xi). \quad (3.7)$$

The polynomials $\{\psi_i(\xi)\}$ are orthogonal with respect to the inner product defined by expectation in the stochastic space,

$$\langle \psi_i(\xi), \psi_j(\xi) \rangle \equiv \int_{\Omega} \psi_i(\xi(\omega)) \psi_j(\xi(\omega)) dP(\omega) = \delta_{ij}. \quad (3.8)$$

As a second test-case, the diffusion coefficient $a(x, \omega)$, is modeled as a log-normal random field [5] where $a(x, \omega) = \exp(g(x, \omega))$ and $g(x, \omega)$ is a Gaussian random field with exponential covariance (3.4) and approximated with a truncated K-L expansion (3.5). In this case the random variables ξ_i are standard normal random variables and thus are independent. It also can be shown that the polynomials $\{\psi_i\}$ are tensor products of one-dimensional Hermite polynomials. For a given total polynomial order p , the total number of polynomials $\{\psi_i(\xi)\}$ is $N_\xi + 1 = \frac{(M+p)!}{M!p!}$.

4. Stochastic Galerkin method. In the stochastic Galerkin method, we seek the solution of the variational problem 2.4 in a tensor product space $X_h \otimes Y_p$, where, $X_h \subset H_0^1(D)$ is finite dimensional space of continuous polynomials corresponding to the spatial discretization of D and $Y_p \subset L_2(\Omega)$ is the space of random variables spanned by polynomial chaos [6] of order up to p . Then the finite dimensional approximation $u_{X_h Y_p}(x, \omega)$ of the exact solution $u(x, \omega)$ on the tensor product space $X_h \otimes Y_p$ is given as the solution to

$$b(u_{X_h Y_p}, v) = l(v) \quad \forall v \in X_h \otimes Y_p. \quad (4.1)$$

In Eq. 4.1 the random field $a(x, \omega)$ in the bilinear form $b(u_{X_h Y_p}, v)$ can be approximated using either a K-L expansion or a polynomial chaos expansion depending on the either linear or nonlinear dependence of the random field on the input random variables. The resulting set of coupled PDEs are then discretized using standard techniques such as the finite element or finite difference methods. In the former case, a trial function, $u_{X_h Y_p}$ can be written as

$$u_{X_h Y_p}(x, \xi) = \sum_{i,j} u_{ij} N_i(x) \psi_j(\xi), \quad (4.2)$$

where $\{N_i(x)\}$ and $\{\psi_j(\xi)\}$ are the finite element shape functions and polynomial chaos polynomials respectively. Substituting the trial function $u_{X_h Y_p}(x, \xi)$ and the test function $v(x, \xi) = N_k(x) \psi_l(\xi)$ in Eq. 4.1, the discretized equations can be written as

$$\sum_{j=0}^{N_\xi} \sum_{i=0}^{\hat{P}} c_{ijk} K_i u_j = f_k, \quad k = 0, \dots, N_\xi, \quad (4.3)$$

where $f_k = E\{f(x, \xi) \psi_k\}$, and $c_{ijk} = E\{\xi_i \psi_j \psi_k\}$ and $\hat{P} = M$ when a is approximated by a truncated K-L expansion, or $c_{ijk} = E\{\psi_i \psi_j \psi_k\}$ and $\hat{P} = \hat{N}_\xi$ when a is approximated by a polynomial chaos expansion. Here $\{K_i \in \mathbb{R}^{N_x \times N_x}\}_{i=0}^{\hat{P}}$ are the polynomial chaos coefficients of the stiffness matrix (section (3.4) of [12])

$$(K_i)_{lm} = \int_D a_i(x) \nabla N_l(x) \cdot \nabla N_m(x) dx, \quad i = 0, \dots, \hat{P}, \quad l, m = 1, \dots, N_x, \quad (4.4)$$

and $\{u_j \in \mathbb{R}^{N_x}\}_{j=0}^{N_\xi}$ are the polynomial chaos coefficients of the discrete finite-element solution vector

$$u_j = [u_{0j}, \dots, u_{N_x j}]^T, \quad j = 0, \dots, N_\xi. \quad (4.5)$$

$\{K_i\}_{i=0}^1$ and $\{K_i\}_{i=2}^{\hat{P}}$ are symmetric positive definite and symmetric indefinite matrices respectively. Equation 4.3 can be written in the form of a global stochastic stiffness matrix of size $((N_\xi + 1) \times N_x)$ by $((N_\xi + 1) \times N_x)$ as

$$\begin{bmatrix} K^{0,0} & K^{0,1} & \dots & K^{0,N_\xi} \\ K^{1,0} & K^{1,1} & \dots & K^{1,N_\xi} \\ \vdots & \vdots & \ddots & \vdots \\ K^{N_\xi,0} & K^{N_\xi,1} & \dots & K^{N_\xi,N_\xi} \end{bmatrix} \times \begin{Bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{N_\xi} \end{Bmatrix} = \begin{Bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{N_\xi} \end{Bmatrix} \quad (4.6)$$

where $K^{jk} = \sum_{i=0}^{\hat{P}} c_{ijk} K_i$. We will denote this system as $\bar{K}\bar{u} = \bar{f}$. In practice it is prohibitive to assemble and store the global stochastic stiffness matrix in this form, rather each block of the stochastic stiffness matrix can be computed from the $\{K_i\}$ when needed.

4.1. Solution methods for stochastic Galerkin systems. In this section, various solver techniques and preconditioning methods for solving the linear algebraic equations arising from stochastic Galerkin discretizations (4.3) are described. The solver methods discussed are: a Jacobi mean method, a Gauss-Seidel mean method, and Krylov-based iterative methods [14]. Also various stochastic preconditioners used to accelerate convergence of the Krylov methods are discussed, including mean-based [12], Gauss-Seidel mean, approximate Gauss-Seidel mean and approximate Jacobi mean preconditioners. In Jacobi and Gauss-Seidel methods, mean splitting is used rather than traditional diagonal block splitting as it allows use of the same mean matrix K_0 for all inner deterministic solves (and thus reuse of the preconditioner $P_0 \approx K_0$).

Jacobi mean algorithm. In this method, systems of equations of size equal to that of the deterministic system are solved iteratively by updating the right-hand-side to obtain the solution to the stochastic Galerkin system of equations (4.3):

$$c_{kk0} K_0 u_k^{new} = f_k - \sum_{j=0}^{N_\xi} \sum_{i=1}^{\hat{P}} c_{ijk} K_i u_j^{old}, \quad k = 0, \dots, N_\xi. \quad (4.7)$$

The above system of equations are solved for $k = 0, \dots, N_\xi$ using any solution technique appropriate for the mean matrix K_0 . Thus existing legacy software can be used with minimal modification to solve the stochastic Galerkin system. In this work, Krylov-based iterative methods with appropriate preconditioners will be used. One cycle of solves from $k = 0, \dots, N_\xi$ is considered one Jacobi outer iteration, and after each outer iteration, the right hand side in Eq. 4.7 is updated replacing $\{u_j^{old}\}$ with the new solution $\{u_j^{new}\}$. These outer iterations are continued until the required convergence tolerance is achieved. Note that for a given outer iteration, all of the right-hand-sides for $k = 0, \dots, N_\xi$ are available simultaneously, and thus their solution can be efficiently parallelized. Moreover block algorithms optimized for multiple right-hand-sides may be used to further increase performance. Finally this approach does not require a large amount of memory to compute the solution. The disadvantage of the method is it may not converge or converge very slowly.

Gauss-Seidel mean iterative method. The Gauss-Seidel method considered is similar to the the Jacobi method above, except the right-hand-side in Eq. 4.7 is updated after each deterministic solve with the newly computed u_k^{new} . Symbolically this is written

$$c_{kk0}K_0u_k^{new} = f_k - \sum_{j=0}^{k-1} \sum_{i=1}^{\hat{P}} c_{ijk}K_iu_j^{new} - \sum_{j=k}^{N_\xi} \sum_{i=1}^{\hat{P}} c_{ijk}K_iu_j^{old}, \quad k = 0, \dots, N_\xi. \quad (4.8)$$

As before, one cycle of solves from $k = 0, \dots, N_\xi$ is considered one outer iteration of the Gauss-Seidel method, and these outer iterations are repeated until the required convergence tolerance is achieved. Often this method converges in fewer iterations than the Jacobi method, at the expense of no longer having all of the right-hand-sides available simultaneously. This requires recomputing needed matrix-vector products $K_iu_j^{new}$ for each k , which adds additional computational cost, or storing them as they are computed, which adds additional memory requirements. In both the Jacobi and Gauss-Seidel methods, the left hand side matrix is the mean matrix for all inner deterministic problems and only the right hand side changes. In such cases recycled Krylov basis methods can be explored to increase performance.

Krylov based iterative methods with matrix-free operations. Krylov based iterative methods [14] such as the conjugate gradient (CG) method and the generalized minimal residual (GMRES) method can be used to solve the stochastic Galerkin system (4.3) in which matrix vector products $\bar{v} = \bar{K}\bar{u}$ are computed using “matrix free” operations:

$$v_k = \sum_{j=0}^{N_\xi} \sum_{i=0}^{\hat{P}} c_{ijk}K_iu_j, \quad k = 0, \dots, N_\xi. \quad (4.9)$$

If the matrix vector products are computed from Eq. 4.9, it is not required to assemble the full stochastic Galerkin stiffness matrix, drastically decreasing memory requirements. However if a large number of iterations of a Krylov method such as GMRES are required, allocation of the Krylov basis may still require a very large amount of memory. Thus good preconditioning strategies for the stochastic Galerkin system are required, several of which will be discussed below.

Mean-based preconditioner. The mean-based preconditioner [12] is given by $P = \text{diag}\{P_0, \dots, P_0\}$ where $P_0 \approx K_0$ is a preconditioner for the mean. The mean-based preconditioner is very efficient to compute and apply, since it only must be generated once from a matrix that is of the size of the deterministic system. However it doesn't incorporate any higher-order stochastic information, thus its performance degrades as the stochastic dimension, polynomial order, or random field variance increases [16].

Gauss-Seidel preconditioner. One or more outer iterations of the Gauss-Seidel mean algorithm can be used as a preconditioner to the Krylov based iterative methods. An advantage of this method is that the cost of applying the preconditioner can be controlled by adjusting the tolerance of the inner deterministic solves and number of outer iterations. Decreasing this tolerance and increasing the number of outer iterations will reduce the number of iterations in the Krylov method, but make the preconditioner more expensive to apply, and thus these must be balanced to minimize overall computational cost. Generally we have found the cost of the preconditioner to be dominated by solving the mean systems, and thus the best choice was a very loose inner solver tolerance (i.e., 0.1) and only one Gauss-Seidel iteration. However to prevent stagnation of the outer Krylov solver, a flexible variant of the Krylov method (e.g., FGMRES) was necessary.

Approximate Gauss-Seidel preconditioner. The process of increasing the inner solver tolerance can be taken to its extreme of replacing the inner mean solves by application of the mean preconditioner. As with the Gauss-Seidel preconditioner above, we found experimentally that this approach worked best with only one Gauss-Seidel iteration, and adding additional iterations did not improve the quality of the preconditioner. We also found the cost of the preconditioner was reduced dramatically if only the first-order terms in the expansion for the stiffness matrix are used in the preconditioner and using higher-order terms did not improve performance. We refer to this as the approximate Gauss-Seidel preconditioner.

Approximate Jacobi preconditioner. Similar to the approximate Gauss-Seidel preconditioner, Jacobi iterations can be used using a preconditioner in place of the mean stiffness matrix. In this case we used two outer Jacobi iterations, since the first iteration is equivalent to mean-based preconditioning (i.e., the additional terms on the right-hand-side of Eq. 4.7 are zero). Increasing the number of outer iterations did not improve the efficiency of the overall solver. We refer to this as the approximate Jacobi preconditioner.

5. Sparse grid collocation method. In the collocation method, the solution to the PDE is sampled at a pre-selected set of points called collocation points, $\Theta = (\xi^{(1)}, \dots, \xi^{(N)})$. The stochastic solution is constructed by interpolating at these collocation points,

$$u(x, \xi) \approx \sum_{k=0}^N u_k(x) L_k(\xi) \quad (5.1)$$

where $\{L_k(\xi)\}$ are Lagrange interpolatory polynomials defined by ξ_k ($L_k(\xi_l) = \delta_{kl}$) and u_k is the solution of following deterministic PDE,

$$-\nabla \cdot (a(x, \xi^{(k)}) \nabla u_k(x)) = f(x, \xi^{(k)}) \text{ in } D, \quad (5.2)$$

$$u_k(x) = 0 \text{ on } \partial D. \quad (5.3)$$

The collocation points can be chosen as tensor products of 1-D Gaussian quadrature points and the interpolating polynomials as tensor products of 1-D Lagrange interpolating functions. However the number of collocation points then grows exponentially with the number of random variables. An alternative method is to use Smolyak sparse grid quadrature ([15, 10]) where collocation points that do not increase asymptotic accuracy are removed from the tensor product grid. This results in many fewer collocation points but still more than the number of stochastic degrees of freedom in the stochastic Galerkin method. This method is fully non-intrusive and is easy to implement with existing legacy software (once the sparse grid is generated) [1].

6. Numerical illustration. To compare the performance of the different solvers and preconditioners discussed above, the 2-D stochastic diffusion problem presented in section 2 is solved using both the stochastic Galerkin and stochastic collocation methods from sections 4 and 5. For both solution approaches, the random field is treated as both a uniform random field discretized using a truncated K-L expansion (section 3.1) and a log-normal random field discretized using a truncated polynomial chaos expansion (section 3.2). In the first case, the orthogonal polynomials used in the stochastic Galerkin method are tensor products of 1-D Legendre polynomials whereas the collocation points used in the sparse grid stochastic collocation method are built from Gauss-Legendre points, and in the second case tensor products of Hermite polynomials and Gauss-Hermite quadrature points are used. The Dakota package [1] is used to generate the resulting sparse stochastic collocation grids. The spatial dimensions are discretized using a five-point finite-difference stencil on a 32×32 grid

in the domain $D = [0, 1] \times [0, 1]$, resulting in a total number of spatial degrees of freedom $N_x = 1024$. For simplicity a constant unit force $f(x, \omega) = 1$ is used as the right-hand-side in Eq. 2.2. The corresponding stochastic Galerkin linear system is constructed using the Stokhos and Epetra packages in Trilinos. For the Jacobi solver, Gauss-Seidel solver, Gauss-Seidel preconditioner, and stochastic collocation method, the linear systems are solved via multi-grid preconditioned GMRES provided by the AztecOO and ML Trilinos packages. For a consistent comparison of all of the preconditioning methods, FGMRES provided by the Belos Trilinos package is used as the outer Krylov solver, with ML providing the preconditioner in the mean-based and approximate Gauss-Seidel and Jacobi preconditioners. GMRES Krylov methods are employed instead of CG for generality and the numerical implementation of the boundary conditions resulted in unsymmetric matrices K_i .

The solution time for these solvers and preconditioning techniques as a function of the standard deviation of the input random field, stochastic dimension, and polynomial order are tabulated in Tables 6.1-6.6. In the tables, MB, AGS, AJ and GS are the mean-based, approximate Gauss-Seidel, approximate Jacobi, and Gauss-Seidel preconditioners respectively for the FGMRES Krylov method. GS_1 and GS_2 are Gauss-Seidel solvers where GS_1 refers to the Gauss-Seidel algorithm in which the matrix vector products $K_i u_j$ are saved in an array for reuse in later iterations of the “ k ” loop, whereas GS_2 refers to the variant where these products are recomputed when needed. GS_1 is generally more efficient, but for higher stochastic dimension or polynomial order, it requires a large amount of memory. “Jacobi” refers to the Jacobi mean solver, and “collocation” is the solution time using the Smolyak sparse grid collocation method. The solution tolerance for all of the stochastic Galerkin solvers, as well as the solver tolerance for the collocation method is $1e - 12$. For the Gauss-Seidel and Jacobi solvers, the inner solver tolerance is $3e - 13$, and for the Gauss-Seidel preconditioner, the inner solver tolerance is 0.1.

In the tables, DNC means “did not converge”, “Div.” means diverged, and “memory” means system memory was exceeded. For the uniform random field with small variance ($\sigma = 0.1$), it can be observed from Tables 6.1 and 6.2 that the more intrusive Krylov-based stochastic Galerkin solvers are more efficient than the less intrusive Gauss-Seidel and Jacobi solvers, which are in turn generally more efficient than the non-intrusive stochastic collocation method. Moreover the approximate Gauss-Seidel and Jacobi preconditioners are a significant improvement over the traditional mean-based approach. However as the variance of the random field is increased, we see from Table 6.3 the Gauss-Seidel and Jacobi solvers suffer considerably, whereas the the Krylov-based approaches (excluding the Gauss-Seidel preconditioner) still perform quite well. This is not unexpected, as the operator becomes more indefinite as the variance increases. However for the log-normal random field, we see from Tables 6.4 and 6.5 that the Krylov-based stochastic Galerkin approach is only more efficient than the collocation approach for larger stochastic dimension or polynomial order when using the approximate Gauss-Seidel or approximate Jacobi preconditioners. It is also interesting to see that Gauss-Seidel solver, GS_1 is faster than GMRES with mean-based preconditioning in this case. For higher variance of the random field, we see from Table 6.6 the Krylov iterative method with the approximate Jacobi preconditioner failed to converge and the Jacobi solver diverged. This problem can be rectified by using the true diagonal matrix $K^{k,k} = \sum_{i=0}^M c_{ikk} K_i$ from global stochastic stiffness matrix as the left-hand-side in the Jacobi solver and preconditioner instead of the mean matrix K_0 .

Figures 6.1(a) and 6.1(b) show a plot of relative residual error vs iteration count for the stochastic Galerkin system with stochastic dimension 5 and polynomial order 5. It can be observed that the Gauss-Seidel solver takes the least number of iterations where as the Jacobi solver takes highest number of iterations for a given tolerance. However in terms of

TABLE 6.1

Solution time (sec) vs stochastic dimension for uniform random field, PC order = 5 and $\sigma=0.1$

Stoch. dim	Preconditioners for GMRES				GS and Jacobi Solvers			SprseGrid collocation
	MB	AGS	AJ	GS	GS_1	GS_2	Jacobi	
2	0.20	0.12	0.18	0.25	1.23	1.21	2.22	0.52
3	0.70	0.39	0.54	0.77	3.84	3.87	7.46	3.18
4	1.78	1.01	1.38	2.02	9.56	9.73	18.60	10.24
5	4.34	2.31	3.05	4.59	20.40	20.90	41.10	26.96
6	10.24	5.41	7.10	9.81	46.10	46.30	87.50	64.09
7	19.50	10.24	12.96	19.64	81.20	80.40	160.00	134.45

solution time, the matrix free Krylov solver with the approximate Gauss-Seidel or Jacobi preconditioner is the most efficient.

TABLE 6.2

Solution time (sec) vs order of polynomial chaos when diffusion coefficient is uniform random field, Stoch. dim=3, $\sigma = 0.1$

PC order	Preconditioners for GMRES				GS and Jacobi Solvers			SprseGrid collocation
	MB	AGS	AJ	GS	GS_1	GS_2	Jacobi	
2	0.10	0.05	0.07	0.11	0.47	0.45	0.83	0.11
3	0.19	0.12	0.16	0.24	1.10	1.08	2.09	0.42
4	0.39	0.23	0.32	0.47	2.20	2.18	4.15	1.27
5	0.69	0.39	0.54	0.77	3.80	3.88	7.47	3.21
6	1.08	0.90	0.91	1.54	6.22	6.61	12.3	7.06
7	1.63	1.05	1.27	1.87	9.16	9.33	17.90	14.20
8	2.51	2.65	2.14	3.53	13.10	14.20	26.10	26.08
9	3.81	3.43	2.94	5.16	18.20	20.40	36.40	46.72
10	5.22	4.44	3.92	6.73	24.40	27.30	47.30	78.87

TABLE 6.3

Solution time (sec) vs standard deviation (σ) when diffusion coefficient is uniform random field, Stoch dim = 3 and PC order = 5

σ	Preconditioners for GMRES				GS and Jacobi Solvers			SprseGrid collocation
	MB	AGS	AJ	GS	GS_1	GS_2	Jacobi	
0.10	0.66	0.40	0.59	0.77	3.80	3.88	7.37	3.18
0.11	0.75	0.46	0.64	0.87	4.73	4.79	9.15	3.32
0.12	0.88	0.50	0.72	0.98	5.83	5.94	11.50	3.46
0.13	1.09	0.58	0.80	1.17	7.67	7.83	14.90	3.63
0.14	1.38	0.69	1.00	1.48	10.80	11.00	21.20	3.85
0.15	1.91	0.97	1.29	1.89	15.60	18.00	34.70	4.16

7. Conclusions. In this work, various preconditioners for Krylov-based methods and solver methods based on Gauss-Seidel and Jacobi method are introduced. Results are compared with GMRES with mean-based preconditioning and collocation. In the case of linear dependence on the random variables, we generally find the more intrusive Krylov-based approaches to be more efficient than the non-intrusive collocation approach, with the less

TABLE 6.4

Solution time (sec) vs stochastic dimension when diffusion coefficient is log-normal random field, PC order = 5 and $\sigma=0.1$

Stoch. dim	Preconditioners for GMRES				GS and Jacobi Solvers			SprseGrid collocation
	MB	AGS	AJ	GS	GS_1	GS_2	Jacobi	
2	0.26	0.23	0.23	0.41	0.59	0.78	1.61	0.57
3	1.40	1.15	1.06	2.13	2.16	3.57	5.26	3.25
4	7.10	5.45	4.79	10.17	7.14	14.30	16.80	10.91
5	28.08	20.60	19.46	38.68	22.30	47.90	49.70	27.60
6	93.20	66.05	59.84	132.30	memory	139.00	137.00	63.98

TABLE 6.5

Solution time (sec) vs order of polynomial chaos when diffusion coefficient is lognormal random field, Stoch. dim=3, $\sigma = 0.1$

PC order	Preconditioners for GMRES				GS and Jacobi Solvers			SprseGrid collocation
	MB	AGS	AJ	GS	GS_1	GS_2	Jacobi	
2	0.06	0.05	0.07	0.11	0.21	0.21	0.45	0.11
3	0.16	0.13	0.15	0.25	0.51	0.50	1.11	0.42
4	0.47	0.40	0.37	0.77	1.04	1.23	2.47	1.29
5	1.39	1.15	1.06	2.12	2.15	3.53	5.23	3.28
6	4.01	3.05	2.74	5.50	4.39	10.60	11.50	7.39
7	8.69	6.23	5.50	11.18	8.03	22.30	23.50	15.02
8	23.79	15.40	13.88	30.47	17.30	63.50	54.10	28.40
9	54.36	35.40	34.80	71.38	36.40	161.00	126.00	50.57
10	106.40	66.95	66.80	125.96	64.70	295.00	253.00	87.27

TABLE 6.6

Solution time (sec) vs standard deviation (σ) when diffusion coefficient is log-normal random field, Stoch dim = 3 and PC order = 5

σ	Preconditioners for GMRES				GS and Jacobi Solvers			SprseGrid collocation
	MB	AGS	AJ	GS	GS_1	GS_2	Jacobi	
0.10	1.51	1.23	1.14	2.08	2.11	3.56	5.21	3.25
0.15	1.86	1.42	1.25	2.49	2.86	4.87	9.15	3.64
0.20	2.20	1.72	2.06	3.06	3.63	6.17	46.80	4.11
0.25	2.66	2.03	DNC	3.58	4.87	8.40	Div.	4.59
0.30	3.11	2.35	DNC	4.35	6.90	11.90	Div.	5.13
0.35	3.65	2.87	DNC	5.17	11.20	19.30	Div.	5.72
0.40	4.44	3.39	DNC	6.16	24.30	41.90	Div.	6.53

intrusive Gauss-Seidel/Jacobi approaches in between. This demonstrates a trade-off in performance versus intrusiveness when existing legacy simulation codes must be used. In the case of nonlinear dependence on the random variables however, extrapolating beyond these tables suggests the non-intrusive collocation approach is in fact the most efficient. This generally suggests that for linear problems (in the random variables and the solution), intrusive stochastic Galerkin approaches would be preferred, but for nonlinear problems (in either the random variables or the solution), non-intrusive approaches would be preferred. Regardless, the use of approximate Gauss-Seidel or Jacobi preconditioners is a significant improvement over traditional mean-based preconditioning. In the future, we want to compare the Kro-

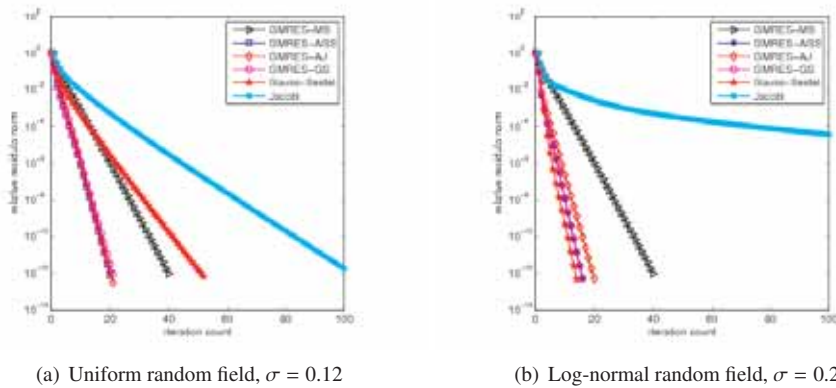


FIG. 6.1. Relative error norm vs iteration count for various solvers and preconditioners, where $M = 5$ and $p = 5$

necker product preconditioner proposed in [16] with the above methods. We would also like to investigate block and recycled Krylov methods to improve the efficiency of the Jacobi and Gauss-Seidel solvers.

REFERENCES

- [1] B. M. ADAMS, K. R. DALBEY, M. S. ELDER, D. M. GAY, L. P. SWILER, W. J. BOHNHOFF, J. P. EDDY, K. HASKELL, AND P. D. HOUGH, *DAKOTA, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis*, Sandia National Laboratories, tech. rep.sand2010-2183 ed., May 2010.
- [2] I. BABUŠKA, F. NOBILE, AND R. TEMPONE, *A stochastic collocation method for elliptic partial differential equations with random input data*, SIAM J. Numer. Anal., 45 (2007), pp. 1005–1034.
- [3] I. BABUŠKA, R. TEMPONE, AND G. E. ZOURARIS, *Galerkin finite element approximations of stochastic elliptic partial differential equations*, SIAM J. Numer. Math., 42 (2004), pp. 800–825.
- [4] M. EIERMANN, O. G. ERNST, AND E. ULLMANN, *Computational aspects of the stochastic finite element method*, Computing and Visualization in Science, 10 (2007), pp. 3–15.
- [5] R. GHANEM, *Ingredients for a general purpose stochastic finite elements implementation*, Computer Methods in Applied Mechanics and Engineering, 168 (1999), pp. 19–34.
- [6] R. GHANEM AND P. SPANOS, *Stochastic Finite Elements: A Spectral Approach*, Springer-Verlag, 1991.
- [7] R. G. GHANEM AND A. DOOSTAN, *On the construction and analysis of stochastic models: characterization and propagation of the errors associated with limited data*, Journal of Computational Physics, 213 (2006), pp. 63–81.
- [8] R. G. GHANEM AND R. M. KRUGER, *Numerical solution of spectral stochastic finite element systems*, Comput. Methods Appl. Mech. Engrg., 129 (1996), pp. 289–303.
- [9] C. W. MILLER, R. S. TUMINARO, E. T. PHIPPS, AND H. C. ELMAN, *Assessment of collocation and galerkin approaches to stochastic partial differential equations*, in CSRI Summer Proceedings 2009.
- [10] F. NOBILE, R. TEMPONE, AND C. G. WEBSTER, *A sparse grid stochastic collocation method for partial differential equations with random input data*, SIAM Journal of Numerical Analysis, 46 (2008), pp. 2309–2345.
- [11] M. F. PELLISSETTI AND R. G. GHANEM, *Iterative solution of systems of linear equations arising in the context of stochastic finite elements*, Advances in Engineering Software, 31 (2000), pp. 607–616.
- [12] C. POWELL AND H. ELMAN, *Block-diagonal preconditioning for spectral stochastic finite element systems*, IMA Journal of Numerical Analysis, 29 (2009), pp. 350–375.
- [13] E. ROSSEEL AND S. VANDEWALLE, *Iterative solvers for the stochastic finite element method*, SIAM J. Sci. Comput., 32 (2010), pp. 372–397.
- [14] Y. SAAD, *Iterative Methods for Sparse Linear Systems*, SIAM, 1996.
- [15] S. SMOLYAK, *Quadrature and interpolation formulas for tensor products of certain classes of functions*, Dokl. Akad. Nauk SSSR, 148 (1963), pp. 1042–1043.
- [16] E. ULLMANN, *A kronecker product preconditioner for stochastic galerkin finite element discretizations*, SIAM J. Sci. Comput., 32 (2010), pp. 923–946.
- [17] N. WEINER, *The homogeneous chaos*, American Journal of Mathematics, 60 (1963), pp. 897–936.

- [18] D. XIU AND J. HESTHAVEN, *High-order collocation methods for differential equations with random inputs*, SIAM Journal of Scientific Computing, 60 (2005), pp. 1118–1139.

UNCERTAINTY QUANTIFICATION OF THE SEMICONDUCTOR DRIFT-DIFFUSION EQUATIONS

CHRISTOPHER W. MILLER*, RAYMOND S. TUMINARO[†], ERIC T. PHIPPS[‡], AND HOWARD C. ELMAN[§]

Abstract. The movement of charge carriers in a semiconductor device is modeled by a set of coupled non-linear partial differential equations known as the drift-diffusion equations. The physical parameters involved in defining these equations are subject to large amounts of uncertainty. The aim of this paper is to examine the applicability of uncertainty quantification techniques to this problem. We express the uncertainty regarding these parameters by modeling them as random variables and apply anisotropic sparse grid collocation to the resulting set of stochastic partial differential equations. We identify the most sensitive parameters using local sensitivity analysis, and use this information to formulate a reduced problem. We apply Sobol' sensitivity analysis to the solution of the reduced model and analyze the probability distribution of the model outputs at various time steps. This preliminary work reveals new approaches for quantifying uncertainty in semiconductor devices.

1. Introduction. Radiation interacts with semiconductor devices by knocking atoms from the device's silicon lattice. These defect species, which can consist of silicon or the P-type or N-type dopants, can carry charge and propagate through the device. Examining the behavior of semiconductor devices in radioactive environments is complicated by the cost and lack of availability of experimental facilities. To alleviate this, Sandia National Laboratories has invested in the use of computational modeling to examine the performance of semiconductor devices in radioactive environments. In particular, the finite element code *Charon* was developed to discretize and solve the semiconductor drift-diffusion equations that model the movement of charge carriers inside semiconductor devices [5] [15].

A difficulty with this model is that the parameters that describe the interactions of the defect species are often only known to a limited accuracy. Confidence in the model solutions is then limited by lack of confidence in the accuracy of the parameter values. The lack of knowledge represents an epistemic uncertainty which can be quantified by modeling the parameters as random variables. As a consequence of the Doob-Dynkin lemma, the solution of the drift-diffusion equations can be represented as a random process of the uncertain parameters [4]. Several methods have been developed to approximate the statistics associated with such a random solution process, including the Monte-Carlo method [9], and more recently, the stochastic collocation methods [2],[10],[11],[16]. In this paper we apply the anisotropic sparse grid collocation method because of its favorable convergence properties and minimal dependence on the size of the parameter space [11]. The solution process arising from the collocation method can be post processed to compute statistical quantities associated with the solution process.

This work is viewed as a continuation of [12], which performed a transient sensitivity analysis of the solution of the deterministic drift-diffusion equations with respect to the defect reaction parameters. Here we seek to identify additional methods for quantifying the uncertainty inherent in this problem. The structure of this paper is as follows. Section 2 describes the deterministic formulation of the semiconductor drift diffusion equations and describes our extension of this problem from a deterministic setting to a non-deterministic one. Section 3 describes the sparse grid stochastic collocation method used to perform the uncertainty quantification calculations. Section 4 describes how the solution derived using

*The University of Maryland at College Park: Department of Applied Mathematics and Scientific Computation, cmiller@math.umd.edu

[†]Sandia National Laboratories, rstumin@sandia.gov

[‡]Sandia National Laboratories, ethipp@sandia.gov

[§]The University of Maryland at College Park: Department of Computer Science and Institute for Advance Computer Studies, elman@cs.umd.edu

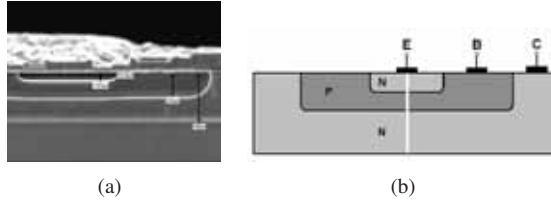


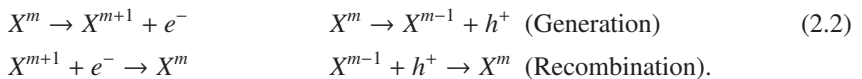
FIG. 2.1. *Scanning electron microscope of an NPN BJT (a) and diagram of the emitter, base, and collector regions (b)[12].*

the collocation method can be examined to explore interactions among parameters. Section 5 describes the application of these methods to a semiconductor device under the influence of a radiation pulse. Finally, in Section 6 we draw some conclusions and propose additional applications.

2. Non-deterministic Problem Formulation. The device considered in this paper is an NPN bipolar junction transistor (BJT) subjected to a radiation pulse. The device is pictured in Figure 2.1. In a BJT the silicon lattice has been modified by the introduction of dopants to produce an excess of free electrons in the N-regions (N-doping), and to produce an excess of holes (positive charge carriers) in the P-region (P-doping). The P-dopant in the device considered here is boron, while the N-dopant is phosphorus. Radiation interacts with the device by knocking an atom free from the lattice. This creates a free interstitial atom and a vacancy referred to as a Frenkel pair. Both the free atom and vacancy can carry charge, move through the device, and interact with other defect species. The diffusion, transport, and generation of charge carriers are governed by the following set of coupled partial differential equations

$$\begin{aligned}
 -\nabla \cdot (\lambda^2 \nabla \psi) &= \left(p - n + C \sum_{i=1}^N Z_i Y_i \right) \\
 \nabla \cdot (-\mu_n n \nabla \psi + D_n \nabla n) &= \frac{\partial n}{\partial t} + R_n \\
 \nabla \cdot (\mu_p p \nabla \psi + D_p \nabla p) &= \frac{\partial p}{\partial t} + R_p \\
 \nabla \cdot (\mu_{Y_i} Y_i \nabla \psi + D_{Y_i} \nabla Y_i) &= \frac{\partial Y_i}{\partial t} + R_{Y_i}, \quad i = 1, \dots, N,
 \end{aligned} \tag{2.1}$$

where ψ is the scalar electric potential, n , p , and Y_i are the electron, hole, and i^{th} defect species densities respectively. For $x \in \{n, p, Y_i\}$, D_x and μ_x are mobility and diffusivity coefficients for species x , Z_i is the integer charge of the i^{th} defect species, λ is the Debye length of the device and C is the doping profile. The generation and recombination of species x is given by the right hand side term R_x [15]. The reactions we are concerned with are the so-called carrier-defect reactions: reactions between a defect and a hole or electron. These reactions have the form



The forcing term associated with these reactions is modeled by

$$R_{X^{m+1}} = \sigma A X^m \exp\left(\frac{\Delta E}{kT}\right). \tag{2.3}$$

Here X denotes the concentration of a certain defect species with superscripts denoting the integer charge of the defect. A is a constant, σ is the reaction cross section, ΔE is the activation energy, k is Boltzmann's constant and T is the lattice temperature. The parameters that we investigate are the reaction cross-sections and the activation energies. A subscript on σ and ΔE is omitted to improve readability; however each reaction has a different value for each of these parameters. The activation energy for a recombination reaction is known to be equal to zero [12]. For our problem there are 84 carrier-defect reactions involving 35 defect species and a total of 127 reaction parameters. Table 2.1 shows a small sample of the carrier-defect reactions along with an estimate of one of the associated parameters.

#	Reaction	Parameter	Approximate Value
13	$e^- + V^- \rightarrow V^{--}$	σ	3.0×10^{-16}
14	$V^{--} \rightarrow e^- + V^-$	ΔE	0.09
40	$e^- + BV^+ \rightarrow BV^0$	σ	3.0×10^{-14}

TABLE 2.1
A sample of the 127 carrier-defect reactions [12]

Discretization and numerical solution to the partial differential equations in (2.1) is accomplished using Sandia's *Charon* software. Chemical kinetics computations are accomplished using *CHEMKIN* [6]. *Charon* uses a Galerkin finite element discretization consisting of two-dimensional piecewise bilinear finite element functions defined on a mesh of quadrilaterals with streamline upwind Petrov-Galerkin stabilization. Further details of the discretization procedure used can be found in [5]. In this study we perform the calculations on the pseudo one-dimensional domain shown as the vertical white strip under the emitter in Figure 2.1. Two-dimensional effects do not arise for the device operating under the conditions considered in this paper.

As stated previously, the reaction parameters σ and ΔE appearing in (2.2) are subject to a large degree of uncertainty. Our aim here is to develop quantitative insight into the effects of these reaction parameters, denoted generically here as $\{\xi_i\}$. To accomplish this, we model each of the 127 reaction parameters ξ_i as a uniform random variable centered at μ_i ; the estimated value from previous deterministic studies [12]. The probability distribution of each random variable is then given by

$$\rho_i(\xi_i) = \frac{1}{1.2\mu_i} \mathbb{1}_{[\mu_i - .6\mu_i, \mu_i + .6\mu_i]}. \quad (2.4)$$

Although these intervals are large, they still may underestimate the uncertainty associated with the parameters. We assume that the uncertainties with respect to each parameter are independent and so we can define a joint probability distribution on the parameter space by

$$\rho(\xi) = \prod_{i=1}^{127} \rho_i(\xi_i) = \prod_{i=1}^{127} \frac{1}{1.2\mu_i} \mathbb{1}_{[\mu_i - .6\mu_i, \mu_i + .6\mu_i]}. \quad (2.5)$$

The problem is now to find random processes

$$\psi(\mathbf{x}, t, \xi), \quad n(\mathbf{x}, t, \xi), \quad h(\mathbf{x}, t, \xi), \quad Y_k(\mathbf{x}, t, \xi) \quad (2.6)$$

that satisfy (2.1) almost surely. It should be noted that additional uncertainties are associated with the doping profile and the diffusivity and mobility coefficients. These are not considered in this paper

The assumption of independence deserves some scrutiny since the parameters associated with all of the reactions involving a given defect species are almost certainly correlated. However it is shown in [2] that the replacement of the true joint density function with the product of the marginal density functions only affects the convergence of collocation methods up to a constant. This constant may be large however and further work is required to better approximate the joint PDF appearing in (2.5).

3. The Stochastic Collocation Method. A methodology for computing the random processes in (2.6) is the stochastic collocation method. The sparse grid collocation method was first described in [16] and error analysis was performed in [2] and [10]. These methods are all suited to problems whose dependence on the random parameters is isotropic. We choose to apply an anisotropic version of the sparse grid collocation method developed by [11]. Other approaches for addressing anisotropic problems can be found in [7] and [8]. Here we only present the derivation of the isotropic method described in [16].

In order to derive the stochastic collocation method, one begins by considering interpolation operators defined for one-dimensional functions defined on a finite interval. Without loss of generality, we can assume that the interval is $[-1, 1]$. Let $f : [-1, 1] \rightarrow \mathbb{R}$ and define the interpolation operator

$$\mathcal{U}^m f(\xi) = \sum_{k=1}^m f(\xi^{(k)}) l_k(\xi), \quad (3.1)$$

where $\{\xi^{(k)}\} = \theta_m$ is a set of m distinct points and where l_k is the Lagrange interpolating polynomial of degree $m - 1$ defined by

$$l_k(\xi^{(j)}) = \delta_{kj}. \quad (3.2)$$

Evaluation of the interpolant requires the evaluation of the function f at the points contained in θ_m . By construction we have that $f(\xi^{(k)}) = \mathcal{U}^m f(\xi^{(k)})$ for all $\xi^{(k)}$ in θ_m .

Now we consider interpolation in multiple dimensions. Let $f : [-1, 1]^M \rightarrow \mathbb{R}$. In order to generalize the one-dimensional interpolation operators to multiple dimensions, an obvious approach would be to take tensor products of one-dimensional interpolation operators $\mathcal{U}^{m_{i_k}}$ along each coordinate axis. Define the tensor product interpolant by

$$\mathcal{A}_{\mathbf{i}}^{Tensor} f(\xi) = \mathcal{U}^{m_{i_1}} \otimes \mathcal{U}^{m_{i_2}} \otimes \cdots \otimes \mathcal{U}^{m_{i_M}} f(\xi). \quad (3.3)$$

The multi-index $\mathbf{i} \in \mathbb{N}^{+M}$ describes how many interpolation points are used along each axis. The evaluation of $\mathcal{A}_{\mathbf{i}}^{Tensor}$ requires the evaluation of the function f on the grid

$$\Theta_{\mathbf{i}} = \times_{k=1}^M \theta_{m_{i_k}}, \quad (3.4)$$

with the cardinality of this grid given by

$$|\Theta_{\mathbf{i}}| = \prod_{k=1}^M m_{i_k}. \quad (3.5)$$

The relation (3.5) is referred to as the *curse of dimensionality* [16] since the cardinality of the grid grows exponentially in the dimension M . Thus for problems involving a moderate or large number of parameters, use of the tensor product formula (3.3) is computationally infeasible.

Smolyak sparse grids provide a method of approximating multi-dimensional functions

that avoids the curse of dimensionality. Sparse grid interpolation was introduced in [13]. The sparse grid interpolant is formed by taking a selective sum of tensor product rules appearing in (3.3). Define the index set

$$Y_{q,M} = \left\{ \mathbf{i} \in \mathbb{N}^M, \mathbf{i} \geq 1 : q - M + 1 \leq \sum_{k=1}^M (i_k - 1) \leq q \right\}. \quad (3.6)$$

Then the sparse grid interpolation operator is given by

$$\mathcal{A}_{q,M}(f) = \sum_{\mathbf{i} \in Y_{q,M}} (-1)^{q+M-|\mathbf{i}|} \binom{M-1}{q+M-|\mathbf{i}|} \cdot (\mathcal{U}^{m_{i_1}} \otimes \cdots \otimes \mathcal{U}^{m_{i_M}}). \quad (3.7)$$

Evaluation of this interpolation operator requires the evaluation of the function f on the sparse grid

$$\mathcal{H}_{q,M} = \bigcup_{\mathbf{i} \in Y_{q,M}} (\theta_{m_{i_1}} \times \cdots \times \theta_{m_{i_M}}). \quad (3.8)$$

In order to fully define the sparse grid interpolation operator it is necessary to specify the points used in constructing the one-dimensional interpolation operators. It is advantageous if the grids have the property that $\mathcal{H}_{q,M} \subset \mathcal{H}_{q+1,M}$. One way to accomplish this is to construct the one-dimensional operators using the Clenshaw-Curtis abscissas [16]. Let

$$\xi_j^i = -\cos\left(\frac{\pi(j-1)}{m_i-1}\right), \quad j = 1, \dots, m_i \quad (3.9)$$

$$m_i = \begin{cases} 1 & \text{if } i = 1, \\ 2^{i-1} + 1 & \text{if } i > 1 \end{cases} \quad (3.10)$$

With this choice of points we obtain $\theta_i \subset \theta_{i+1}$ and hence $\mathcal{H}_{q,M} \subset \mathcal{H}_{q+1,M}$. In this case (3.7) and (3.8) simplify to

$$\mathcal{A}_{q,M}f = \sum_{\mathbf{i} \in X_{q,M}} (-1)^{q+M-|\mathbf{i}|} \binom{M-1}{q+M-|\mathbf{i}|} \cdot (\mathcal{U}^{m_{i_1}} \otimes \cdots \otimes \mathcal{U}^{m_{i_M}}) \quad (3.11)$$

and

$$\mathcal{H}_{q,M} = \bigcup_{\mathbf{i} \in X_{q,M}} (\theta_{m_{i_1}} \times \cdots \times \theta_{m_{i_M}}) \quad (3.12)$$

$$\text{where } X_{q,M} = \left\{ \mathbf{i} \in \mathbb{N}^M : \sum_{k=1}^M (i_k - 1) = q \right\}.$$

It is known that if f is a M -variate polynomial of total degree $q - M + 1$ then $\mathcal{A}_{q,M}f = f$ [3]. Thus one can expect that if f is sufficiently regular than the approximation $\mathcal{A}_{q,M}f$ converges quickly in the sparse grid level q . This statement is made precise in [10].

The solution obtained through collocation can be post-processed to compute various other quantities of interest [16]. Moments of f can be approximated by

$$\mathbb{E}(f^m) \approx \mathbb{E}(\mathcal{A}_{q,M}f^m) \quad (3.13)$$

where the quantity on the right of (3.13) can be computed exactly and efficiently using Clenshaw-Curtis quadrature. One may also want to compute probability distributions associated with the solution. These can be approximated by sampling the collocation solution

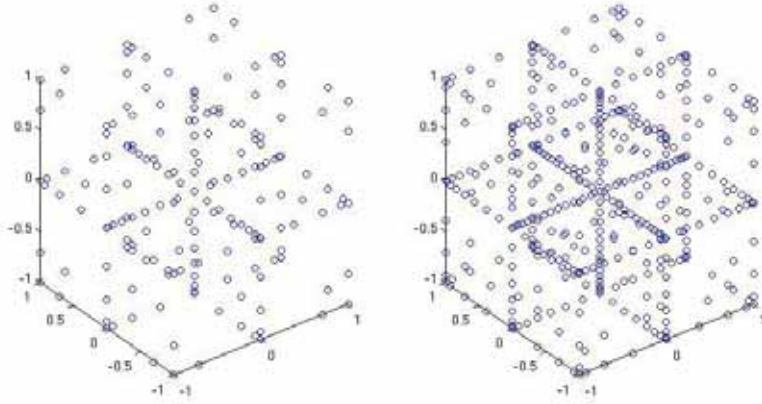


FIG. 3.1. Clenshaw-Curtis based sparse grids $\mathcal{H}(2,3)$ and $\mathcal{H}(3,3)$

at a random set of points and then measuring the relative frequency of the desired event. Assume that we sample $\mathcal{A}_{q,M}(f)$ N times. Let $A = \{\xi^{(k)}\}_{k=1}^N$ be the set of sample points and let $A_{\mathcal{A}_{q,M}f \leq c}$ be the subset of points in A such that $\mathcal{A}_{q,M}f(\xi^{(k)}) \leq c$, then

$$P[f \leq c] \approx \frac{|A_{\mathcal{A}_{q,M}f \leq c}|}{|A|}. \quad (3.14)$$

The advantage of this approach to computing the CDF instead of direct sampling of the function f is that evaluation of $\mathcal{A}_{q,M}(f)$ can often require much less computational effort than directly evaluating f . Our routines for performing the sparse grid collocation method and post processing the collocation solution are provided by Sandia's *Dakota* software [1].

4. Sobol' Sensitivity Analysis. Given a function $f : [-1, 1]^M \rightarrow \mathbb{R}$ of M parameters one may be interested in how perturbations of each parameter contribute to changes in the function value. In many cases however the function value is not sensitive with respect to perturbations of a single parameter but rather is sensitive with respect to simultaneous changes in multiple parameters. *Sobol' sensitivity analysis* is one method for describing the sensitivity of the function with respect to coupled subsets of parameters. The method proceeds by performing a standard ANOVA decomposition of the function f and then computing the *Sobol' indices* as a ratio of the partial variance to the total variance [14].

The ANOVA decomposition of f is as follows. Assuming that f is square integrable, decompose f into the sum

$$\begin{aligned} f(\xi) &= f_0 + \sum_{s=1}^M \sum_{l_1 < \dots < l_s} f_{l_1, \dots, l_s}(\xi_{l_1}, \xi_{l_2}, \dots, \xi_{l_s}) \\ f_0 &= \int_{[-1,1]^M} f(\xi) d\xi \\ f_{l_1}(\xi_{l_1}) &= \int_{[-1,1]^{M-1}} f(\xi) \prod_{k \neq l_1} d\xi_k - f_0 \\ f_{l_1 l_2}(\xi_{l_1}, \xi_{l_2}) &= \int_{[0,1]^{M-2}} f(\xi) \prod_{k \neq l_1, l_2} d\xi_k - f_0 - f_{l_1}(\xi_{l_1}) - f_{l_2}(\xi_{l_2}). \end{aligned} \quad (4.1)$$

Then define the partial and total variances as

$$D_{l_1, \dots, l_s} = \int_{[-1,1]^s} f_{l_1, \dots, l_s}^2 d\xi_{l_1} \dots d\xi_{l_s} \text{ and } D = \int_{[-1,1]^M} f^2(\xi) d\xi - f_0^2. \quad (4.2)$$

Finally define the Sobol' index and total Sobol' index as

$$S_{l_1, \dots, l_s} = \frac{D_{l_1, \dots, l_s}}{D} \text{ and } TS_i = \sum_{s=1}^{M-1} \sum_{l_1 < \dots < l_i < \dots < l_s} S_{l_1, \dots, l_i, \dots, l_s}. \quad (4.3)$$

The Sobol' index measures the dependence of the function on each subset of the M parameters while the total Sobol' index TS_i measures the dependence of the function f on the i^{th} parameter. There is a total of 2^M terms appearing in (4.1), $2^M - 1$ Sobol' indices, and M total Sobol' indices. The Sobol' indices can be used to measure the strength of coupling effects between specific subsets of parameters on the function f . The Sobol' indices S_i for $1 < i < M$ are referred to as the main effect of parameter i . The total Sobol' index can be used as a global sensitivity measure of f with respect to a given parameter. We define the ratio

$$r_i = \frac{TS_i - S_i}{TS_i} \quad (4.4)$$

as the relative error between the i^{th} total Sobol' index and the main effect of parameter i for $1 < i < M$. Obviously r_i satisfies $0 \leq r_i \leq 1$. If r_i is close to 1 then most of the sensitivity with respect to parameter i is tied up in coupling effects. If r_i is close to 0 then the i^{th} parameter is only weakly coupled to the rest of the parameters. This information can be used to examine the strength of parameter coupling effects on the model solution.

5. Application to the semiconductor drift diffusion equations. In this section we apply the above techniques to the analysis of a radiation damaged BJT. A radiation pulse is simulated by adding a transient source of electrons and holes [12]. The shape of this pulse is shown in Figure 5.1. The radiation pulse begins at time $t = 1 \times 10^{-5}$ and ends at time $t = 2 \times 10^{-4}$.

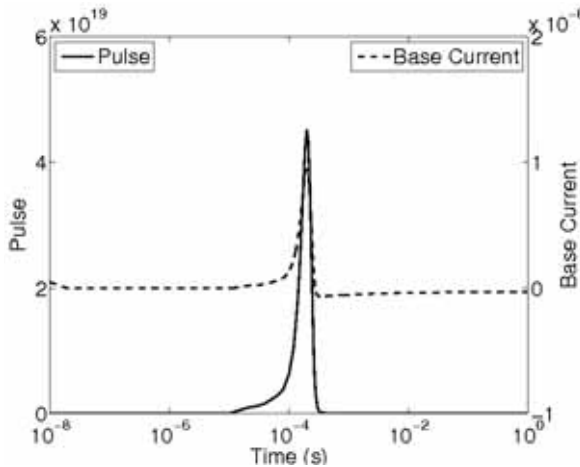


FIG. 5.1. Radiation pulse [12]

Our goal is to investigate the current $I(t, \xi)$ at the base contact as a function of time and

the uncertain reaction parameters. In principle one could use the model for the reaction parameters described in (2.5) to perform a collocation study that would generate an approximate response surface over the entire parameter space. However it is computationally intractable to do this on the 127 parameter space that includes all of the reaction parameters. So first a form of model reduction is necessary.

In order to reduce the number of parameters to be included in the collocation study we first execute a “one-at-a-time” (OAT) study to estimate the sensitivity of the solution with respect to each individual parameter. For this, we fix 126 of the 127 uncertain parameters at their mean value and perform a one-dimensional collocation to approximate the function $I(\mu_1, \dots, \xi_k, \dots, \mu_M, t)$ as a function of time and a single reaction parameter ξ_k . This approach scales very well since it only requires the solution of a series of one dimensional problems. From the collocation approximation of this function we can compute estimates of the sensitivity of I with respect to a single parameter. We use two metrics to measure the sensitivity, σ_i , the standard deviation of the current with respect to the i^{th} parameter and $\frac{\mu_i}{I} \frac{dI}{d\xi_i} |_{\xi_i=\mu_i}$, the scaled sensitivity of parameter i evaluated at the mean parameter value. The evaluation of the scaled sensitivity is also investigated in [12]. These two measures are plotted for each of the 127 parameters at a variety of times in Figure 5.2. The plots in Figure 5.2 are normalized so that $\sum_{i=1}^{127} \sigma_i = 1$ and $\sum_{i=1}^{127} \frac{\mu_i}{I} \frac{dI}{d\xi_i} |_{\xi_i=\mu_i} = 1$. Thus each bar can be interpreted as the relative importance of parameter i with respect to each sensitivity metric.

We make two observations. First the standard deviation and the scaled sensitivity generally show broad agreement in which parameters are considered important. However there are a few instances where the two are noticeably different. The differences are attributable to the fact that the standard deviation is an inherently global measure of sensitivity whereas the point derivative is an inherently local measure. We believe that the standard deviation may be a more reliable sensitivity metric because it takes into account the behavior of the current over a range of parameter values rather than simply at a single point. Second, the current only exhibits sensitivity with respect to a relatively small number of parameters. A response surface constructed using only the 15 most sensitive parameters should be of similar accuracy to the response surface constructed using the full 127 parameter space.

We use the 15 most sensitive reaction parameters to perform a multi-dimensional collocation study. We perform a level 6 collocation method on the 15 dimensional parameter space. In order to simplify the problem further we use the scaled standard deviations of the current in each of these parameters as weights for an anisotropic sparse grid collocation method, as in [11]. This method requires the discretization and solution of (2.1) at 371 points in the reduced parameter space. This computation was performed on Sandia’s *Red Sky* supercomputer and took approximately 41 hours using 64 cores. The results of this multi-dimensional collocation study were post processed to examine the Sobol’ sensitivity indices and the probability distributions of the current at various time steps.

Figure 5.3 shows the cumulative distribution function of the current at a series of time steps. At time $t = 1 \times 10^{-4}$ the solution is nearly in a steady state defined by the initial boundary value because the radiation pulse hasn’t yet generated many defects and as expected, the CDF of the current is nearly a Heaviside function. Immediately after the pulse has ended at $t = 1 \times 10^{-3}$ the current exhibits a large degree of variability owing to the large number of Frenkel pairs generated by the radiation pulse. The solution then quickly settles down into a new steady state at time $t = 1 \times 10^{-2}$. The current at this steady state displays a relatively small amount of variation over the parameter space. The overall uncertainty in the device current varies as a function of time. This indicates that the total number of collocation points as well as the anisotropic weights should be allowed to vary as a function of time to optimally capture the behavior of the solution. Such a technique would require more intrusive modifi-

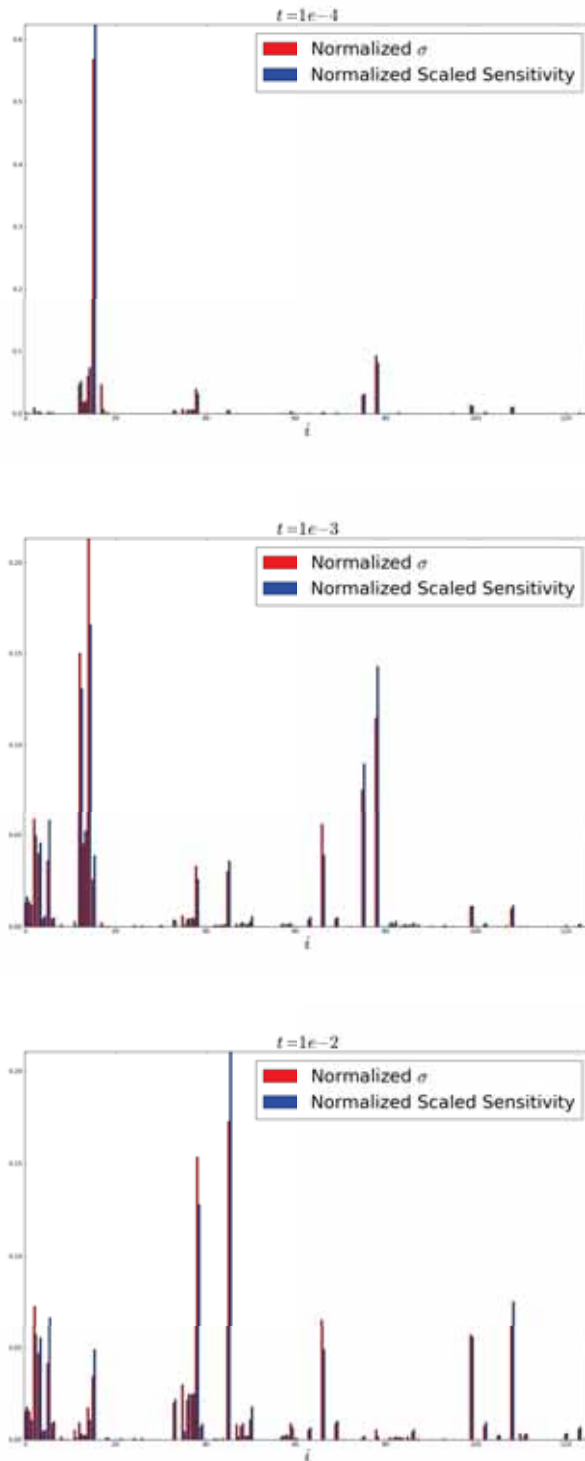


FIG. 5.2. Sensitivity Metrics at $t = 1 \times 10^{-4}$ (top), $t = 1 \times 10^{-3}$ (middle), and $t = 1 \times 10^{-2}$ (bottom)

cation of the existing code as the collocation method would need to be executed as part of the time series integrator.

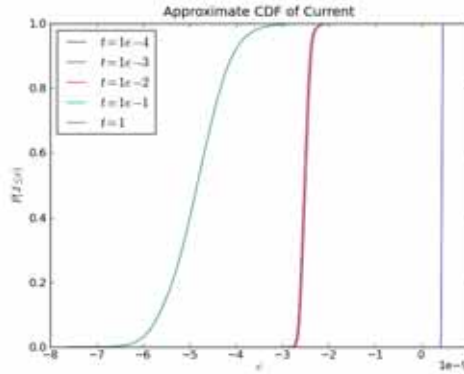


FIG. 5.3. CDF of Current at various time steps

Figure 5.4 shows the ratios $r_i = \frac{TS_i - S_i}{TS_i}$ computed from the 15 parameter study. In performing such a sensitivity analysis, it may be the case that parameter interactions account for only a small portion of the sensitivity in the current. In this case we would expect r_i to be close to zero for all of the parameters. This would indicate that most of the information contained in the response surface could be derived from the behavior of the current along each parameter axis. Figure 5.4 indicates that this is not the case. The figure reports that many of the parameters are coupled to produce changes in the current. Therefore we see that the OAT study does not contain enough information to reconstruct the response surface since it doesn't contain information regarding what is occurring in the corner of the parameter space. The fact that OAT studies do not explore the behavior of a function in the corners of the parameter domain is considered a major weakness. The data contained in the Sobol' indices can be used to isolate correlated parameters that should not be considered separately in OAT studies.

6. Conclusions. The goal of this paper was to determine what types of information could be obtained from applying modern uncertainty quantification techniques to large scale engineering problems. A great deal of information can be obtained from uncertainty quantification techniques that approximate the response functions on the entire parameter space as opposed to a single point. In particular, the standard deviation and Sobol' sensitivity indices provide insight to the global sensitivity of the response function with respect to the parameters. Also, collocation methods can be used to obtain approximations of the density function associated with the solution process which can be valuable as part of a reliability analysis. In the future we believe that it may be possible to use the results of a Sobol' sensitivity analysis to uncover hidden parameters which may lead to further reduced models and additional insights into the physics of the problem.

There are a number of additional areas which are natural extensions of this work. One possibility is the implementation of a time adaptive sparse grid collocation method to efficiently compute the response function at intermediate time steps. Another interesting possibility is to use the reduced model within an inverse formulation to assess a tolerable range of uncertainty within individual parameters. Understanding which of the problem's uncertainties are most important may help guide future experimentation. The possible dependence of the uncertain quantities also warrants further study.

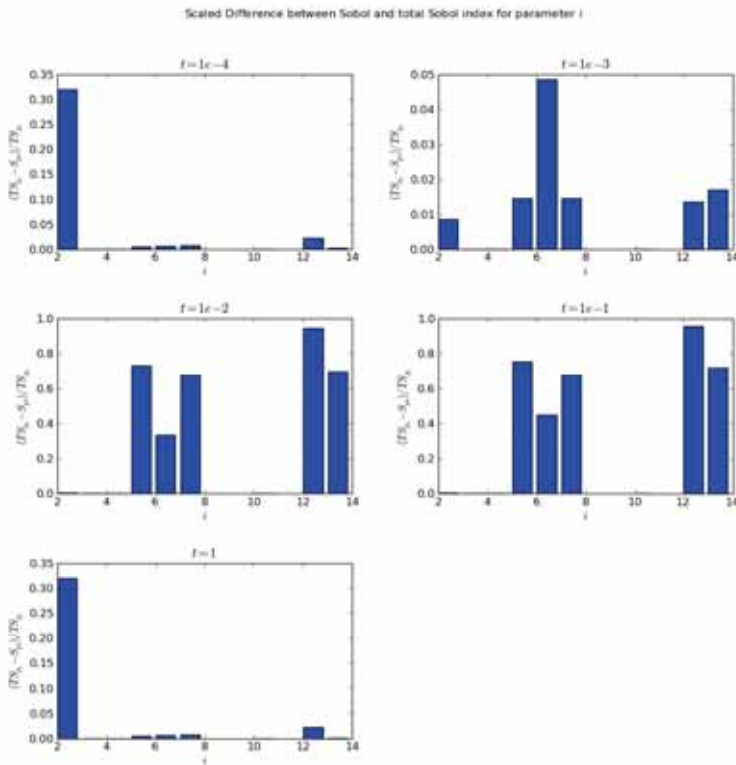


FIG. 5.4. $r_i = \frac{TS_i - S_i}{TS_i}$ computed for the 15 most sensitive parameters at various time steps.

REFERENCES

- [1] B. A. AMD W.J. BOHNHOFF, K. DALBEY, J. EDDY, M. ELDRED, D. GAY, K. HASKELL, P. HOUGH, AND L. SWILER, *Dakota, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: Version 5.0 users manual*, Tech. Rep. SAND2010-2183, Sandia National Laboratories, December 2009.
- [2] I. BABUŠKA, F. NOBILE, AND R. TEMPONE, *A stochastic collocation method for elliptic partial differential equations with random input data*, SIAM Journal on Numerical Analysis, 45 (2007), pp. 1005–1034.
- [3] J. BÄCK, F. NOBILE, L. TAMELLINI, AND R. TEMPONE, *Stochastic Galerkin and collocation methods for PDEs with random coefficients: a numerical comparison*, Tech. Rep. 09-33, Institute for Computational Engineering and Sciences, University of Texas at Austin, 2009. To appear in Proceedings of ICOSAHOM’09, Lecture Notes in Computational Science and Engineering, Springer-Verlag, New York.
- [4] A. BOBROWSKI, *Functional Analysis for Probability and Stochastic Processes: An Introduction*, Cambridge University Press, 2005.
- [5] G. HENNIGAN, R. HOEKSTRA, J. CASTRO, D. FIXEL, AND J. SHADID, *Simulation of neutron radiation damage in silicon semiconductor devices*, Tech. Rep. SAND2007-7151, Sandia National Laboratories, 2007.
- [6] R. J. KEE, F. M. RUPLEY, E. MEEKS, AND J. A. MILLER, *CHEMKIN-III: A fortran chemical kinetics package for the analysis of gas-phase chemical and plasma kinetics*, Tech. Rep. SAND96-8216, Sandia National Laboratories, 1996.
- [7] X. MA AND N. ZABARAS, *An adaptive hierarchical sparse grid collocation algorithm for the solution of stochastic differential equations*, Journal of Computational Physics, 228 (2009), pp. 3084–3113.
- [8] ———, *High-dimensional stochastic model representation technique for the solution of stochastic partial differential equations*, Journal of Computational Physics, 229 (2010), pp. 3884–3915.
- [9] N. METROPOLIS AND S. ULAM, *The monte carlo method*, Journal of the American Statistical Association, 44 (1949), pp. 335–341.

- [10] F. NOBILE AND R. TEMPONE, *A sparse grid stochastic collocation method for partial differential equations with random input data*, SIAM Journal on Numerical Analysis, 45 (2008), pp. 2309–2345.
- [11] F. NOBILE, R. TEMPONE, AND C. WEBSTER, *An anisotropic sparse grid collocation algorithm for the solution of stochastic differential equations*, SIAM Journal on Numerical Analysis, 46 (2008), pp. 2411–2442.
- [12] E. PHIPPS, R. BARTLETT, D. GAY, AND R. HOEKSTRA, *Large-scale transient sensitivity analysis of a radiation-damaged bipolar junction transistor via automatic differentiation*, Lecture Notes in Computational Science and Engineering, 64 (2008), pp. 351–362.
- [13] S. SMOLYAK, *Quadrature and interpolation formulas for tensor products of certain classes of functions*, Doklady Akademii Nauk, 148 (1963), pp. 1042–1043.
- [14] I. SOBOL, *Sensitivity estimates for nonlinear mathematical models*, Mathematical Modelling and Computational Experiments, 1 (1993), pp. 407–414.
- [15] S. M. SZE, *Physics of Semiconductor Devices*, Wiley and Sons, 2nd edition ed., 1981.
- [16] D. XIU AND J. HESTHAVEN, *High-order collocation methods for differential equations with random inputs*, SIAM Journal on Scientific Computing, 27 (2005), pp. 1118–1139.

KRYLOV RECYCLING FOR CLIMATE MODELING AND UNCERTAINTY QUANTIFICATION

KAPIL AHUJA*, MICHAEL L. PARKS†, ERIC T. PHIPPS‡, ANDREW G. SALINGER§, AND
ERIC DE STURLER¶

Abstract. Krylov subspace recycling is a technique to accelerate the convergence of sequences of slowly changing linear systems. Ice sheet modeling and embedded uncertainty quantification are two application areas where such systems arise. Typically, recycling algorithms assume that the number of vectors selected for recycling is less than the total number of iterations required to solve a system. Hence, these algorithms are useful when each system in the sequence requires a relatively large number of iterations to converge. For the application areas under study, the number of iterations required for convergence of a system is small. Hence, our existing family of recycling algorithms are not suitable in their current forms. However, this is not a problem inherent to recycling. We modify GCRO-DR such that the recycle space can be built even for rapidly converging linear systems. Hence, the number of vectors selected for recycling is no longer constrained by the total number of iterations required to solve a system.

GCRO-DR uses approximate invariant subspace as the recycle space. This choice is not always the best. We show in this paper that another choice works better for the application areas under consideration. We use Arnoldi vectors that have large components in the right hand side as the recycle space. In addition, since our systems converge rapidly, we modify GCRO-DR to avoid an extra orthogonalization step, leading to reduced cost. Numerical experiments show the benefit of using modified GCRO-DR for embedded uncertainty quantification. We show savings in iteration count as well as time. We also modify the recycling conjugate gradients (RCG) algorithm in a similar way. The details for RCG are not described here.

We also do performance studies on standard GCRO-DR and standard RCG for the above mentioned application areas. The results show that recycling does not impose substantial overhead. The experiments are done using a combination of recycling solvers implemented in the Belos package of the Trilinos project and Matlab.

1. Introduction. Recycling algorithms are appropriate for solving sequences of linear systems

$$A^{(i)}x^{(i)} = b^{(i)}, \quad (1.1)$$

where $A^{(i)} \in \mathbb{C}^{n \times n}$ and $b^{(i)} \in \mathbb{C}^n$ vary with i , and the matrices $A^{(i)}$ are large and sparse. For any particular system in the sequence, recycling algorithms store a recycle space associated with that system, and use this subspace to accelerate convergence for the next system in the sequence. This idea originates from ‘thick restarting’ used for solving a single linear system in the GCROT [2] and the GMRES-DR [12] algorithms. For solving a sequence of linear systems, ‘recycling’ was first proposed in [13] where it is applied to the GCROT and the GCRO-DR algorithms. The idea is further adapted in the recycling minimal residual algorithm (RMINRES [19]), the recycling conjugate gradients algorithm (RCG [14]), and the recycling bi-conjugate gradients (RBiCG [1]) algorithm.

Such sequences of linear systems arise in many application areas. We focus on two such areas; climate modeling [16] and embedded uncertainty quantification [11, 10, 9]. The linear systems arising in these application areas have ill-conditioned matrices. Thus, expensive preconditioners are required for the Krylov methods to converge in a reasonable number of iterations. Hence, it is worthwhile to look for methods that can help reduce the iteration count.

The matrices arising in climate modeling are nonsymmetric and not positive real, hence we use GCRO-DR, a recycling variant of GMRES [18]. The embedded uncertainty quantification application consists of a stochastic, elliptic, partial differential equation (PDE) [11],

*Department of Mathematics, Virginia Tech, kahuja@vt.edu

†Applied Mathematics and Applications, Sandia National Laboratories, mlparks@sandia.gov

‡Optimization & Uncertainty Quantification, Sandia National Laboratories, ethipp@sandia.gov

§Applied Mathematics and Applications, Sandia National Laboratories, agsalin@sandia.gov

¶Department of Mathematics, Virginia Tech, sturler@vt.edu

and ideally should lead to matrices that are symmetric positive definite (SPD). However, the current implementation of boundary conditions in the code based on [11] destroys the symmetry in the matrices. The system still remains B-normal(1) and hence the CG method works [6, 3]. Therefore, we use both RCG (which is a recycling variant of CG) and GCRO-DR for this application. A GMRES based recycling solver for stochastic elliptic PDEs is used in [10] as well, where the authors use a recycling FGMRES [17].

Recycling algorithms typically have two input parameters that control how the recycle space is built. The first parameter controls the frequency of updates of the recycle space. Ideally, the recycle space should be updated at the end of the solve, but that would be expensive if the system does not converge quickly (see [13] for details). The iteration process between two updates of the recycle space is referred to as a ‘cycle’. The length of the cycle, m , refers to the number of iterations between updates of the recycle space. The second parameter, usually denoted by k , controls the size of the recycle space. Typically, recycling algorithms are designed such that k is less than the expected number of iterations required for convergence of the average system (denoted by `niter`). This is a reasonable assumption because recycling algorithms target applications where `niter` is large. For the two application areas under consideration, the linear systems converge quickly, due to the application of effective but also expensive preconditioners. Hence, k must be very small. Because of this, the recycle space does not contain sufficient information to accelerate convergence, and recycling does not show much benefit. We modify the GCRO-DR algorithm such that k can be set greater than `niter`. We show in Section 4 that this modification leads to faster convergence.

In many cases, deflation of eigenvalues close to the origin improves the convergence rate [12]. Hence, current recycling algorithms use approximate invariant subspaces corresponding to small eigenvalues (in magnitude) as the recycle space (eigenvalues can be deflated by including the corresponding eigenvectors in the Krylov subspace). However, this approach does not work well for the applications we consider. Therefore, we propose an alternative mechanism for the recycle subspace selection. We build our recycle space from Arnoldi vectors that have a large component in the right hand side. We modify the GCRO-DR algorithm with this new recycle subspace selection approach. The results show that this strategy improves convergence. We also optimize the GCRO-DR code exploiting the fact that each system in our sequence of systems converges quickly. The main optimization involves avoiding an extra orthogonalization step. This reduces the runtimes for our algorithm. We do the above three modifications ($k \geq \text{code_niter}$, different recycle subspace selection criteria, and code optimizations) to the RCG algorithm also. As the modifications are similar we do not discuss them here.

To show that recycling does not impose substantial overhead, we do performance studies on the standard GCRO-DR algorithm and the standard RCG algorithm.

The rest of the paper is divided as follows. In Section 2, we give a brief description of the two application areas. We describe the standard GCRO-DR algorithm and the three modifications ($k \geq \text{code_niter}$, different recycle subspace selection criteria, and code optimizations) in Section 3. In Section 4, we give results from solving the embedded uncertainty quantification application by modified GCRO-DR. We also show performance results for standard GCRO-DR for climate modeling and standard RCG for embedded uncertainty quantification. For the experiments, we use a combination of recycling solvers implemented in the Belos package of the Trilinos project [5] and Matlab. The conclusions and future work are discussed in Section 5.

2. Application Areas. Climate modeling consists of the following four components: an atmospheric model, an ice model, a land model, and an ocean model. We focus on ice sheet modeling using Glimmer, a community ice sheet model [16]. The ice sheet thickness (H) is

described by the following continuity equation [4, 15]:

$$\frac{\partial H}{\partial t} = B - \nabla \cdot (\bar{u}H), \quad (2.1)$$

where B is the surface mass balance, \bar{u} is the average velocity of the ice sheet in the vertical direction, and ∇ is the gradient operator (horizontal). To model large ice sheets, shallow ice approximation is commonly used. This approximation assumes that the bedrock and the ice surface have small slopes. Hence, the normal stress components are neglected [4, 7]. Finite difference discretization of the resulting equation, on a staggered grid with periodic boundary conditions, leads to a sequence of linear systems.

Embedded uncertainty quantification involves the following stochastic linear elliptic PDE [11]:

$$\nabla \cdot (a(\vec{x}, \xi) \nabla u(\vec{x}, \xi)) = f \quad (2.2)$$

in the domain $[-0.5, 0.5]^2$ with zero Dirichlet boundary conditions. In (2.2), ξ is the input random variable. The diffusivity is randomized and is given by

$$a(\vec{x}, \xi) = 1 + \sigma \frac{1}{\pi^2} \xi \cos \left[\frac{\pi}{2} (x^2 + y^2) \right], \quad (2.3)$$

where σ is the standard deviation of the random field a . The forcing function f is chosen by applying the following exact solution to (2.2):

$$u(\vec{x}, \xi) = \exp(-|\xi|^2) 16(x^2 - 0.25)(y^2 - 0.25). \quad (2.4)$$

Solving this stochastic PDE (2.2) by a sparse grid collocation method leads to a set of uncoupled, deterministic PDEs. We use a finite difference discretization on a 5-point stencil to obtain a sequence of linear systems (for details see [11]).

3. Standard and Modified GCRO-DR. In Sections 3.1 and 3.2, we briefly describe the theory behind the standard GCRO-DR algorithm and its modified version, respectively. The pseudo-code for our modified GCRO-DR is given in the appendix. The changes from standard GCRO-DR are marked in blue. For the pseudo-code of the standard GCRO-DR algorithm, see the appendix in [13].

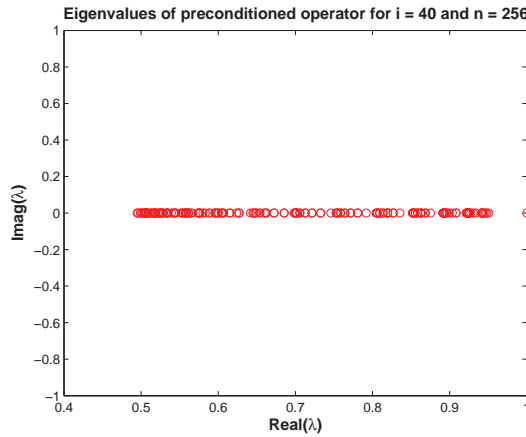
3.1. Standard GCRO-DR. After solving the i^{th} system in (1.1), GCRO-DR retains k approximate eigenvectors of $A^{(i)}$, which are used to compute the matrices $U, C \in \mathbb{C}^{n \times k}$, such that $\text{range}(U)$ is an approximate invariant subspace of $A^{(i)}$ (and hopefully also of $A^{(i+1)}$), $A^{(i+1)}U = C$, and $C^H C = I$.

GCRO-DR uses a modified Arnoldi process to compute the orthogonal basis for the Krylov subspace such that each new Krylov vector is also orthogonal to $\text{range}(C)$. This produces the Arnoldi relation

$$(I - CC^H)A^{(i+1)}V_m = V_{m+1}\underline{H}_m, \quad (3.1)$$

where \underline{H}_m is an $(m+1) \times m$ upper Hessenberg matrix. GCRO-DR finds the optimal solution over the (direct) sum of the recycle space, $\text{range}(U)$, and the new search space, $\text{range}(V_m)$.

3.2. Modifications to Standard GCRO-DR. We discuss the three modifications to the standard GCRO-DR algorithm that make it attractive for sequences with rapidly converging linear systems. The first modification adapts the algorithm for the case where the number of vectors selected for recycling can be set greater than (or equal to) the number of iterations

FIG. 3.1. *Eigenvalue Distribution.*

required for convergence ($k \geq \text{niter}$). One assumption that we maintained from the standard GCRO-DR is that the recycled subspace size be less than the length of a cycle, or

$$k < m. \quad (3.2)$$

Previously the following combinations were possible:

$$k < m \leq \text{niter} \quad \text{and} \quad k < \text{niter} \leq m. \quad (3.3)$$

Now, the following is also possible:

$$\text{niter} \leq k < m. \quad (3.4)$$

The second modification involves a different recycle subspace selection criterion. As for any deflation based recycling algorithm, GCRO-DR uses an approximate invariant subspace (corresponding to the eigenvalues close to the origin) to build the recycle space. It turns out that for our applications, the spectrum of the preconditioned operator has clustered eigenvalues that are well-separated from the origin (see Figure 3.1). Thus, this criterion does not work well. Hence, we focus on the dominant components of the right-hand side to build the recycle space. A similar approach has been used in [8] (also see [2]). Let $\text{range}(U)$ be the recycle space available from the previous linear system, and $\text{range}(V_m)$ be the new search space from (3.1). We first build the matrix¹

$$\tilde{V}_p = \begin{bmatrix} U & V_p \end{bmatrix}, \quad (3.5)$$

and, then pick those k vectors of \tilde{V}_p that have a large component in the right hand side. The GCRO-DR implementation is modified such that the user can specify the recycle subspace selection criterion via the input parameter `crit`. Currently, the two options for it are `eigen`, for approximate eigenvectors based criterion, and `arnoldi`, for this new component based criterion.

The third modification avoids the last orthogonalization in the algorithm². When $k \geq \text{niter}$, the algorithm performs just one cycle. This implies that the recycle space generated

¹This excludes the first system in the sequence, which requires special handling.

²Again, we have not done this for the first system in the sequence.

in the current system, $A^{(i)}x = b^{(i)}$, is not used as a search space for this same system, rather, it is used as the search space for the next system in the sequence, $A^{(i+1)}x = b^{(i+1)}$. Therefore, the computation for enforcing

$$C^H C = I \text{ with } C = A^{(i)}U \quad (3.6)$$

can be avoided. This does not affect the execution of the subsequent solves, because, at the start of solving the $(i + 1)^{th}$ linear system, the following is always enforced:

$$C^H C = I \text{ with } C = A^{(i+1)}U. \quad (3.7)$$

For details, see the pseudo-code in the appendix (lines 45-51 and 54-58).

4. Numerical Results. We give results from solving the embedded uncertainty quantification application by modified GCRO-DR in Section 4.1. The performance studies of standard GCRO-DR for climate modeling and standard RCG for embedded uncertainty quantification are described in Section 4.2. For experiments in both the subsections, relative convergence tolerance was set to 10^{-6} .

4.1. Modified GCRO-DR for Embedded Uncertainty Quantification. We used the Trilinos software package (version 10.2) to generate a sequence of preconditioned linear systems from the embedded uncertainty quantification application (example twoD_diffusion_collocation_example.cpp in the Stokhos package). The application generated a sequence of matrices, right hand sides, and multilevel preconditioners. The rest of the experimentation was done in Matlab. We did experiments on two system sizes, $n = 256$ and $n = 65536$. We compared the iteration count and the runtime of our modified GCRO-DR with the iteration count and runtime of Matlab's implementation of GMRES. The results are given in Figure 4.1. We used $m = 11$ and $k = 10$. The timing data are given in Table 4.1. As evident from the figure and the table, the saving in the iteration count is around 50%, and the saving in the runtime is close to 13%.

TABLE 4.1
Timing Comparison

n	GMRES Time	Modified GCRODR(11,10) Time	Saving in Time
256	2.51	2.19	12.92%
65536	238.46	208.34	12.63%

4.2. Performance Study. The experiments in this subsection were done using the Trilinos and the Glimmer software packages. We studied performance of standard GCRO-DR with climate modeling as the target application. An ILU type preconditioner was used. The code was profiled to measure the time for recycling, the time for the recycling orthogonalization step (i.e., application of $(I - CC^H)$ in (3.1)), and the time for application of the operator and the preconditioner. The recycling time includes all computations involving use of the recycle space (using U and C), and all computations done to generate the recycle space (generating U and C). Hence, the time for the recycling orthogonalization step is included in the time for recycling. The total time to solve a linear system consists of the time for recycling, the time for application of the operator and the preconditioner, the time for initializations (not reported here), and the time for the other steps of the standard GMRES algorithm e.g. normal orthogonalization time for the Arnoldi basis (not reported here). This was done for several values of m and k . The results for $n = 7904$, $m = 8$, and $k = 4$ are given in Figure 4.2 (a). It

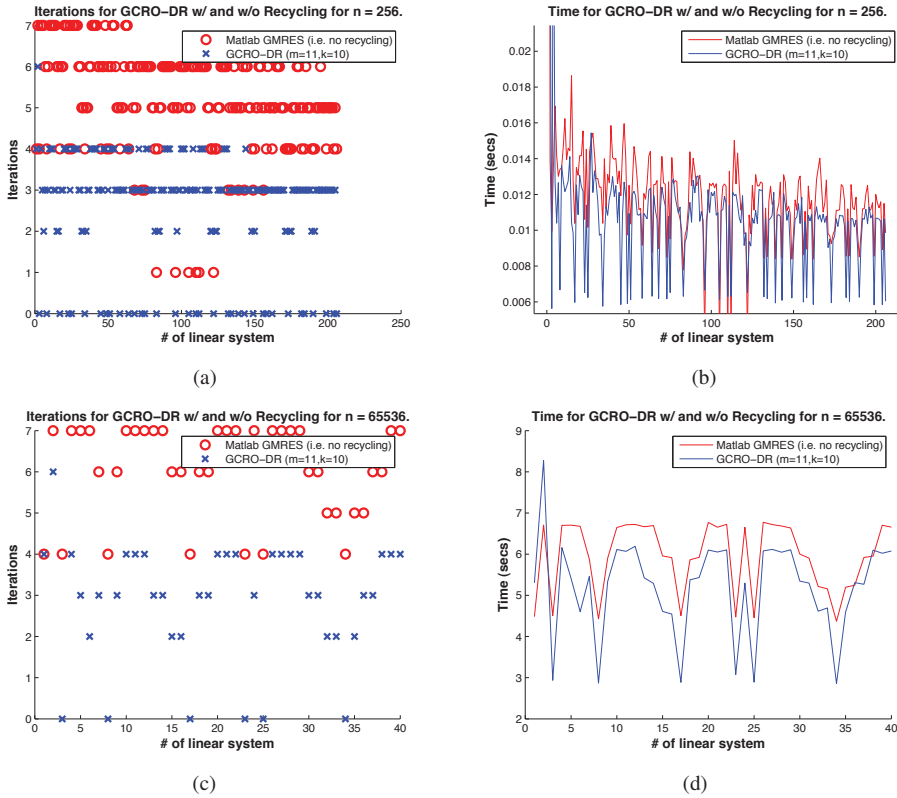


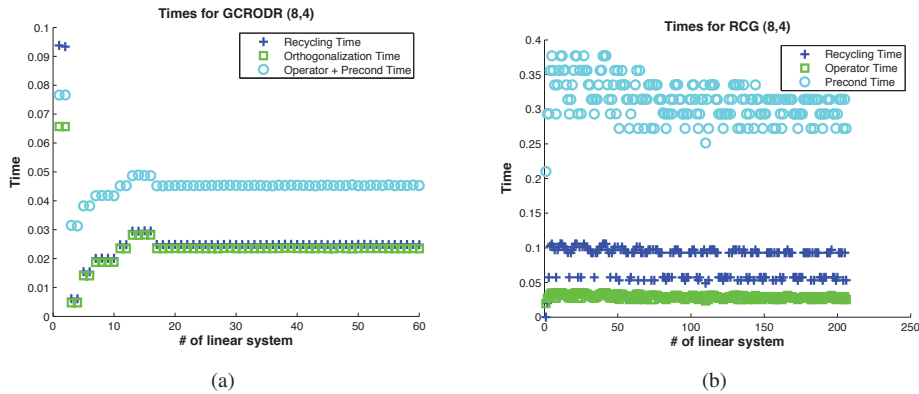
FIG. 4.1. Iteration and Timing Plots.

is clear from the figure that the recycling time is modest compared with the time required to apply the operator and the preconditioner.

For the performance study of RCG, we used the embedded uncertainty quantification application³. As in Section 4.1, a multilevel preconditioner was used. The code was profiled to measure the time for recycling, the time for application of the operator, and the time for multiplying by the preconditioner. As for GCRO-DR, the recycling time includes all computations involving use of the recycle space and generation of the recycle space. The total time to solve a linear system is the sum of the time for recycling, the time for application of the operator, the time for multiplying of the preconditioner, the time for initializations (not reported here), and time for other steps of the standard CG algorithm e.g. computing standard iteration vectors x , r , and p (not reported here). Again, the experiments were done for several values of m and k . The results for $n = 65536$, $m = 8$, and $k = 4$ are given in Figure 4.2 (b). As for GCRO-DR, the recycling time in RCG is modest compared with the time to apply the operator and the preconditioner.

5. Conclusions and Future Work. In this paper, we discuss the modifications to the existing GCRO-DR algorithm for rapidly converging sequence(s) of linear systems. The number of vectors selected for recycling can now be set greater than the total iterations required for convergence of a linear system. We present a new criterion for selecting the recycle sub-

³The original application used AztecOO for linear solvers. We switched it to use Belos. This also demonstrated the use of ML preconditioner with Belos. We also fixed a memory leak encountered in the original application.

FIG. 4.2. *Performance Study Results.*

space, and also incorporate few code optimizations. We show that this modified GCRO-DR solves the embedded uncertainty quantification application faster compared with the standard GMRES algorithm. Similar modifications are done to the existing RCG algorithm but are not described here. We also do performance studies on standard GCRO-DR and standard RCG to show that recycling does not impose a substantial overhead.

Currently, we are fixing the boundary condition implementation for the embedded uncertainty quantification application. This will lead to SPD systems. Since CG is the optimal method for SPD systems, we can then use modified RCG. We plan to modify the implementation of GCRO-DR and RCG in Trilinos, and use them to generate new numerical results.

Future work involves two tasks. Firstly, we plan to profile the GCRO-DR and RCG codes for memory usage using Tau. Secondly, we plan to implement a grouping strategy for the sequence of linear systems arising in the embedded uncertainty quantification application. This strategy has been shown to accelerate the recycling algorithms in [10].

REFERENCES

- [1] K. AHUJA, *Recycling Bi-Lanczos Algorithms: BiCG, CGS, and BiCGSTAB*, MS Thesis, Department of Mathematics, Virginia Tech, <http://scholar.lib.vt.edu/theses/available/etd-08252009-161256/>, 2009.
- [2] E. DE STURLER, *Truncation strategies for optimal Krylov subspace methods*, SIAM Journal on Numerical Analysis, 36 (1999), pp. 864–889.
- [3] A. GREENBAUM, *Iterative Methods for Solving Linear Systems*, SIAM, Philadelphia, PA, USA, 1997.
- [4] M. HAGDORN, I. RUTT, T. PAYNE, AND F. HEBELER, *GLIMMER 1.5.1 Documentation*, <http://www.cesm.ucar.edu/models/cesm1.0/cism/docs/glimmer.pdf>, 2010.
- [5] M. A. HEROUX, R. A. BARTLETT, V. E. HOWLE, R. J. HOEKSTRA, J. J. HU, T. G. KOLDA, R. B. LEHOUCQ, K. R. LONG, R. P. PAWLOWSKI, E. T. PHIPPS, A. G. SALINGER, H. K. THORNQUIST, R. S. TUMINARO, J. M. WILLENBRING, A. WILLIAMS, AND K. S. STANLEY, *An overview of the Trilinos project*, ACM Trans. Math. Softw., 31 (2005), pp. 397–423.
- [6] M. R. HESTENES AND E. STIEFEL, *Methods of conjugate gradients for solving linear systems*, Journal of Research of the National Bureau of Standards, 49 (1952), pp. 409–436.
- [7] K. HUTTER, *Theoretical Glaciology*, Mathematical Approaches to Geophysics, D. Reidel Publishing Company, Dordrecht, Boston, Lancaster, 1983.
- [8] C. P. JACKSON AND P. ROBINSON, *A numerical study of various algorithms related to the preconditioned conjugate gradient method*, Internat. J. Numer. Methods Engrg., 21 (1985), pp. 1315–1338.
- [9] C. JIN AND X.-C. CAI, *A preconditioned recycling GMRES solver for stochastic Helmholtz problems*, Communications in Computational Physics, 6 (2009), pp. 342–353.
- [10] C. JIN, X.-C. CAI, AND C. LI, *Parallel domain decomposition methods for stochastic elliptic equations*, SIAM Journal on Scientific Computing, 29 (2007), pp. 2096–2114.

- [11] C. W. MILLER, R. S. TUMINARO, E. T. PHIPPS, AND H. C. ELMAN, *Assessment of collocation and Galerkin approaches to stochastic partial differential equations*, in CSRI Summer Proceedings 2009.
- [12] R. B. MORGAN, *GMRES with deflated restarting*, SIAM Journal on Scientific Computing, 24 (2002), pp. 20–37.
- [13] M. L. PARKS, E. DE STURLER, G. MACKEY, D. D. JOHNSON, AND S. MAITI, *Recycling Krylov subspaces for sequences of linear systems*, SIAM Journal on Scientific Computing, 28 (2006), pp. 1651–1674.
- [14] M. L. PARKS, P. K. V. V. NUKALA, AND S. ŠIMUNOVIĆ, *Efficient simulation of large-scale 3D fracture networks via Krylov subspace recycling*. Paper Draft, 2010.
- [15] A. J. PAYNE AND P. W. DONGELMANS, *Self-organization in the thermomechanical flow of ice sheets*, J. Geophys. Res., 102 (1997), pp. 12219–12233.
- [16] I. C. RUTT, N. R. J. HULTON, AND A. J. PAYNE, *The Glimmer community ice sheet model*, J. Geophys. Res., 114 (2009).
- [17] Y. SAAD, *A flexible inner-outer preconditioned GMRES algorithm*, SIAM Journal on Scientific Computing, 14 (1993), pp. 461–469.
- [18] Y. SAAD AND M. H. SCHULTZ, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM Journal on Scientific and Statistical Computing, 7 (1986), pp. 856–869.
- [19] S. WANG, E. DE STURLER, AND G. H. PAULINO, *Large-scale topology optimization using preconditioned Krylov subspace methods with recycling*, International Journal for Numerical Methods in Engineering, 69 (2006), pp. 2422–2468.

Appendix

A. Pseudo-Code for the Modified GCRO-DR Algorithm.

- 1: Choose m , the length of the cycle, k , the desired number of recycle vectors, and **crit**, the recycle space selection criteria (**eigen** or **arnoldi**). Let tol be the convergence tolerance. Choose an initial guess x_0 . Compute $r_0 = b - Ax_0$, and set $i = 1$.
- 2: **if** \tilde{Y}_s is defined (from a previous system; s is number of columns of \tilde{Y}_s) **then**
- 3: Call **buildUC**.
- 4: $x_1 = x_0 + U_s C_s^H r_0$
- 5: $r_1 = r_0 - C_s C_s^H r_0$
- 6: **else**
- 7: $v_1 = r_0 / \|r_0\|_2$
- 8: $c = \|r_0\|_2 e_1$
- 9: Perform m steps of GMRES, solving $\min \|c - \underline{H}_m y\|_2$ for y . Let GMRES converges in p steps generating V_{p+1} and \underline{H}_p .
- 10: $x_1 = x_0 + V_p y$
- 11: $r_1 = V_{p+1}(c - \underline{H}_p y)$
- 12: **if** $k \geq p$ **then**
- 13: $s = p$
- 14: $\tilde{Y}_s = V_p$
- 15: Call **buildUC**.
- 16: **else**
- 17: $s = k$
- 18: Compute the k eigenvectors \tilde{z}_j of $(H_p + h_{p+1,p}^2 H_p^{-H} e_p e_p^H) \tilde{z}_j = \tilde{\theta}_j \tilde{z}_j$ associated with the smallest magnitude eigenvalues $\tilde{\theta}_j$ and store in P_k .
- 19: $\tilde{Y}_s = V_p P_k$
- 20: Let $[Q, R]$ be the reduced QR-factorization of $\underline{H}_p P_k$.
- 21: $C_s = V_{p+1} Q$
- 22: $U_s = \tilde{Y}_s R^{-1}$
- 23: **end if**
- 24: **end if**
- 25: **while** $\|r_i\|_2 > \text{tol}$
- 26: $i = i + 1$
- 27: Perform m Arnoldi steps with the linear operator $(I - C_s C_s^H)A$, letting $v_1 =$

$r_{i-1}/\|r_{i-1}\|_2$. Again, let the algorithm converges in p steps generating V_{p+1} , \underline{H}_p , and B_p .

```

28:   Let  $D_s$  be a diagonal scaling matrix such that  $\widetilde{U}_s = U_s D_s$  where the columns of
 $\widetilde{U}_s$  have unit norm.
29:    $t = s + p$ 
30:    $\widehat{V}_t = [\widetilde{U}_s \ V_p]$ 
31:    $\widehat{W}_{t+1} = [C_s \ V_{p+1}]$ 
32:    $\underline{G}_t = \begin{bmatrix} D_s & B_p \\ 0 & \underline{H}_p \end{bmatrix}$ 
33:   Solve  $\min \| \widehat{W}_{t+1}^H r_{i-1} - \underline{G}_t y \|_2$  for  $y$ .
34:    $x_i = x_{i-1} + \widehat{V}_t y$ 
35:    $r_i = r_{i-1} - \widehat{W}_{t+1} \underline{G}_t y$ 
36:   if  $k \geq t$  then
37:      $s = t$ 
38:      $\widetilde{Y}_s = \widehat{V}_t$ 
39:     Call buildUC.
40:   else
41:      $s = k$ 
42:     if crit == eigen then
43:       Compute the  $k$  eigenvectors  $\widetilde{z}_i$  of  $\underline{G}_t^H \underline{G}_t \widetilde{z}_i = \widetilde{\theta}_i \underline{G}_t^H \widehat{W}_{t+1}^H \widehat{V}_t \widetilde{z}_i$  associated with
smallest magnitude eigenvalues  $\widetilde{\theta}_i$  and store in  $P_k$ .
44:        $\widetilde{Y}_s = \widehat{V}_t P_k$ 
45:       if  $p == m$  then
46:         Let  $[Q, R]$  be the reduced QR-factorization of  $\underline{G}_t P_k$ .
47:          $C_s = \widehat{W}_{t+1} Q$ 
48:          $U_s = \widetilde{Y}_s R^{-1}$ 
49:       else
50:          $U_s = \widetilde{Y}_s$ 
51:       end if
52:     else
53:       Pick those  $k$  columns of  $\widehat{V}_t$  that have large components in right hand side
and store in  $\widetilde{Y}_s$ .
54:       if  $p == m$  then
55:         Call buildUC.
56:       else
57:          $U_s = \widetilde{Y}_s$ 
58:       end if
59:     end if
60:   end if
61: end while
62: Let  $\widetilde{Y}_s = U_s$  (for the next system)

```

buildUC

- 1: Let $[Q, R]$ be the reduced QR-factorization of $A \widetilde{Y}_s$.
- 2: $C_s = Q$
- 3: $U_s = \widetilde{Y}_s R^{-1}$

STABILITY OF ORDINARY DIFFERENTIAL EQUATIONS WITH COLORED NOISE FORCING

TIMOTHY J. BLASS[§] AND LOUIS A. ROMERO[¶]

Abstract. We present a method for determining the stability of a class of stochastically forced ordinary differential equations, where the forcing term can be of quite general form. We use the Fokker-Planck equation to write a low-order partial differential equation for the second moments, which we turn into an eigenvalue problem for a second-order differential operator. The eigenvalues of this operator determine the stability of the system. Inspired by Dirac's creation and annihilation operator method, we develop "ladder" operators to determine analytic expressions for the eigenvalues and eigenfunctions of the operator.

1. Introduction. The original goal of this work was to develop a framework in which to analyze the stability of the stochastically forced Mathieu equation:

$$\ddot{x} + \gamma\dot{x} + (\omega_0^2 + \varepsilon f(t))x = 0, \quad (1.1)$$

where the stochastic process $f(t)$ is determined by

$$\begin{aligned} \dot{s} &= -\kappa s + \sigma n(t), \\ \dot{u} &= -\alpha u + \beta s, \\ f(t) &= \alpha s(t) - \beta u(t). \end{aligned} \quad (1.2)$$

$n(t)$ is "white noise", meaning that $dW_t = n(t)dt$, where W_t is a Brownian motion. We refer to f as a *second-order filter* because it is generated by a second order ODE system. Equation (1.1) is a model for the dispersion relation of a capillary wave in a time-varying gravitational field, with $f(t)$ producing the random fluctuations in acceleration, [5]. If $\alpha, \kappa > 0$, the stochastic process (1.2), is stationary as $t \rightarrow \infty$, with power spectral density

$$S(\omega) = \frac{\sigma \alpha^2 \omega^2}{(\omega^2 + \kappa^2)(\omega^2 + \beta^2)}.$$

This is of interest for applications where the forcing term represents an acceleration, because acceleration is the second derivative position, so its Fourier transform should have the form $S(\omega) = \omega^2 g(\omega)$ where g is continuous at zero. An example profile for $S(\omega)$ is shown in Figure 1.1.

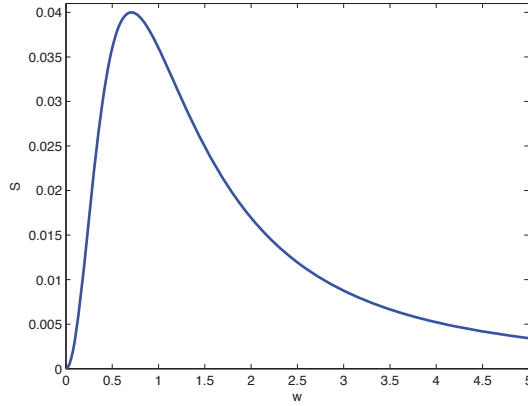
As explained below, the stability is determined by solving an eigenvalue problem for a differential operator derived from the stochastic differential equations. The stability of (1.1) was determined using numerical and perturbation techniques. The analysis begins with the simplified case of a *first-order filter* forcing a first-order equation was very useful. The first-order filter is obtained by eliminating the equation for u in (1.2). This simpler system has the form

$$\begin{aligned} \dot{s} &= -\kappa s + \sigma n(t), \\ \dot{x} &= (-\gamma + \varepsilon f(t))x. \end{aligned} \quad (1.3)$$

This form of f no longer represents an acceleration, but the eigenvalues and eigenfunctions that determine the stability have analytic expressions. The eigenfunctions are given in terms of Hermite polynomials, and the eigenvalues are of the form $\{-n\alpha - 2\gamma + 2\frac{\varepsilon^2}{\kappa}\}_{n \geq 0}$. Using

[§]The University of Texas at Austin, Department of Mathematics, tblast@math.utexas.edu

[¶]Sandia National Laboratories, lromero@sandia.gov

FIG. 1.1. $S(\omega)$ vs ω for $\alpha = 0.3, \kappa = 0.5, \beta = 1, \sigma = 1$

the Hermite polynomials as a basis was key to generalizing the analysis to more complicated systems, and was an important part of developing our numerical approach. This is explained in detail in Section 3.

The next step towards understanding the stability of solutions to (1.3) and the stability of solutions to (1.1) is understanding the stability of solutions to a first-order equation that is forced with a second-order filter. This has the form

$$\begin{aligned}\dot{s} &= -\kappa s + \sigma n(t), \\ \dot{u} &= -\alpha u + \beta s \\ \dot{x} &= -\gamma x + \varepsilon f(t)x, \\ f(t) &= as(t) + bu(t).\end{aligned}\tag{1.4}$$

All parameters in (1.7) are real, and $\kappa, \alpha, \gamma \geq 0$ so, in the absence of noise, the solutions would be bounded. This is not the most general form of a second-order filter, but is a physically relevant form. Surprisingly, we observed numerically and confirmed with perturbation theory that the eigenvalues that determine the stability of (1.4) appeared in constant increments, and had an ε^2 dependence, just as in the case of system (1.2). This led us to search for analytic expressions for the eigenvalues and eigenfunctions, which we have obtained. In doing so, we found that the framework extends to an n -th order filter, similar to the form (1.7) but with n equations generating f . That is, of the form

$$\begin{aligned}\dot{\mathbf{s}} &= -\mathbf{B}\mathbf{s} + \mathbf{n}(t), \\ \dot{x} &= -\gamma x + \varepsilon \mathbf{c}^\top \mathbf{s}(t)x,\end{aligned}\tag{1.5}$$

\mathbf{c} and $\mathbf{s}(t)$ are n -vectors, and \mathbf{B} is an $n \times n$ matrix, and $\mathbf{n}(t)$ is an n -vector of white noises.

1.1. Stability. We determine the stability of (1.4) by the long-time behavior of the second moments of the solutions. The case of forcing by white noise has been studied extensively, but white noise is not realistic as a forcing term. In addition, if the system is linear and the forcing is by white noise, then the associated equations for the moments of the solutions are a simple system of ordinary differential equations (see [1]). The nonlinear term $f(t)x$ in our system precludes this type of approach to our study of stability. Van Kampen has presented a heuristic approach to the case of colored noise, [6]. We improve on this by

developing a rigorous theory for colored noise forcing. We consider the second moment in x as a function of s and u , as well as t :

$$M_{xx}(s, u, t) = \int_{\mathbb{R}} x^2 P(x, s, u, t) dx. \quad (1.6)$$

In the most general case of a second-order filter. That is,

$$\begin{aligned} \dot{s} &= -\kappa s + \nu u + \sigma n_1(t), \\ \dot{u} &= -\alpha u + \beta s + \rho n_2(t) \\ \dot{x} &= -\gamma x + \varepsilon f(t)x, \\ f(t) &= as(t) + bu(t), \end{aligned} \quad (1.7)$$

M_{xx} will satisfy a PDE in s, u, t , and in the case of a first-order filter, the equation will be only in s, t . The function $P(x, s, u, t)$ in (1.6) is the joint probability density function for (1.7), which is the solution to the associated Fokker-Planck equation, with $f(t) = as(t) + bu(t)$,

$$\begin{aligned} \partial_t P &= \frac{\sigma^2}{2} \partial_s^2 P + \frac{\rho^2}{2} \partial_u^2 P + \partial_s[(\kappa s - \nu u)P] + \\ &\quad + \partial_u[(\alpha u - \beta s)P] + \partial_x[(\gamma - \varepsilon(as + bu))xP]. \end{aligned} \quad (1.8)$$

If we multiply this equation by x^2 and integrate with respect to dx , we arrive at the equation for M_{xx} :

$$\begin{aligned} \partial_t M_{xx} &= \frac{\sigma^2}{2} \partial_s^2 M_{xx} + \frac{\rho^2}{2} \partial_u^2 M_{xx} + \partial_s[(\kappa s - \nu u)M_{xx}] + \\ &\quad + \partial_u[(\alpha u - \beta s)M_{xx}] + (-2\gamma + 2\varepsilon(as + bu))M_{xx}. \end{aligned} \quad (1.9)$$

For notational convenience, we define the operator \mathcal{D} by the operator giving the right-hand-side of (1.9). That is, we can rewrite (1.9) as $\partial_t M_{xx} = \mathcal{D}M_{xx}$. If \mathcal{D} has a complete set of eigenfunctions, then we can write a solution to (1.9) as a linear combination of the eigenvectors multiplied by an exponential in t . Thus, we are led to look for $M_{xx}(s, u, t) = e^{\lambda t} M(s, u)$. Such a function M would solve the eigenvalue problem

$$\lambda M = \mathcal{D}M. \quad (1.10)$$

If there is a solution to (1.10) with $\lambda > 0$, then the system (1.7) will be unstable. We find an analytic expression for the eigenvalues, λ and are thus able to predict the stability.

As explained above, two useful and illuminating examples are for the cases (1.3) and (1.4). Another motivation for studying (1.3) is that the form of f , while no longer representing an acceleration, is a commonly used process (Ornstein-Uhlenbeck) in modeling, so it should be of interest to those interested in many applications. The result in this case, which is proved in Section 2.1, is

THEOREM 1.1. *The solutions to (1.3) are stable for $0 \leq \varepsilon \leq \frac{\kappa \sqrt{\gamma}}{|\sigma|}$. For the second-order filter, we have the following theorem, as proved in Section 2.2.*

THEOREM 1.2. *The solutions to (1.4) are stable for $0 \leq \varepsilon \leq \frac{\kappa \alpha \sqrt{\gamma}}{|\sigma|[\alpha^2 - \beta^2]}$.*

The operator \mathcal{D} shares similar structure to the quantum harmonic oscillator, which Dirac solved using creation and annihilation operators (also called ladder operators), see [3]. Inspired by this work, we determine the eigenvalues and eigenfunctions of \mathcal{D} via ladder operators. A ladder operator is any operator X , such that

$$[\mathcal{D}, X] = \mu X, \quad (1.11)$$

for $\mu \neq 0$. Here $[\cdot, \cdot]$ denotes the commutator. If $\mathcal{D}\phi = \lambda\phi$ then $\mathcal{D}(X\phi) = (\lambda + \mu)X\phi$, so that $X\phi$ is also an eigenfunction of \mathcal{D} with eigenvalue $\lambda + \mu$, provided that $X\phi$ is not identically zero.. In the second-order filter case, we find pairs of these operators, X_1^\pm, X_2^\pm where X_i^\pm increments by $\pm\mu_i$. Thus, the eigenvalues of (1.10) form a bi-infinite family, given by $\{\lambda_0 + i\mu_1 + j\mu_2\}_{i,j \in \mathbb{Z}}$, for some $\lambda_0 \in \mathbb{R}$. In the case of the first-order filter, one of the μ_i is zero, so that the family is indexed only by one copy of \mathbb{Z} . The details of constructing these operators is the subject matter of Section 2. In general, an n -th order system will have n pairs of such operators, where if one increments by μ the other in the pair increments by $-\mu$.

The entire stability analysis for (1.7) can be done in terms of solutions to (1.11). If we define the operators $L_i = \partial_{s_i}$, for $0 \leq i \leq n$ and $L_i = s_i$ for $n+1 \leq i \leq 2n$ and $L_{2n+1} = 1$, we can write each solution X_k of (1.11) as $X_k = \sum_{i=1}^{2n+1} X_i^k L_i$, and (1.11) becomes an eigenvalue problem of the form $\mathbf{S} \mathbf{A} \mathbf{x}^k = \mu_k \mathbf{x}^k$, where \mathbf{S} is symmetric and \mathbf{A} is antisymmetric. From this, we can show that X_k^+ commutes with all the other solutions to (1.11), except for its paired operators, X_k^- , where the commutator is a constant. This, in turn allows us to write

$$\mathcal{D} = \sum_k v_k X_k^- X_k^+ + \lambda_0 \quad (1.12)$$

and to determine the base eigenvalue. That is we can find λ_0 , which will be the largest eigenvalue of \mathcal{D} , thereby determining the stability of solutions to (1.7).

2. Ladder Operators. We are interested in constructing ladder operators for \mathcal{D} . That is, we seek operators X satisfying the commutation relation

$$[\mathcal{D}, X] = \mu X. \quad (2.1)$$

To find such an X , we observe that the differential operators ∂_s, ∂_u , as well as multiplication by s or u , all have nice commutation relations with \mathcal{D} :

$$\begin{aligned} [\mathcal{D}, \partial_s] &= -\kappa \partial_s + \beta \partial_u - 2a\varepsilon \\ [\mathcal{D}, s] &= \sigma^2 \partial_s + \kappa s - \nu u \\ [\mathcal{D}, \partial_u] &= -\alpha \partial_u + w \partial_s - 2b\varepsilon \\ [\mathcal{D}, u] &= \rho^2 \partial_u + \alpha u - \beta s. \end{aligned} \quad (2.2)$$

Except for the constants that appear in the relations for $[\mathcal{D}, \partial_s]$ and $[\mathcal{D}, \partial_u]$, the commutator of \mathcal{D} with each operator can be written in terms of the other operators. Hence, if we include constants, (with $[\mathcal{D}, 1] = 0$) we can build X as a linear combination of the operators. We write

$$X = c_1 \partial_s + c_2 s + c_3 \partial_u + c_4 u + c_5 \quad (2.3)$$

and require the c_j to satisfy (2.1). This becomes a simple matrix eigenvalue problem $\mathbf{T} \mathbf{c} = \mu \mathbf{c}$, in detail

$$\begin{pmatrix} -\kappa & \sigma^2 & \nu & 0 & 0 \\ 0 & \kappa & 0 & -\beta & 0 \\ \beta & 0 & -\alpha & \rho^2 & 0 \\ 0 & -\nu & 0 & \alpha & 0 \\ -2a\varepsilon & 0 & -2b\varepsilon & 0 & 0 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{pmatrix} = \mu \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{pmatrix} \quad (2.4)$$

The eigenvalues of \mathbf{T} are

$$\mu \in \left\{ 0, \pm \frac{1}{2} \left(\alpha + \kappa \pm \sqrt{(\alpha - \kappa)^2 + 4\beta\nu} \right) \right\} \quad (2.5)$$

2.1. First Order Filter. In the particular cases discussed in the introduction, these simplify greatly. In the first order case, we have $\alpha = \nu = \beta = \rho = b = 0$ and $a = 1$. This is the case of a first-order filter, and \mathcal{D} has the form: $\mathcal{D}\phi = \frac{\sigma^2}{2}\partial_s^2\phi + \kappa\partial_s(s\phi) + (-2\gamma + 2\varepsilon s)\phi$. The stability of the solutions to (1.7) are completely characterized by the following theorem.

THEOREM 2.1. *In the first-order filter case, the eigenvalues of \mathcal{D} are given by*

$$\lambda_n = -n\alpha - 2\gamma + 2\frac{\varepsilon^2\sigma^2}{\kappa^2}, \quad n \geq 0.$$

The eigenfunctions of \mathcal{D} and are given by $\phi_n = (X^-)^n \phi_0(s)$ where

$$\phi_0(s) = e^{-\kappa s^2/\sigma^2 + 2\varepsilon s/\kappa}, \quad \text{and} \quad X^- = \kappa\partial_s + 2\varepsilon.$$

Hence, the eigenfunctions are products of exponentials with the Hermite polynomials, which are complete in L^2 . Note that the result of Theorem 1.1 follows immediately from the result of Theorem 2.1. If the eigenvalues are indeed given by $\lambda_n = -n\alpha - 2\gamma + 2\frac{\varepsilon^2\sigma^2}{\kappa^2}$, where $n \geq 0$, then the largest eigenvalue is $\lambda_0 = -2\gamma + 2\frac{\varepsilon^2\sigma^2}{\kappa^2}$, and the stability of the solutions to (1.7) is decided precisely by the sign of this eigenvalue. Hence, the stability barrier is $\lambda_0 = 0$, for which $\varepsilon = \frac{\kappa\sqrt{\gamma}}{|\sigma|}$, which is the condition in Theorem 1.1.

Theorem 2.1 depends on the following three lemmas.

LEMMA 2.2. *If ϕ is an eigenfunction of \mathcal{D} with eigenvalue λ , then $\phi^\pm = X^\pm\phi$ is either the zero function or is an eigenfunction of \mathcal{D} with eigenvalue $\lambda \pm \kappa$, where the operators X^\pm are defined by*

$$X^+ = \kappa\partial_s + 2\frac{\kappa^2}{\sigma^2}s - 2\varepsilon, \quad X^- = \kappa\partial_s + 2\varepsilon.$$

Proof. For $\alpha = \nu = \beta = \rho = b = 0$ and $a = 1$, the nontrivial eigenvalues and their eigenvectors of (2.4) become

$$\mu_1 = \kappa, \quad v_1 = \begin{pmatrix} \kappa \\ 2\frac{\kappa^2}{\sigma^2} \\ 0 \\ 0 \\ -2\varepsilon \end{pmatrix}, \quad \mu_2 = -\kappa, \quad v_1 = \begin{pmatrix} \kappa \\ 0 \\ 0 \\ 0 \\ 2\varepsilon \end{pmatrix}.$$

This means that the differential operators X^\pm satisfy $[\mathcal{D}, X^\pm] = \pm\kappa X^\pm$. Hence, $\mathcal{D}X^\pm\phi = X^\pm\mathcal{D}\phi \pm \kappa X^\pm\phi = (\lambda \pm \kappa)X^\pm\phi$. Hence, if $X^\pm\phi$ is not identically zero, then it is a new eigenfunction for \mathcal{D} . \square

LEMMA 2.3. *The eigenvalues of \mathcal{D} are bounded above by $\lambda_0 = -2\gamma + 2\frac{\varepsilon^2\sigma^2}{\kappa^2}$*

Proof. This follows from two simple relationships between X^+ and X^- . Let $L^2(w)$ be the weighted Hilbert space of equivalence classes of measurable functions such that $\int_{\mathbb{R}} \phi(x)\psi(x)w(x)dx < \infty$. Then, it is a simple calculus exercise to verify that if $\psi, \phi \in L^2(w)$, for $w(s) = e^{\kappa s^2/\sigma^2}$, then

$$\int_{\mathbb{R}} (X^+\phi)\psi e^{\kappa s^2/\sigma^2} ds = - \int_{\mathbb{R}} (X^-\psi)\phi e^{\kappa s^2/\sigma^2} ds. \quad (2.6)$$

It is also easy to see that

$$\frac{\sigma^2}{2\kappa} X^+ X^- \phi = \mathcal{D}\phi + (2\gamma - 2\frac{\varepsilon^2\sigma^2}{\kappa^2})\phi. \quad (2.7)$$

Combining these facts, we see that if ϕ is an eigenfunction of \mathcal{D} with eigenvalue λ , the

$$\begin{aligned}
 (\lambda + 2\gamma - 2\frac{\varepsilon^2\sigma^2}{\kappa^2})\|\phi\|_{L^2(w)}^2 &= \int_{\mathbb{R}} (\lambda + 2\gamma - 2\frac{\varepsilon^2\sigma^2}{\kappa^2})\phi^2 e^{\kappa s^2/\sigma^2} ds \\
 &= \int_{\mathbb{R}} (\mathcal{D}\phi)\phi + (2\gamma - 2\frac{\varepsilon^2\sigma^2}{\kappa^2})\phi^2 e^{\kappa s^2/\sigma^2} ds \\
 &= \int_{\mathbb{R}} \frac{\sigma^2}{2\kappa} (X^+ X^- \phi) \phi e^{\kappa s^2/\sigma^2} ds \\
 &= - \int_{\mathbb{R}} \frac{\sigma^2}{2\kappa} (X^- \phi) (X^- \phi) e^{\kappa s^2/\sigma^2} ds \\
 &= -\frac{\sigma^2}{2\kappa} \|X^- \phi\|_{L^2(w)}^2 \leq 0.
 \end{aligned}$$

Hence, we must have $\lambda \leq -2\gamma + 2\frac{\varepsilon^2\sigma^2}{\kappa^2}$. \square

LEMMA 2.4. *The function $\phi_0(s) = e^{-\kappa s^2/\sigma^2 + 2\varepsilon s/\kappa}$ solves $X^+ \phi_0 = 0$, and is an eigenfunction of \mathcal{D} with eigenvalue*

$$\lambda_0 = -2\gamma + 2\frac{\varepsilon^2\sigma^2}{\kappa^2}, \quad (2.8)$$

which is the largest eigenvalue of \mathcal{D} .

Proof. The previous two lemmas imply that we cannot generate new eigenfunctions of \mathcal{D} indefinitely by applying X^+ to an eigenfunction. That is at some point, we must have $X^+ \phi \equiv 0$, otherwise we could endlessly generate new eigenfunctions with eigenvalues increasing to infinity, contradicting Lemma 2.3. It is easy to see that ϕ_0 is the solution to $X^+ \phi = 0$, and that $\mathcal{D}\phi_0 = \lambda_0\phi_0$. The other eigenfunctions of \mathcal{D} can be written as $X^- \phi_0$ because not only are these eigenfunctions of \mathcal{D} as shown in Lemma 2.2, but they generate the Hermite polynomials, which are complete. This completes the proof of Lemma 2.4 as well as the proof of Theorem 2.1. \square

2.2. Second Order Filter. For the second-order filter, with $\nu = \rho = 0, a = \alpha, b = -\beta$, the operator \mathcal{D} is

$$\mathcal{D}\phi = \frac{\sigma^2}{2}\partial_s^2\phi + \kappa\partial_s(s\phi) + \partial_u((\alpha u - \beta s)\phi) + (-2\gamma + 2\varepsilon(\alpha s - \beta u))\phi.$$

This case is similar to the first order filter, though some details are more complicated. We will sketch the main ideas, many of which generalize filters of arbitrary degree.

The result is the theorem:

THEOREM 2.5. *With $\nu = \rho = 0, a = \alpha, b = -\beta$ the eigenvalues of \mathcal{D} are*

$$\lambda_{n,j} = -n\alpha - j\kappa - 2\gamma + 2\frac{(\alpha^2 - \beta^2)^2\sigma^2\varepsilon^2}{\alpha^2\kappa^2}, \quad n, j \geq 0.$$

The eigenfunctions of \mathcal{D} are given by $\phi_{n,j}(s, u) = (X_\alpha^-)^n (X_\kappa^-)^j \phi_{0,0}(s, u)$, where $X_\alpha^- = \partial_u - \frac{2\beta}{\alpha}\varepsilon$, $X_\kappa^- = \partial_s + \frac{\beta}{\alpha-\kappa}\partial_u - \frac{2(\beta^2+\alpha\kappa-\alpha^2)}{\kappa(\alpha-\kappa)}\varepsilon$, and

$$\begin{aligned}
 \phi_{0,0}(s, u) = \exp \left(-\frac{\alpha + \kappa}{\sigma^2} s^2 - \frac{\alpha(\alpha + \kappa)^2}{\beta^2\sigma^2} u^2 + \frac{2\alpha(\alpha + \kappa)}{\beta\sigma^2} su \right. \\
 \left. - \frac{2(\alpha^2 - \beta^2)\varepsilon}{\alpha\kappa} s - \frac{2\varepsilon(2\alpha^3 - 2\alpha\beta^2 + 2\alpha^2\kappa - \beta^2\kappa)}{\alpha\beta\kappa} u \right). \quad (2.9)
 \end{aligned}$$

The form of $\phi_{0,0}$ and $\phi_{n,j}$ indicate that the Hermite polynomials will again be the building blocks of the eigenfunctions. However, in this case it is not as simple as exponentials times Hermite polynomials, because there is a mixing of the variables s and u .

As mentioned above, this setting is considerably more complicated than the first-order filter. In particular, the analogue of Lemma 2.3 is not available directly. However, the practical techniques for finding the eigenvalues and eigenfunctions are the same. In fact, many of the interesting features of the previous section can be seen simply by studying the eigenvalue problem (2.4) in more detail. It is in this framework that we investigate the second-order case.

To begin, by solving equation (2.4), we find the four operators:

$$\begin{aligned} X_\alpha^+ &= \partial_s + \frac{\alpha + \kappa}{\sigma^2} s + \frac{\beta}{2\alpha} \partial_u + \frac{\beta^2 - 2\alpha^2}{\alpha^2} \varepsilon, \\ X_\alpha^- &= \partial_u - \frac{2\beta}{\alpha} \varepsilon, \\ X_\kappa^+ &= \partial_s + \frac{\kappa}{2\sigma^2} s + \frac{\beta}{\alpha + \kappa} \partial_u + \frac{2(\beta^2 - \alpha\kappa - \alpha^2)}{k(\alpha + \kappa)} \varepsilon, \\ X_\kappa^- &= \partial_s + \frac{\beta}{\alpha - \kappa} \partial_u - \frac{2(\beta^2 + \alpha\kappa - \alpha^2)}{\kappa(\alpha - \kappa)} \varepsilon. \end{aligned}$$

where each X satisfies the relation $[\mathcal{D}, X] = cX$, where $c = \pm\alpha, \pm\kappa$. Hence, each X generates a new eigenfunction from an old one, and increments the eigenvalue by $\pm\alpha, \pm\kappa$, provided the function is not identically zero. The reasoning is exactly the same as for Lemma 2.2.

Furthermore, the largest eigenvalue is

$$\lambda_{0,0} = -2\gamma + 2 \frac{(\alpha^2 - \beta^2)^2 \sigma^2 \varepsilon^2}{\alpha^2 \kappa^2}, \quad (2.10)$$

so we must have that at some point $X_\alpha^+ \phi = 0$ and $X_\kappa^+ \phi = 0$, for the same function ϕ . Otherwise, by applying the raising operators repeatedly to an eigenfunction, we could generate an eigenfunction with eigenvalue exceeding $\lambda_{0,0}$. The expression (2.10) for $\lambda_{0,0}$ gives the range of ε stated in Theorem 1.2, for which the equation is stable.

Once we have this, the analogous result for Lemma 2.4 is obtained by simultaneously solving the first order PDEs

$$X_\alpha^+ \phi = 0, \quad X_\kappa^+ \phi = 0. \quad (2.11)$$

The result is the formula for $\phi_{0,0}$ given in Theorem 2.5. These results have been tested numerically, as described in Section 3.

2.3. General Techniques. The main tool in all of this analysis is the ladder operators. These are written as a linear combination of the basic operators $s, u, \partial_s, \partial_u, 1$ in the case of the second-order filters. We introduce the notation $L_1 = \partial_s, L_2 = s, L_3 = \partial_u, L_4 = u, L_5 = 1$. In a general setting where the variables s, u are replaced by y_1, y_2, \dots, y_n , we would have $2n + 1$ operators $L_1 = \partial_{y_1}, L_2 = y_1, \dots, L_{2n} = y_n, L_{2n+1} = 1$. A ladder operator would be $X = \sum_j x_j L_j$ satisfying $[\mathcal{D}, X] = \mu X$.

This equation for X and μ is the same as (2.4) from Section 2. There are several nice properties of this equation. First, it is easy to see that the matrix \mathbf{A} , with components $a_{i,j} = [L_i, L_j]$ is antisymmetric. In particular, \mathbf{A} is a $2n + 1 \times 2n + 1$ matrix whose last row and last

column are all zeros, and first $2n \times 2n$ block has a block-diagonal form

$$\mathbf{A} = \begin{pmatrix} J & 0 & 0 & \dots & 0 \\ 0 & J & 0 & \dots & 0 \\ 0 & 0 & J & \dots & 0 \\ \vdots & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad J = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}. \quad (2.12)$$

It is also not too difficult to see that \mathcal{D} must be a linear combination of $L_i L_j$. That is $\mathcal{D} = \sum_{i,j} c_{i,j} L_i L_j$. This representation of \mathcal{D} is not unique, and in fact we can arrange it so that the matrix \mathbf{C} , with components $c_{i,j}$, is symmetric.

LEMMA 2.6. *The equation $[\mathcal{D}, X] = \mu X$ can be written as a matrix eigenvalue problem $\mathbf{T}x = \mu x$ where \mathbf{T} is an $2n + 1 \times 2n + 1$ matrix, which can be written as $\mathbf{T} = \mathbf{S}A$ where \mathbf{S} is symmetric and A is antisymmetric.*

Proof. We compute an expression for $[\mathcal{D}, X]$ in terms of \mathbf{C} and \mathbf{A} .

$$\begin{aligned} [\mathcal{D}, X] &= \sum_{i,j,m} c_{i,j} x_m (L_i L_j L_m - L_m L_i L_j) \\ &= \sum_{i,j,m} c_{i,j} x_m (L_i [L_j, L_m] + [L_i, L_m] L_j) \\ &= \sum_{i,j,m} c_{i,j} x_m (L_i a_{j,m} + a_{i,m} L_j) \\ &= \sum_{i,m} ((\mathbf{C} + \mathbf{C}^\top)A)_{i,m} x_m L_i. \end{aligned}$$

For the equation $[\mathcal{D}, X] = \mu X$, this implies that we have

$$\sum_{i,m} ((\mathbf{C} + \mathbf{C}^\top)A)_{i,m} x_m L_i = \mu \sum_i x_i L_i.$$

In matrix notation, this is just $(\mathbf{C} + \mathbf{C}^\top)A\mathbf{x} = \mu\mathbf{x}$. Note that this is the same as equation (2.4) in the case of a second-order filter, but also applies to higher degree filters. We define $\mathbf{S} = \mathbf{C} + \mathbf{C}^\top$, which is obviously symmetric, even if \mathbf{C} is not. Hence, $\mathbf{T} = \mathbf{S}A$. \square

LEMMA 2.7. *If X is a ladder operator of \mathcal{D} , with $[\mathcal{D}, X] = \mu X$, then there is another ladder operator Y such that $[\mathcal{D}, Y] = -\mu Y$.*

Proof. Because \mathbf{T} is the product of a symmetric matrix and an antisymmetric matrix, its eigenvalues must come in pairs that are negatives of each other. Thus, if $\mathbf{T}x = \mu x$, then the coefficients of x define a ladder operator for \mathcal{D} with increment μ . But Since $\mathbf{T} = \mathbf{S}A$, there is another eigenvector of \mathbf{T} such that $\mathbf{T}y = -\mu y$, and elements of y define a ladder operator for \mathcal{D} with increment $-\mu$. Because \mathbf{T} has an odd number of dimensions, it will always have a zero eigenvalue. In this case $x = y$, and the ladder operator increments by 0. In fact, the ladder operator is just the identity. \square

We can also show that, just as in the proof of Lemma 2.3, the operator \mathcal{D} can be written as a linear combination of the ladder operators.

LEMMA 2.8. *There exist computable coefficients v_j such that*

$$\mathcal{D} = \sum_{j=-n}^n v_j X_j^+ X_j^-.$$

Proof. Without loss of generality, we suppose \mathbf{C} is symmetric, so that $\mathbf{S} = 2\mathbf{C}$. Let $x^{\pm k}$ be an eigenvectors of \mathbf{T} with eigenvalues $\mu_{\pm k}$, where $\mu_k = -\mu_{-k}$. We can always do this by the Lemma 2.7 And let $y^{\pm k}$ be the eigenvectors of \mathbf{T}^T with eigenvalues $\mu_{\pm k}$. So we have $\mathbf{S}\mathbf{A}x^k = \mu_k x^k$. The transpose of this is $-\mathbf{A}\mathbf{S}y^k = \mu_k y^k$, or $\mathbf{A}\mathbf{S}y^k = -\mu_k y^k$. If we left-multiply by \mathbf{S} , we get $\mathbf{T}(\mathbf{S}y^k) = \mathbf{S}\mathbf{A}(\mathbf{S}y^k) = -\mu_k(\mathbf{S}y^k) = \mu_{-k}(\mathbf{S}y^k)$. Hence, we have see that $\mathbf{S}y^k$ is an eigenvector of \mathbf{T} with eigenvalue μ_{-k} , so it is a multiple of x^{-k} . Thus, we have the relationship between the eigenvectors of \mathbf{T} and those of \mathbf{T}^T . We define v_k as the real number such that $\mathbf{S}y^k = 2v_k x^{-k}$, where y_k is normalized so that $y_k \cdot x_k = 1$. Therefore, $\mathbf{C}y^k = v_k x^{-k}$. With this normalization, we have $\delta_{i,j} = \sum_k x_i^k x_j^k$, so that

$$c_{i,j} = \sum_p c_{i,p} \delta_{p,j} = \sum_{k,p} x_i^k c_{j,p} y_p^k = \sum_k x_i^k v_k x_j^{-k}.$$

But we have $\mathcal{D} = \sum_{i,j} c_{i,j} L_i L_j$, so now

$$\mathcal{D} = \sum_{i,j,k} x_i^k v_k x_j^{-k} L_i L_j = \sum_k v_k \sum_i x_i^k L_i \sum_j x_j^{-k} L_j = \sum_k v_k X_k^+ X_k^-$$

□ We can also show that $[X_j^+, X_i^-] = 0$, unless $i = j$, in which case the commutator is just a multiple of the identity. With this, we can rewrite \mathcal{D} to be a sum of $X_k^- X_k^+$ for $k \geq 0$, plus the extra constants we pick up by switching X_k^+ and X_k^- . Since there is an eigenfunction ϕ for which each $X_k^+ \phi = 0$, we then have that $\mathcal{D}\phi = \sum_{k \geq 0} v_k X_k^- X_k^+ \phi + \lambda \phi = \lambda \phi$. Thus, we have an effective way to compute the largest eigenvalue of \mathcal{D} .

3. Numerics. To test the stability of (1.7) numerically, one could integrate $x(t)$ for some long time interval $[0, T]$ using a stochastic integrator, and see if the solution grows. However, this is clearly inefficient and slow if one takes large T with small time-steps. We have done this, just for a reference to test a different technique described below. First, we give the details of the numerical integration we used.

We will use the more standard notation here for stochastic differential equations and for Brownian motions, W_t . For an SDE of the form

$$dY_t = a(Y)dt + b(Y)dW_t \quad (3.1)$$

one can construct a second-weak-order scheme simply by writing down the Itô-Taylor expansion for Y_t and taking the first several terms. In doing so the scheme is

$$\begin{aligned} Y_{n+1} = & Y_n + a_n h + b_n \Delta W_h + \frac{1}{2} b_n b'_n (\Delta W_h^2 - h) \\ & + a'_n b_n V_n + \frac{1}{2} \left(a_n a'_n + \frac{1}{2} a''_n b_n^2 \right) h^2 \\ & + \left(a_n b'_n + \frac{1}{2} b''_n b_n^2 \right) (\Delta W_h h - V_n) \end{aligned} \quad (3.2)$$

where h is the time-step, $a_n = a(Y_n)$, etc., and V_n is a process generated such that the covariance of ΔW_h and V_n is

$$\langle \Delta W_h, v_n \rangle = \begin{pmatrix} h & \frac{1}{2} h^2 \\ \frac{1}{2} h^2 & \frac{1}{3} h^3 \end{pmatrix}$$

This scheme is discussed in detail in [4], [2].

From now on, we will assume we are in the case of $\nu = \rho = 0$ and $a = \alpha, b = -\beta$. We use (3.2) to integrate $s(t)$ from (1.7) on some interval $[0, T]$. Then the equation for u is just $\dot{u} = -\alpha u + \beta s(t)$ where s is known on $[0, T]$. The solution to this equation is just

$$u(t) = e^{-\alpha t} u_0 + \beta e^{-\alpha t} \int_0^t e^{\alpha \tau} s(\tau) d\tau.$$

Using a trapezoid method to calculate the integral, we get a second-order accurate representation of u on $[0, T]$. Now with s and u both known we can integrate $\dot{x} = -\gamma x + \varepsilon(\alpha s(t) - \beta u(t))x$ to second order with a Crank-Nicolson method

$$x_{n+1} = \frac{1 + \frac{h}{2}(-\gamma + \varepsilon(\alpha s_n - \beta u_n))}{1 - \frac{h}{2}(-\gamma + \varepsilon(\alpha s_{n+1} - \beta u_{n+1}))} x_n$$

The alternative method is derived as follows. Write the PDE $\partial_t M = \mathcal{D}M$ and take moments in u . That is, multiply by u^n and integrate with respect to du so that the only variable left is s . Now, this will give an infinite system of ODEs for $M_n(s, t) = \int_{\mathbb{R}} u^n M(s, u, t) du = \int u^n x^2 P(s, u, x, t) dx du$. This gives the system

$$\begin{aligned} \partial_t M_n = & \frac{\sigma^2}{2} \partial_s^2 M_n + \kappa \partial_s (s M_n) - n\alpha M_n + n\beta s M_{n-1} \\ & + 2\varepsilon\alpha s M_n - 2\varepsilon\beta M_{n+1} - 2\gamma M_n \end{aligned} \quad (3.3)$$

for $n \geq 0$. We seek $M_n(s, t) = e^{\lambda t} w_n(s)$, and expand the w_n in terms of products of $e^{-\kappa s^2}$ and Hermite polynomials. We will use the notation $\phi_j(s) = e^{-\kappa s^2} H_j(\sqrt{\kappa}s)$, where H_j is the j -th Hermite polynomial. Then $w_n(s) = \sum_{j=0}^{\infty} a_j^n \phi_j(s)$ and one can easily check that $\frac{\sigma^2}{2} \partial_s^2 \phi_j + \kappa \partial_s (s \phi_j) = -j\kappa \phi_j$. Also, the Hermite polynomials satisfy $H_{j+1}(x) - 2xH_j + 2jH_{j-1}(x) = 0$, so that we can write $s\phi_j(s) = \frac{1}{2\sqrt{\kappa}}(\phi_{j+1} + 2j\phi_{j-1})$.

Thus (3.3) can be written as

$$\begin{aligned} \lambda \sum_j a_j^n \phi_j = & \sum_j (-j\kappa - n\alpha) a_j^n \phi_j + \frac{n\beta}{2\sqrt{\kappa}} a_j^{n-1} (\phi_{j+1} + 2j\phi_{j-1}) \\ & + \varepsilon \frac{\alpha}{\sqrt{\kappa}} a_j^n (\phi_{j+1} + 2j\phi_{j-1}) + 2\varepsilon\beta a_j^{n+1} \phi_j - 2\gamma a_j^n \phi_j. \end{aligned} \quad (3.4)$$

The equation for the coefficients of ϕ_j becomes

$$\lambda a_j^n = -(j\kappa + n\alpha + 2\gamma) a_j^n + \frac{n\beta}{2\sqrt{\kappa}} (a_{j-1}^{n-1} + 2(j+1)a_{j+1}^{n-1}) \quad (3.5)$$

$$+ \varepsilon \frac{\alpha}{\sqrt{\kappa}} (a_{j-1}^n + 2(j+1)a_{j+1}^n) + 2\varepsilon\beta a_j^{n+1}. \quad (3.6)$$

If we write truncate the system to $0 \leq n \leq N_1$ and $0 \leq j \leq N_2$, then we can approximate (3.5) by an $N_1 N_2 \times N_1 N_2$ matrix eigenvalue problem, $\lambda \mathbf{w} = \mathbf{Q} \mathbf{w}$. The matrix \mathbf{Q} will be block tri-diagonal, and \mathbf{w} is a column vector with components $\mathbf{w} = (\mathbf{a}^1, \mathbf{a}^2, \dots, \mathbf{a}^{N_1})$, where $\mathbf{a}^n = (a_1^n, a_2^n, \dots, a_{N_2}^n)$. This can be solved very quickly and converges for even small values of N_1 and N_2 .

We can also solve the initial value problem $\dot{\mathbf{w}} = \mathbf{Q} \mathbf{w}$. In particular, if we write $\mathbf{w}(t) = \sum_m c_m \Phi_m e^{\lambda_m t}$ where Φ_m are the eigenvectors of \mathbf{Q} , then we can find the long-time behavior of $M_{xx}(t)$ by looking at the first component of \mathbf{w} . To see this, recall that $M_n(s, t) =$

$\int_{\mathbb{R}^2} x^2 u^n P(x, u, s, t) dx du = \sum_j a_j^n \phi_j(s)$ and that we are interested in the second moment in x , which is $M_{xx}(t) = \int_{\mathbb{R}^3} x^2 P(x, u, s, t) dx du ds$. So if we have an expression for $M_0(s, t)$, we can integrate it $M_{xx}(t) = \int_{\mathbb{R}} M_0(s, t) ds$. Because the Hermite polynomials $H_j(\sqrt{\kappa}s)$ are orthogonal with respect to the weight $e^{-\kappa s^2}$ and $H_0 = 1$ is constant, we have $\int_{\mathbb{R}} \phi_j(s) ds = \sqrt{\frac{\pi}{\kappa}}$ for $j = 0$ and vanishes for $j > 0$. Therefore, once we have solved $\dot{\mathbf{w}} = \mathbf{Q}\mathbf{w}$, the function $\sqrt{\frac{\pi}{\kappa}}\mathbf{w}_0(t)$ will be a good approximation to M_{xx} . In practice, we can even truncate the system further once we know the eigenvalues. Since eigenvalues λ_m with negative real part will play little role for large times (and it is the large time limit we are interested in) we can simply approximate $\mathbf{w}_0(t)$ by the terms in the series corresponding to the eigenvalues with the largest real part. Two examples are shown in Figures 3.1(a) and 3.1(b) for different values of the parameter ε . The solid lines are the results of the stochastic integrator plotted against $t \in [0, T]$ with $T = 10$, and the dotted lines are the results of the truncated initial value problem against t . The critical value of ε for those parameter values is $\varepsilon = 0.0165\dots$, so in fact the solution is

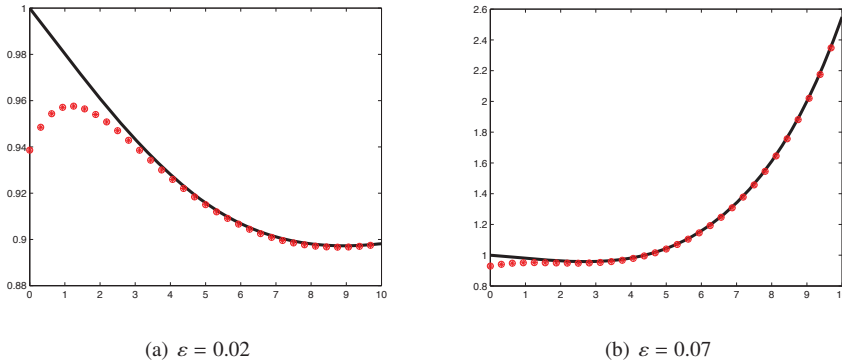


FIG. 3.1. The horizontal axis is time, $t \in [0, 10]$. The parameter values on both plots are $\alpha = 0.3, \kappa = 0.5, \beta = 1, \gamma = 0.01, \sigma = 1$

unstable in both cases. This is easily seen in Figure 3.1(b), but is not apparent in Figure 3.1(a). If the analytic expression for $\lambda_{n,j}$ from Theorem 2.5 were unknown to us, then we might not know that both cases are unstable. To see the instability using the stochastic integrator, one would have to integrate for a much larger value of T . However, when solving the initial value problem, we find the largest eigenvalues of \mathbf{Q} and hence for \mathcal{D} , which in the case $\varepsilon = 0.02$ is $\lambda \approx 0.0094$

4. Conclusions and Future Work. We have developed a method for determining the stability of (1.7) that generalizes to higher-order filters. The stability boundary, defined as the solution to $\lambda_0 = 0$, where λ_0 is the largest eigenvalue of \mathcal{D} has an analytic expression. In the case of first and second-order filters, this is achieved by solving (2.8) and (2.10) for ε . The numerical methods that we have described generalize naturally to higher-order differential equations. In particular, they are applicable to the Mathieu equation (1.1). We have used them to determine the stability of (1.1), and found they match the results from perturbation theory. We plan to investigate whether the ladder operators exist for higher-order differential equations, such as (1.1). If there are ladder operators in this case, it is not clear if they completely determine the stability as in the case of the first-order equation.

REFERENCES

- [1] L. ARNOLD, *Stochastic differential equations as dynamical systems*, in Realization and modelling in system theory (Amsterdam, 1989), vol. 3 of Progr. Systems Control Theory, Birkhäuser Boston, Boston, MA, 1990, pp. 489–495.
- [2] S. ASMUSSEN AND P. W. GLYNN, *Stochastic simulation: algorithms and analysis*, vol. 57 of Stochastic Modelling and Applied Probability, Springer, New York, 2007.
- [3] P. A. M. DIRAC, *The Principles of Quantum Mechanics*, Oxford, at the Clarendon Press, 1947. 3d ed.
- [4] P. E. KLOEDEN AND E. PLATEN, *Numerical solution of stochastic differential equations*, vol. 23 of Applications of Mathematics (New York), Springer-Verlag, Berlin, 1992.
- [5] H. LAMB, *Hydrodynamics*, Cambridge Mathematical Library, Cambridge University Press, Cambridge, sixth ed., 1993. With a foreword by R. A. Caflisch [Russel E. Caflisch].
- [6] N. G. VAN KAMPEN, *Stochastic processes in physics and chemistry*, vol. 888 of Lecture Notes in Mathematics, North-Holland Publishing Co., Amsterdam, 1981.

COMPARISON OF SENSITIVITY ANALYSIS METHODS FOR NUCLEAR REACTOR NEUTRONICS

W. CYRUS PROCTOR*, BRIAN M. ADAMS†, CRISTIAN RABITI‡, AND HANY S. ABDEL-KHALIK§

Abstract. This preliminary study compares adjoint-based local sensitivity analysis to global sensitivity methods on a simplified nuclear reactor neutronics model. Sensitivity analysis methods manipulate computational models to investigate how variations in input parameters affect system responses of interest. For linear models involving large numbers of parameters and few responses, adjoint methods efficiently yield accurate local sensitivities. As nonlinearities are introduced, the associated computational burden grows, at best, in proportion to the number of input parameters to estimate all cross-derivative terms. For strongly nonlinear models, global sensitivity analysis is more appropriate for estimating parameter importance, as it can account for nonlinear interactions and response variations over the whole admissible parameter range, not solely at nominal values. However, as the number of parameters increases, global methods become computationally infeasible. This work exercises local and global methods on a neutron diffusion simulation of a sodium-cooled fast reactor, considered in a linear regime with a modest number of input parameters. It compares the relative rankings and strength of influence resulting from both approaches. Future work will develop hybrid methods to simultaneously address the curse of dimensionality and nonlinearities, and apply them to a simulation of nonlinear reactor phenomena with many input parameters.

1. Introduction. Advances in computational systems and simulations enable ever-more efficient and complex calculations for nuclear reactor phenomena such as neutronics, thermal-hydraulics, structures, or plant dynamics. Of particular interest to the nuclear engineering community is reactor core simulation in novel operating regimes [7]. Such simulations require specification of many governing parameters describing physics, chemistry, or the environment/operating conditions. It is crucial to understand how system responses are influenced by input data as experiments to improve these data, reducing uncertainty, can be exceptionally expensive and must be carefully selected and designed [2]. Sensitivity analysis of model outputs with respect to input parameters critically informs the selection process for experimental data collection and can make uncertainty quantification more computationally feasible by propagating only the most crucial factors.

Nuclear reactor core neutronics simulations (the target of the present work) rely primarily on microscopic cross-section data to estimate the rates of interaction between neutrons and various reactor materials. Cross-section data, which are experimentally evaluated, are complicated functions of neutron energy, reaction type, and reactor materials. To characterize them in raw form would lead to impractical computational cost. Therefore reactor physicists appeal to homogenization techniques to reduce the complexity of cross-section data [12]. Our work employs a simplified core model with cross-section data describing interaction rates for 45 isotopes, each with 5 reaction types, each with 33 energy groups (a group is an energy range over which cross-sections are assumed constant). Uncertainties in these cross-section data are a major source of uncertainty for reactor physics simulation [6] and therefore are the focus of this study.

Historically for neutronics, sensitivity analysis is a common first step in the identification of input data uncertainty contributions to response uncertainty. In particular, adjoint-based methods for uncertainty propagation are commonly used for steady state reactor physics models because in steady state, key performance metrics of interest to design and operation (such as k -eigenvalue, peak power, or reactivity coefficients) behave almost linearly within the range of interest for cross-sections variations. Since analyses typically involve only a

*North Carolina State University Nuclear Engineering, wprocto@ncsu.edu

†Sandia National Laboratories, briadam@sandia.gov

‡Idaho National Laboratory, cristian.rabiti@inl.gov

§North Carolina State University Nuclear Engineering, abdelkhalik@ncsu.edu

handful of integral response metrics and orders of magnitude more cross section input variables, adjoint approaches compute the necessary derivatives efficiently, using only marginally more computation than a nominal forward calculation.

In this special linear case, measured experimental variance-covariance information of model input parameters is readily combined with sensitivity information to propagate moments to obtain confidence bounds on responses of interest. In this method, only the first and second moments of input probability density functions are propagated through the model to estimate the first and second moments of the responses [9]. The accuracy of this method depends on the validity of the linearity assumption used in extrapolating the function value and gradient data to the entire range of input parameter variations considered. For cross-sections, the range of variation is set by the accuracy of the experimental procedure employed to measure cross-section data. For steady-state neutronics calculations, it has been reported that models' deviation from linear behavior could be ignored, thereby allowing the use of adjoint-based methods for sensitivity analysis and uncertainty quantification [6]. As neutronics models are tightly coupled with other physics however, such as thermal hydraulics, and fuel performance codes, the linearity assumption breaks down, and other methods for sensitivity analysis must be explored.

This manuscript numerically compares various sample-based sensitivity techniques available in Sandia National Laboratories' DAKOTA software to the derivative-based adjoint approach common in nuclear engineering applications. It will provide a brief computational model overview followed by general overviews of the sensitivity techniques considered. Results in the form of relative sensitivity coefficients are presented for comparison. Additional insights from simple and partial rank correlations as well as Sobol indices are discussed.

2. Reactor Neutronics Simulation. The sodium-cooled fast reactor model of interest in this work is simulated with a FORTRAN 90 code developed specifically for this comparison. It utilizes point-wise Gauss-Seidel with successive over relaxation to solve a multi-group neutron diffusion equation in both forward and adjoint modes. Consider the forward form of equations, [3] for energy groups $g = 1, \dots, G$

$$\begin{aligned}
 -\nabla \cdot D_1 \nabla \phi_1 + \Sigma_{R_1} \phi_1 &= \frac{\chi_1}{k} \sum_{g'=1}^G \nu_{g'} \Sigma_{f g'} \phi_{g'} + \sum_{g'=1}^{g-1} \Sigma_{s, g'} \longrightarrow_g \phi_{g'} + \sum_{g'=g+1}^G \Sigma_{s, g'} \longrightarrow_g \phi_{g'} \\
 -\nabla \cdot D_2 \nabla \phi_2 + \Sigma_{R_2} \phi_2 &= \frac{\chi_2}{k} \sum_{g'=1}^G \nu_{g'} \Sigma_{f g'} \phi_{g'} + \sum_{g'=1}^{g-1} \Sigma_{s, g'} \longrightarrow_g \phi_{g'} + \sum_{g'=g+1}^G \Sigma_{s, g'} \longrightarrow_g \phi_{g'} \\
 &\vdots \\
 -\nabla \cdot D_G \nabla \phi_G + \Sigma_{R_G} \phi_G &= \frac{\chi_G}{k} \sum_{g'=1}^G \nu_{g'} \Sigma_{f g'} \phi_{g'} + \sum_{g'=1}^{G-1} \Sigma_{s, g'} \longrightarrow_g \phi_{g'} + \sum_{g'=G+1}^G \Sigma_{s, g'} \longrightarrow_g \phi_{g'}
 \end{aligned} \tag{2.1}$$

where the spatial dependence \bar{r} has been suppressed, and the physical parameters are defined in Fig. 2.1.

The removal cross section contains total and within group scatter terms, while the right-hand side would include the fission, downscatter and upscatter terms. k is called the multiplication factor which balances the system of equations such that the number of neutrons produced equals the number of neutrons lost. A multiplication factor equal to one implies that the associated reactor system is in steady state, i.e. capable of self-sustained production of neutrons.

D_g	diffusion coefficient [cm]
ϕ_g	neutron flux [$cm^{-2}sec^{-1}$]
Σ_{r_g}	macroscopic removal cross section [cm^{-1}]
χ_g	fission neutrons yield [-]
k	multiplication factor (inverse of largest eigenvalue) [-]
ν_g	average number of neutrons created per fission [-]
Σ_{fg}	macroscopic fission cross section [cm^{-1}]
$\Sigma_{s,g' \rightarrow g}$	macroscopic group to group scattering cross section [cm^{-1}].

FIG. 2.1. Physical parameters for the neutron diffusion equations.

Specifically for cylindrical (r, z) geometry we may write for energy group g

$$-\left(\frac{1}{r} \frac{\partial}{\partial r} r D^g(r, z) \frac{\partial}{\partial r} + \frac{\partial}{\partial z} D^g(r, z) \frac{\partial}{\partial z}\right) \phi^g(r, z) + \left(\Sigma_t(r, z) - \Sigma_{s,g \rightarrow g}(r, z)\right) \phi^g(r, z) = S^g(r, z), \quad (2.2)$$

where

$$S^g(r, z) = \sum_{g'=1}^{g-1} \Sigma_{sg' \rightarrow g}(r, z) \phi^{g'}(r, z) + \sum_{g'=g+1}^G \Sigma_{sg' \rightarrow g}(r, z) \phi^{g'}(r, z) + \frac{\chi^g(r, z)}{k} \sum_{g'=1}^G \nu^{g'}(r, z) \Sigma_f^{g'}(r, z) \phi^{g'}(r, z).$$

Equation (2.2) was discretized using a Finite-Volume scheme, and the resulting system of equations may be written matrix form as:

$$L\phi = \frac{1}{k}F\phi,$$

and solved via r -line ordering starting from $r = 0$ to $r = R$ and $z = 0$ to $z = H$. Thus, the corresponding eigenvalue system may be transformed into the iterative solution of G fixed-source matrix systems via a power iteration technique to obtain the dominant eigenvalue λ and corresponding eigenvector ϕ [3].

This eigenvalue system may be written in terms of a broader general framework designed to encompass not only eigensystems but also fixed-source problems:

$$A\phi = Q,$$

where $A = L - \frac{1}{k}F$ and $Q = 0$ in this case. Utilizing the variational method of [8], we may consider the system as constraints on the output response R . These constraints may be incorporated with the response using Lagrange multipliers to create an augmented functional T :

$$T = R - \langle Z(A\phi - Q) \rangle,$$

where Z is the Lagrange multiplier. A change in some input parameter α results in $T \rightarrow T'$. We may expand T' via first order Taylor series expansion about the reference state to obtain

$$T' = T + \left\langle \frac{\partial T}{\partial Z} \Delta Z + \frac{\partial T}{\partial \phi} \Delta \phi + \frac{\partial T}{\partial \alpha} \Delta \alpha \right\rangle.$$

The perturbation in the response may be rewritten as

$$\Delta R \cong \left\langle \frac{\partial T}{\partial \alpha} \Delta \alpha \right\rangle + \left\langle \frac{\partial T}{\partial Z} \Delta Z \right\rangle + \left\langle \frac{\partial T}{\partial \phi} \Delta \phi \right\rangle.$$

If we wish T to be stationary to changes in the Lagrange multiplier we must set $A\phi - Q = 0$, which, for the case of our eigensystem, is true. If we wish T to be stationary to changes in the neutron flux we must choose Z to be the adjoint neutron flux ($Z = \phi^*$) which satisfies the equation

$$A^* \phi^* = \frac{\partial R}{\partial \phi}.$$

Thus, the perturbation of a general response R due to a change in parameter α is given as

$$\Delta R \cong \left\langle \frac{\partial T}{\partial \alpha} \Delta \alpha \right\rangle = \left\langle \left(\frac{\partial R}{\partial \alpha} - \phi^* \frac{\partial A}{\partial \alpha} \phi + \phi^* \frac{\partial Q}{\partial \alpha} \right) \Delta \alpha \right\rangle. \quad (2.3)$$

The adjoint flux physically denotes the importance of neutrons in the phase space to the response of interest. $\frac{\partial R}{\partial \phi}$ is determined based on the response, implying that the adjoint equation must be solved independently for every response of interest. This places a limitation on the use of adjoint methods for problems involving many responses. In this paper we focus only on the multiplication factor as a response, for which $\frac{\partial R}{\partial \phi} = 0$. Carrying out a first-order approach, an estimate of the change in the multiplication factor ($R = k$) due to a change in the cross-section data σ is given by:

$$\frac{\partial k}{\partial \sigma} = -k^2 \frac{\left\langle \phi^*, \left(\frac{\partial L}{\partial \sigma} - \frac{1}{k} \frac{\partial F}{\partial \sigma} \right) \phi \right\rangle}{\left\langle \phi^*, F \phi \right\rangle}. \quad (2.4)$$

This result may be compared directly to a finite-difference approach for derivative approximation. Equation(2.4) may also serve as the basis for a relative sensitivity coefficient between the output response k and the input parameters σ

$$S(k, \sigma) = \frac{\frac{\partial k}{k}}{\frac{\partial \sigma}{\sigma}} = -k\sigma \frac{\left\langle \phi^*, \left(\frac{\partial L}{\partial \sigma} - \frac{1}{k} \frac{\partial F}{\partial \sigma} \right) \phi \right\rangle}{\left\langle \phi^*, F \phi \right\rangle}. \quad (2.5)$$

3. Global Sensitivity Analysis. In contrast to local sensitivity, the goal of global sensitivity analysis is to assess the influence of input parameters, considered over their whole possible range, on output responses. Such an approach is typically used to rank the importance of the input factors, determine the effect of their variance on the variance of the output, or assess whether higher-order interactions between parameters affect output responses [10]. Global sensitivity and uncertainty analysis methods may offer additional problem insight when response linearity as a function of input variables is violated. Sampling-based approaches to sensitivity and uncertainty analysis, such as Latin hypercube sampling (LHS), can be very robust even in the presence of strong nonlinearity, but can be computationally expensive for screening studies, where ($10 \times$ number of input variables) evaluations of the model are typically used. An advantage of sampling, however, is that it can be applied without modifying the

solver (simulator); this favorable characteristic is also true for other “black-box” approaches to sensitivity and uncertainty analysis, such as design and analysis of computer experiments (DACE), reliability analysis, and stochastic expansion methods such as polynomial chaos and stochastic collocation.

Global sensitivity analysis methods typically identify an ensemble of well distributed points in the input variable space, evaluate the computational model at these points, and perform statistical analysis of the resulting function values (and possibly derivatives, if available). Sandia’s DAKOTA (Design Analysis Kit for Optimization and Terascale Applications) includes numerous algorithms for sensitivity analysis and uncertainty quantification, including those briefly described here (for further details, consult the DAKOTA User’s Manual [1]). DAKOTA provides a flexible, extensible interface to any analysis code, permitting its ready use for global sensitivity analysis of a fast reactor simulated with a neutronics model.

3.1. Latin hypercube sampling and correlation. Latin hypercube sampling (LHS) is among the most robust, ubiquitous, and accepted global sampling and analysis techniques, which include other sampling methods such as standard Monte Carlo, quasi-Monte Carlo, orthogonal arrays, and jittered sampling. It relies on a probabilistic characterization of input uncertainties (cross section deviations in the present context), from which realizations of the input variables are generated for model evaluation, and then statistical analysis on the corresponding response values can be performed. LHS typically resolves statistics with fewer samples than standard Monte Carlo and can generate sample designs respecting input variable correlation structure [15]. DAKOTA reports the mean, standard deviation, and coefficient of variation of each response (together with confidence intervals based on the number of samples used) and correlation coefficients (both on the data and on their ranks). For example, a simple (Pearson) correlation between output y and input x is given by

$$\rho_{x,y} = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}}, \quad (3.1)$$

whereas partial correlation coefficients adjust for the effects of other variables. The results presented here focus on the simple and partial correlation coefficients, which are scaled between -1 and 1 . Larger absolute magnitudes indicate stronger linear relationship between the input and output (see [1]). Additional statistical techniques (such as regression analysis) can also be used to analyze the parameter/response pairs resulting from an LHS study [13].

3.2. Variance-based decomposition. Variance-based decomposition (VBD) summarizes how model output variability can be attributed to variability in individual input variables. This relationship is captured in a main effect sensitivity index

$$S_i = \frac{\text{Var}_{x_i} [E(Y|x_i)]}{\text{Var}(Y)}, \quad (3.2)$$

which reflects the fraction of output uncertainty attributable to x_i alone, and the total effect index

$$T_i = \frac{E [\text{Var}(Y|x_{-i})]}{\text{Var}(Y)} = \frac{\text{Var}(Y) - \text{Var}(E[Y|x_{-i}])}{\text{Var}(Y)}, \quad (3.3)$$

where x_{-i} indicates variable i is omitted from the vector of input variables, which accounts for variability due to x_i and its interactions with other input variables. Larger values of these Sobol indices indicate a stronger influence of an input on variance of the output. The sum of main effect indices is less than or equal to 1 (and equal to 1 for a linear model), whereas the total effect indices need not be. See [10] and [11] for further information.

For d input parameters, VBD requires the evaluation of d -dimensional integrals and, when implemented with replicates in sampling, typically requires $d + 2$ replicates of N LHS samples. As this can be prohibitively expensive, even for tens of variables, the sensitivity indices are often calculated based on a surrogate model or polynomial chaos approximation.

3.3. Surrogate model acceleration. Another use of LHS sample designs is to construct global surrogate models, also referred to as response surfaces or meta-models. For instance, a modest number of evaluations (typically on the order of 2–10 times the number of input variables) of the computational model can be used to train a Kriging (Gaussian process), MARS, or artificial neural network model [5]. This surrogate model is comparatively inexpensive to evaluate and can be sampled tens or hundreds of thousands of times to calculate correlation coefficients or Sobol sensitivity indices.

Polynomial chaos expansions (PCE) globally approximate the output y as a function of input random variables x :

$$y(x) \approx \sum_{j=0}^P \alpha_j \psi_j(x), \quad (3.4)$$

where orthogonal polynomials ψ_j are selected to yield optimal convergence of the approximation [4]. Specifically, they are chosen to be orthogonal with respect to the probability distribution of the inputs x with the same support, e.g., Hermite polynomials are used with normal random variables whereas a Legendre basis is optimal for uniform. The coefficients α_j can be calculated with spectral projection and multi-dimensional integration or regression. Here, sparse-grid quadrature and cubature integration techniques for PCE are considered.

Once constructed, a PCE can again be exhaustively sampled, but often statistics of interest can be calculated analytically using the structure of the approximation. Sudret [14] demonstrated that Sobol sensitivity indices can be calculated directly from a PCE, and that approach as implemented in DAKOTA [16] is used here.

4. Reactor Core Model. The model employed in this study is a cylindrical, axially symmetric, two-region, and spatially homogeneous approximation of the Sodium-cooled Fast Reactor. Approximations in geometry and compositions are necessary to reduce the size of cross-section data used to describe the reactor core (the total number of cross-section data is 7425, which is reduced from 380952). The fuel region is capped axially by two neutron reflectors to reduce neutron leakage and hence reduce the core size necessary for a self-sustained reaction. Figure 4.1 illustrates the geometry and boundary conditions used. See reference [7] for fuel and reflector materials composition. Overall dimensions were set at $R = 100$ cm, $h = 75$ cm and $H = 100$ cm. A radial and axial mesh of 30 by 40 nodes is used throughout the comparison. This spacing allowed for adequate resolution of the neutron flux at the fuel interfaces. The cross section data are based on an ERANOS model for a Sodium-cooled Fast Reactor also discussed in [7].

5. Comparison of global and local sensitivity approaches.

5.1. Analysis approaches used. Several DAKOTA SA methods as well as the adjoint-based sensitivity coefficients output from the computational model were compared in terms of importance rankings and relative sensitivities. The DAKOTA methods compared were Latin hypercube sampling (LHS) with or without variance-based decomposition (VBD) or surrogates (Gaussian process or Kriging models), and polynomial chaos expansions (PCE) directly yielding Sobol indices and local sensitivity estimates. The Gaussian process model used has a quadratic trend function, while the Kriging model has a constant trend. While exhaustive numerical experiments were conducted varying sampling size and method type,

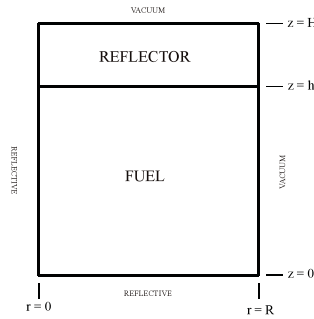


FIG. 4.1. Cross section of cylindrical two-region layout of reactor.

TABLE 5.1
Abbreviations for DAKOTA methods and options.

DAKOTA Method	Shorthand	# model evals
LHS, 180 samples	LHS 180	180
LHS, 1800 samples	LHS 1800	1800
VBD, 180 samples, 20 replicates	VBD 180	3600
PCE, cubature level 3	C3	36
PCE, cubature level 5	C5	649
PCE, sparse grid level 2	SG2	721
PCE sparse grid level 3	SG3	9841
Kriging, 180 true samples	KRIG 180	180
+ 1000 surrogate samples		
Gaussian process, 180 true samples	GP 180	180
+ 1000 surrogate samples		

only a few are considered for brevity. Table 5.1 lists the DAKOTA methods/options presented here.

Global sensitivity methods in DAKOTA are tractable for tens of variables. The relatively large input space (7425 parameters) defined for this comparison required choosing groups of parameters and assigning one control for DAKOTA to manipulate. First, the eighteen fissioning isotopes out of the total forty-five isotopes in the reactor were selected as the focus of this comparison. Then, the microscopic fission cross sections of each of these isotopes were bundled together into eighteen independent meta-parameters. In other words, for each of 18 parameters DAKOTA perturbed, a total of thirty-three microscopic cross section values for that specific isotope were perturbed by the same multiplier.

A perturbation of thirty-three cross sections in ensemble compares well with the output of the adjoint-based method embedded in the model due to the predominantly linear behavior of the system. The sensitivity coefficient for a group of cross sections is simply the sum of the individual cross section sensitivity coefficients. As an illustrative example, consider the isotope Plutonium-239 (PU239). The DAKOTA control parameter will perturb all thirty-three energy groups of the fission cross section of PU239. As a result, a relative sensitivity coefficient for response y_k with respect to this set of cross sections can be designated as

$S(y_k, \sigma_{\text{PU239,FISSION,ALL}})$. This sensitivity coefficient may be computed with

$$S(y_k, \sigma_{\text{PU239,FISSION,ALL}}) = \sum_{g=1}^G S(y_k, \sigma_{\text{PU239,FISSION,g}}).$$

5.2. Conversion of DAKOTA Results. Outputs from DAKOTA and the neutronics model may be compared both qualitatively via rankings of relative importances and also quantitatively via relative sensitivity coefficients introduced in equation (2.5). To compare the global statistical approaches of DAKOTA to the local sensitivities of the adjoint form of the model, the simple correlation, ρ_{x_j, y_k} , between the k^{th} output response, y_k , and the j^{th} input parameter, x_j , output by DAKOTA must be transformed to a metric comparable to a local gradient. DAKOTA's LHS based methods calculate Pearson correlations in the form of Equation (3.1). These are based on a regression in which the model is assumed linear. For the neutronics model, the input that DAKOTA provides to the model during sampling is a multiplier on the nominal value of the input parameter \tilde{x}_j (relative variation) rather than the absolute parameter value. In other words,

$$x_j = c_j \tilde{x}_j,$$

the perturbed input parameter x_j is derived from multiplying the nominal input parameter value \tilde{x}_j by a multiplier c_j which has been chosen as $c_j \sim N(1, 0.03)$ to represent roughly $\pm 10\%$ variation. It follows that,

$$\frac{\partial y_k}{\partial c_j} = \frac{\partial y_k}{\partial x_j} \frac{\partial x_j}{\partial c_j} = \frac{\partial y_k}{\partial x_j} \tilde{x}_j.$$

As the number of samples increases, the sample means converge to their nominal values and the relative sensitivities may be compared as follows:

$$\frac{\partial y_k}{\partial x_j} \frac{\tilde{x}_j}{\bar{y}_k} \quad \text{to} \quad \rho_{c_j, y_k} = \frac{\sqrt{\sum_{i=1}^N (y_{ik} - \bar{y}_k)^2}}{\sqrt{\sum_{i=1}^N (c_{ij} - \bar{c}_j)^2}} \frac{1}{\bar{y}_k}, \quad (5.1)$$

where the terms in the square roots are the standard deviation of output k over standard deviation of input multiplier j . Similarly, all PCE based methods return the absolute local sensitivities (based on the global polynomial approximation's derivative) directly and need only to be scaled by the inverse of nominal output \bar{y}_k . This is possible since the nominal values of the c_j multipliers are one.

Relative importances may be measured by simple correlations, partial correlations, Sobol main effect indices or total Sobol indices. Based on the absolute values associated with these metrics, a larger value implies stronger correlation and, hence, a stronger sensitivity. Various DAKOTA methods may be compared based solely on the rankings of these metrics.

5.3. Sensitivity analysis results. Results here focus on sensitivity of the multiplication factor k . In the discussion, the result from the neutronics model is treated as a reference result because it provides an exact (to within numerical precision) estimate of local derivatives. The most common application of global SA is to rank the importance of input parameters, however, it should also offer reasonable predictions of local first-order sensitivities for this predominantly linear parameter to response mapping. Table 5.2 shows rankings based on

TABLE 5.2

Comparison of isotope importance rankings as determined by various methods. LHS, Kriging and Gaussian Process methods show simple correlations, whereas PCE/C3 and VBD show Sobol index-based rankings. Anomalies are indicated in blue italics. All methods omitted from results produced the same rankings as the local adjoint and LHS 1800 results.

ADJOINT (relative)	LHS 180 (correl)	KRIG 180 (correl)	GP 180 (correl)	C3 (Sobol)	VBD 180 (Sobol)	LHS 1800 (correl)
PU239	PU239	PU239	PU239	PU239	PU239	PU239
PU240	<i>PU241</i>	PU240	PU240	PU240	<i>PU238</i>	PU240
PU241	<i>PU240</i>	PU241	PU241	PU241	<i>PU242</i>	PU241
AM242M	AM242M	AM242M	AM241M	AM242M	AM242M	AM242M
PU238	PU238	PU238	PU238	PU238	<i>AM243</i>	PU238
PU242	PU242	PU242	PU242	PU242	<i>CM245</i>	PU242
U238	<i>CM245</i>	U238	U238	U238	<i>NP237</i>	U238
CM245	<i>U238</i>	CM245	CM245	CM245	<i>CM246</i>	CM245
AM241	<i>CM244</i>	AM241	<i>CM244</i>	AM241	<i>CM243</i>	AM241
CM244	<i>AM241</i>	CM244	<i>AM241</i>	CM244	<i>U236</i>	CM244
NP237	<i>AM243</i>	NP237	NP237	NP237	<i>CM242</i>	NP237
AM243	<i>U234</i>	AM243	AM243	AM243	<i>CM244</i>	AM243
U235	<i>NP237</i>	U235	U235	U235	<i>U234</i>	U235
CM242	<i>U236</i>	CM242	CM242	CM242	<i>U235</i>	CM242
CM246	<i>CM242</i>	CM246	CM246	CM246	<i>AM241</i>	CM246
U234	<i>CM246</i>	<i>U236</i>	<i>CM243</i>	U234	<i>U238</i>	U234
CM243	CM243	CM243	<i>U236</i>	CM243	<i>PU241</i>	CM243
U236	<i>U235</i>	<i>U234</i>	<i>U234</i>	U236	<i>PU240</i>	U236

relative sensitivity coefficient, simple correlation, or Sobol index S_i . Most methods, except LHS 180 and VBD 180, which can probably be considered under-resolved, agree on the rankings. Those that are under-resolved still get approximate ordering correct. Taking 180 LHS samples and building a Kriging model helps with the ranking, as does directly using 1800 LHS samples.

The isotope importances given in Table 5.2 agree reasonably on physical grounds based on the initial nuclide concentration given in [7]. The loading for this reactor is predominantly Plutonium-239 and Uranium-238. These two isotopes and their neutron capture daughter isotopes, e.g. Plutonium-240 or Plutonium-241, are most crucial for reactor criticality.

Figure 5.1 offers a comparison of the relative sensitivity coefficients and the comparable DAKOTA-based measures derived in Section 5.2. There is significant discrepancy between the adjoint-based approach and LHS 180, but as the number of samples increases to 1800 the differences nearly vanish. All other methods (some requiring far fewer computational model evaluations), also capture these relative sensitivities. When 180 LHS samples are used to construct a Kriging model (KRIG 180), results comparable to the LHS 1800 case are achieved.

A somewhat arbitrary, but often useful, screening heuristic is that correlations greater than 0.2 in magnitude can generally be considered relevant and those greater than 0.5 significant. The results for simple correlations shown in Figure 5.2 indicate significant linear relationship between a number of isotopes and response k . The partial correlation coefficients shown in Figure 5.3 illustrate that controlling for the effects of other input factors may reveal other significant factors, indeed all but a handful are likely significant. The sample size has little influence on these metrics, reflecting that for a linear analysis, few samples are needed

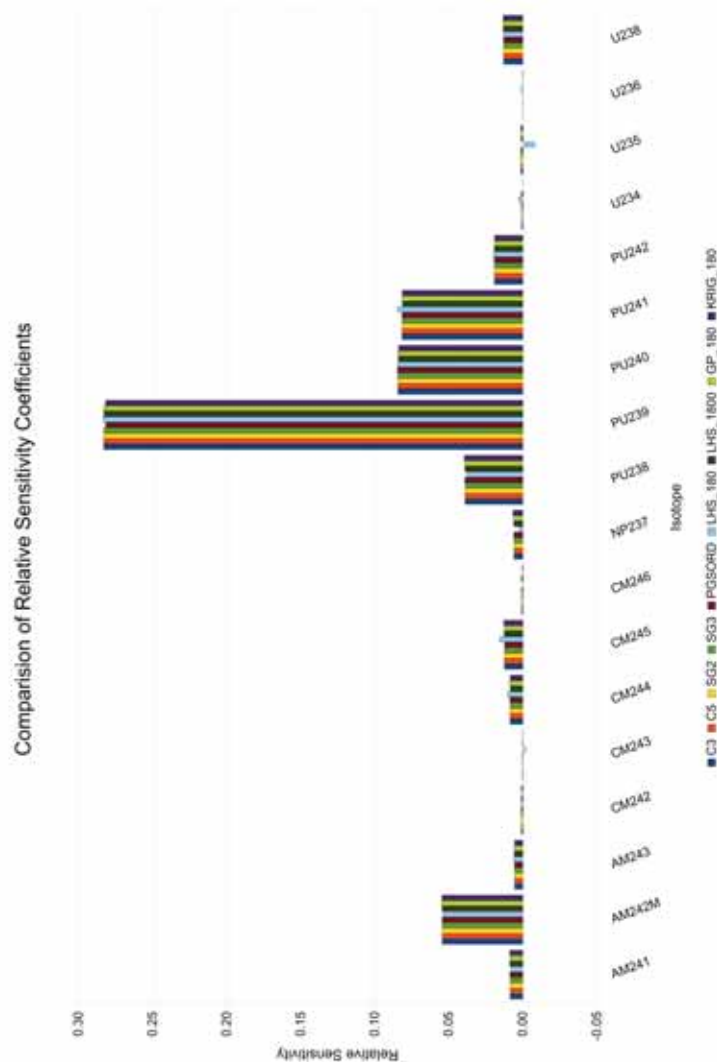


FIG. 5.1. Relative sensitivities (RS) estimated by local derivatives compared to scaled global estimates from DAKOTA methods.

to capture the effects.

Sobol indices shown in Table 5.3 offer another measure of variable importance. They indicate a dominant effect of Pu239, and marginal effects of Pu240, Pu241, and Am242M.

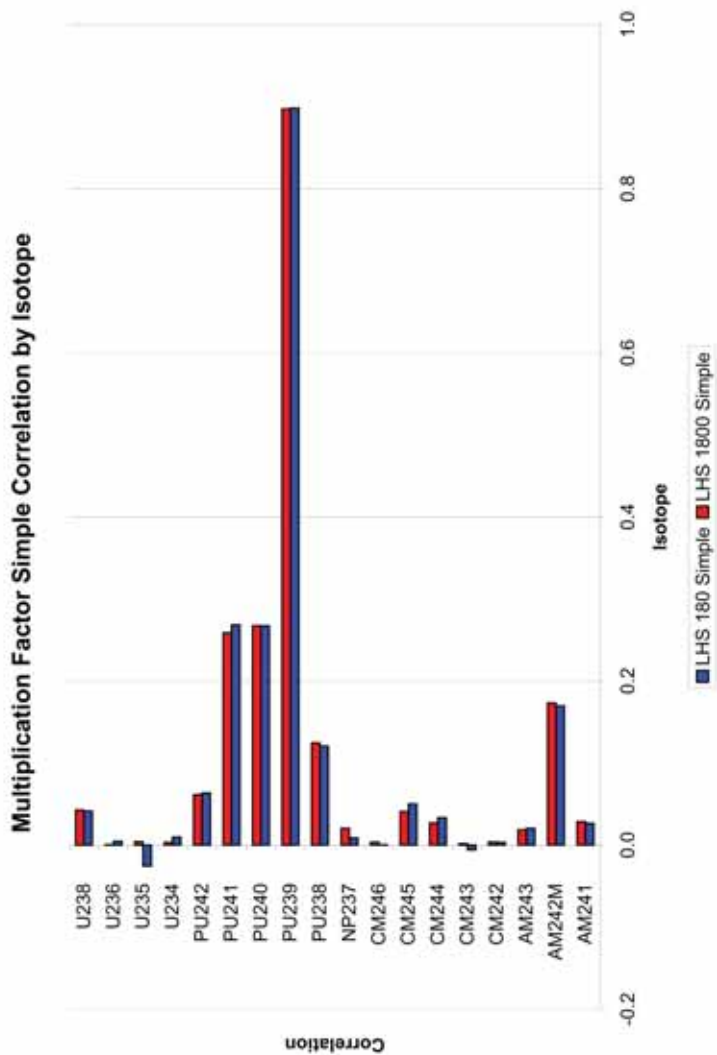


FIG. 5.2. Simple correlation coefficients calculated using 180 and 1800 LHS samples.

6. Conclusions. This exploratory comparison of DAKOTA and adjoint methods for a linear neutronics model will serve as the foundation for future development of hybrid global/local-adjoint methods to combat the curses of nonlinearity and dimensionality. Reassuringly, the comparisons of the qualitative rankings and quantitative sensitivity coefficients between local adjoint and DAKOTA methods employed for this study converge as the number of function evaluations increase. In as few as 36 function evaluations for PCE-based methods,

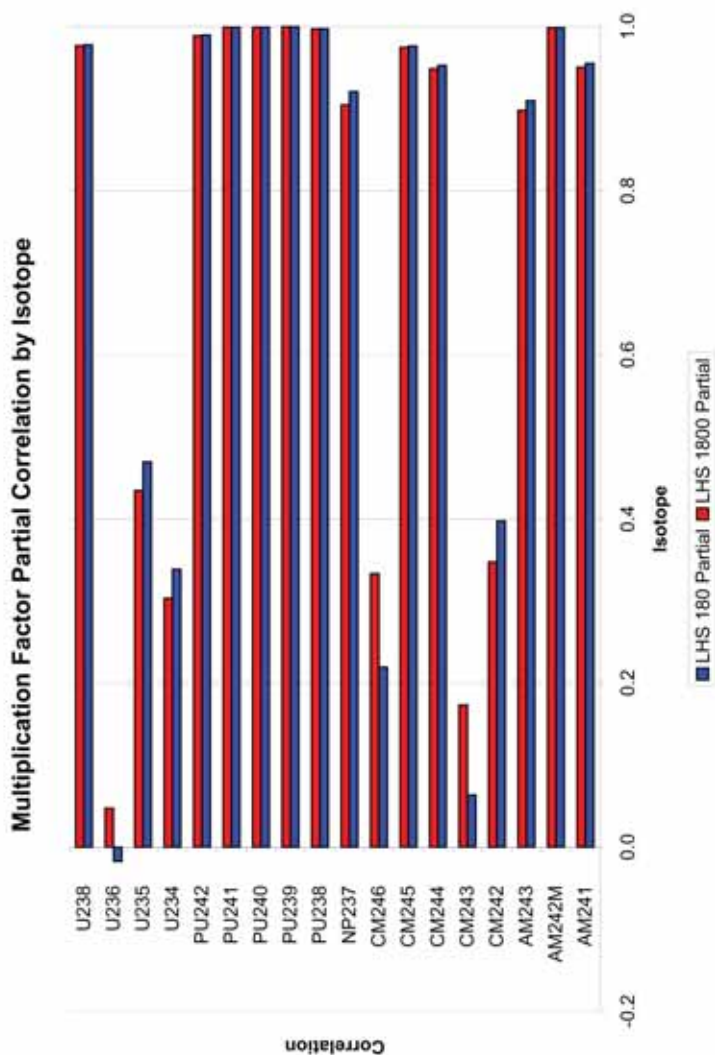


Fig. 5.3. Partial correlation coefficients calculated using 180 and 1800 LHS samples.

the Sobol indices and sensitivity coefficients were appropriately captured as compared with the adjoint-based approach used in the neutronics model. Similarly, the surrogate-enhanced Kriging model successfully captured almost all rankings and sensitivity coefficients within 1.3% with 180 function evaluations. These methods show promise in capturing main effects while they are known to be less likely to misinform in the presence of nonlinearity, see [11], pp.10–12, 16–33, and references therein, though with a significant increase in computational

TABLE 5.3

Sobol total effects T_i and main effects S_i for each isotope as calculated with PCE, cubature level 5 (all other stochastic expansion methods produced identical results).

Isotope	T_i	S_i
AM241	0.001	0.001
AM242M	0.030	0.030
AM243	0.000	0.000
CM242	0.000	0.000
CM243	0.000	0.000
CM244	0.001	0.001
CM245	0.002	0.002
CM246	0.000	0.000
NP237	0.000	0.000
PU238	0.015	0.015
PU239	0.806	0.806
PU240	0.072	0.072
PU241	0.067	0.067
PU242	0.004	0.004
U234	0.000	0.000
U235	0.000	0.000
U236	0.000	0.000
U238	0.002	0.002

cost (10x to 100x due to required forward runs of the model). However, their scalability in number of parameters is limited, so ongoing research is investigating the use of adjoint-generated derivative information in global sensitivity approaches.

Acknowledgments. This work was supported by the DOE Nuclear Energy Advanced Modeling and Simulation (NEAMS) program, and performed in large part at Sandia National Laboratories (SNL) and Idaho National Laboratory (INL). Internships facilitated by Dr. Cristian Rabiti of INL and Dr. James Stewart of SNL made this work possible. Dr. Laura Swiler of SNL provided valuable insight on the mechanics of global sensitivity methods; and Drs. William Rider and James Kamm of SNL gave excellent feedback on the manuscript.

REFERENCES

- [1] B. M. ADAMS, W. J. BOHNOFF, ET AL., *DAKOTA: A multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: Version 5.0 users manual*, Tech. Rep. SAND2010-2183, Sandia National Laboratories, December 2009.
- [2] M. CHADWICK, P. OBLOZINSKI, M. HERMAN, N. GREENE, R. MCKNIGHT, D. SMITH, P. YOUNG, R. MACFARLANE, G. HALE, S. FRANKLE, ET AL., *ENDF/B-VII. 0: Next generation evaluated nuclear data library for nuclear science and technology*, Nuclear Data Sheets, 107 (2006), pp. 2931–3060.
- [3] J. DUDERSTADT AND L. HAMILTON, *Nuclear reactor analysis*, John Wiley & Sons New York, 1976.
- [4] M. S. ELDRED, C. G. WEBSTER, AND P. CONSTANTINE, *Evaluation of non-intrusive approaches for wiener-asky generalized polynomial chaos*, in Proceedings of the 10th AIAA Non-Deterministic Approaches Conference, no. AIAA-2008-1892, Schaumburg, IL, April 7–10 2008.
- [5] A. A. GIUNTA, J. M. MCFARLAND, L. P. SWILER, AND M. S. ELDRED, *The promise and peril of uncertainty quantification using response surface approximations*, Structure and Infrastructure Engineering, 2 (2006), pp. 175–189.
- [6] M. JESSEE, H. ABDEL-KHALIK, AND P. TURINSKY, *Evaluation of BWR Core Attribute Uncertainties due to Multi-group Cross-Section Uncertainties*, Proc. of M&C, (2007).

- [7] C. RABITI, G. PALMIOTTI, M. ASSAWAROONGRUENGCHOT, M. SALVATORES, H. S. ABDEL-KHALIK, W. C. PROCTOR, R. NOURGALIEV, B. M. ADAMS, AND L. P. SWILER, *NEAMS VU: Report on sensitivity techniques selection, FY2009*, tech. rep., Idaho National Laboratories, September 2009.
- [8] Y. RONEN, *CRC handbook of nuclear reactors calculations*, vol. 3, CRC, 1986.
- [9] ———, *Uncertainty analysis*, Franklin Book Company, Incorporated, 1988.
- [10] A. SALTELLI, *Sensitivity analysis in practice: a guide to assessing scientific models*, John Wiley & Sons, Inc., 2004.
- [11] A. SALTELLI, K. CHAN, AND E. SCOTT, eds., *Sensitivity analysis*, John Wiley & Sons, Inc., 2000.
- [12] K. SMITH, *Assembly homogenization techniques for light water reactor analysis*, *Progress in Nuclear Energy*, 17 (1986), pp. 303–335.
- [13] C. STORLIE, L. SWILER, J. HELTON, AND C. SALLABERRY, *Implementation and evaluation of nonparametric regression procedures for sensitivity analysis of computationally demanding models*, *Reliability Engineering and System Safety*, 94 (2009), pp. 1735–1763.
- [14] B. SUDRET, *Global sensitivity analysis using polynomial chaos expansions*, *Reliability Engineering & System Safety*, 93 (2008), pp. 964–979.
- [15] L. P. SWILER AND G. D. WYSS, *A user's guide to Sandia's latin hypercube sampling software: LHS UNIX library and standalone version*, Tech. Rep. SAND04-2439, Sandia National Laboratories, Albuquerque, NM, July 2004.
- [16] G. TANG, G. IACCARINO, AND M. ELDRED, *Global sensitivity analysis for stochastic collocation expansion*, in *Proceedings of the 51st AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference (12th AIAA Non-Deterministic Approaches Conference)*, no. AIAA-2010-2922, Orlando, FL, April 2010.

Meshing and Optimization

The articles in this section have an over arching theme of optimization. The first three articles use optimization technology to improve the quality of meshes used in finite element analysis. The remaining articles focus on practical implementation of various optimization strategies.

Voshell et al. describe a mesh untangling algorithm using the target matrix paradigm for quadratic elements. The authors present results using quadratic triangles, quadrilaterals, tetrahedrons and hexahedron that show the algorithm is beneficial. *Franks and Knupp* present a new metric for quadratic 2D mesh untangling. Using the target matrix paradigm the authors present numerical results that indicate comparable performance to previously developed metrics. *Park et al.* consider the effect different vertex orderings have on the performance of a local mesh smoothing algorithm. Timings varied by a factor of 2, with no clearly optimal strategy. However several strategies were seen as appropriate for an all-purpose ordering. *Berwald et al.* summarize two projects. The first explores using tools from computational topology to estimate the local density of points in an attractor. The second describes the extension of the TEVA-SPOT software suite. *Steele and Watson* describe an implementation of the Benders decomposition for large scale linear programming in Pyomo. *Hunter et al.* consider the use of stochastic optimization for energy economy optimization models. *Orsini and Gray* discuss an ACRO implementation of the EAGLS algorithm for hybrid optimization.

E.C. Cyr
S.S. Collis

December 17, 2010

QUADRATIC ELEMENT MESH UNTANGLING AND SHAPE OPTIMIZATION VIA THE TARGET-MATRIX PARADIGM

NICHOLAS VOSHELL*, PATRICK KNUPP†, AND JASON KRAFTCHECK‡

Abstract. When meshes containing high-order elements are inverted, it is often due to the projection of high-order nodes onto the domain boundary. Addressing the inversion requires either re-meshing or post-processing the mesh. The Target-matrix paradigm (TXP) for mesh optimization provides a framework we use to develop a node-movement post-processing method that addresses meshes containing inverted quadratic elements. Our proposed algorithm includes two optimization stages; one stage uses a non-barrier size quality metric to untangle the mesh, and the second stage uses a barrier shape quality metric to improve shape quality while preventing the mesh from inverting. The method is tested on non-hybrid meshes with four element types: triangles, quadrilaterals, tetrahedra, and hexahedra. Overall, the algorithm proved to be beneficial, although there is no guarantee that the optimized mesh will be untangled.

1. Introduction. Quadratic finite elements are sometimes used to achieve high accuracy simulations. Such meshes are usually created by first creating elements with straight sides and then curving boundary sides by projecting mid-side and mid-face nodes onto the bounding geometry [15]. This approach creates boundary-conforming, quadratic element meshes on complex geometries in which the resulting boundary elements can be poor or even inverted, especially if elements are large compared to the associated geometry curvature. The poor quality elements can impact simulation accuracy, reduce efficiency and, if the Jacobian is negative, even prevent proceeding with calculations.

Finite element methods are based on a local map from a logical element to a given mesh element. The element is usually considered inverted if and only if there exists some point p in the logical element such that $\det(J) \leq 0$, where J is the Jacobian matrix of the map. This test is relatively simple for linear elements because the determinant is linear in the logical coordinates. Quadratic and other higher-order elements are more difficult to determine if they are inverted. Convergence theory for finite elements requires that the mesh contain no inverted elements.

There have been limited studies into untangling or improving element shape qualities within meshes that include high-order elements. In [17, 18], the authors derive the regions that mid-side nodes can be placed to ensure that elements are non-inverted. Others, in [13, 14], have looked into untangling and improving quality of high order meshes using vertex insertion and removal, flipping of edges and faces, along with other topological mesh modifications. In [16], the authors investigate extending quality metrics to assess the quality of quadratic elements. Approaches using node-movement based on optimization appear to be limited to [2, 3, 12]. One approach to extend linear quality metrics using an angle-based penalty term was used in [2, 3]. They smoothed triangle and quadrilateral meshes but are limited to isotropic two-dimensional meshes. This work continues and extends the work of [12].

The present work focuses not on detecting whether an element is inverted, but instead on optimizing shape quality such that inverted elements rarely occur. This work follows a node movement strategy called the Target-matrix paradigm [11]. Topological techniques are recognized to be quite powerful and it is hoped that they can eventually be combined with effective node movement strategies such as TXP. This work also expands on recent work to add node-movement capabilities to the Mesquite mesh quality improvement library [1] and

*The Pennsylvania State University, njv116@cse.psu.edu

†Sandia National Laboratories, pknupp@sandia.gov

‡The University of Wisconsin, kraftche@cae.wisc.edu

involved adding support for the 27-point hexahedral element.

2. Mesh Optimization. Mesh optimization here refers to a technique for moving vertices without changing the connectivity. The optimization uses an objective function to quantitatively compare the quality of alternative meshes. Various constraints are also used, for example limits on the positions of boundary vertices. Then the optimization attempts to find the mesh with the minimum objective function score that satisfies the constraints. Often the objective function is a combination of individual component qualities, for example, in this approach we select many sample points and combine the local quality metric scores for all the sample points to create the objective function. As of this writing, most effort on mesh optimization has focused on linear elements. This has produced good element quality metrics for many linear elements.

Currently, Mesquite supports multiple methods to constrain vertex positions. Among them are (a) fixed (the vertex does not move), (b) free (the vertex can move anywhere in the space), and (c) constrained to some geometry (the vertex stays within some lower dimensional space, such as a plane, curve, or point).

Some work has dealt with tangled meshes (meshes containing inverted elements) using mesh optimization. Multiple mesh untangling algorithms have been proposed (see [4, 5, 6, 21] and the contained references). However, none of these algorithms have theoretical guarantees of untangling. Also, focusing only on untangling means that the results often have elements of poor shape quality. This issue is likely to arise in quadratic element meshes often, since curving the boundary of the initial linear mesh to the boundary geometry inverts some elements and causes others to have poor shape quality. The approach in this work does not use such untangling approaches, instead using a TXP approach in an attempt to untangle the mesh while addressing other shape quality issues.

In the Target-matrix paradigm, each element of the mesh has a C^1 mapping from a logical element to itself (the maps used in this work are described in appendix A). For an ideal target element, the Jacobian matrix of this mapping is referred to as W . For an active element the Jacobian matrix is denoted using A . From this one can get a Jacobian matrix for a transformation from the target element to the active element $T = AW^{-1}$. This Jacobian is evaluated at some sample point, k , and is used as the input to a local quality metric which determines the quality of the mesh at the sample point, $\mu_k = \mu(T_k)$. The local quality metric gives a non-negative real number that represents how well A represents W , often using properties such as shape, size, and orientation.

Local quality metrics, μ_k , in the Target-matrix paradigm can be classified based on whether they are barrier metrics or not. A barrier metric can be used with a mesh that is not initially inverted to guarantee that the resulting optimal mesh will not be inverted. A non-barrier metric should be used when the initial mesh is inverted since it allows elements to transition between being inverted and non-inverted. If optimization with a non-barrier metric creates a non-inverted mesh, the result can be further optimized using a barrier metric without worry of the result being inverted.

The first metric of interest is the Shape metric in [7, 8]. It was shown that $\mu_S \geq 0$ for all T and $\mu_S = 0$ if and only if T is a scaled rotation matrix. In that case, A has the form $A = sRW$ with arbitrary positive scalar s and arbitrary rotation R . Then the shape of the active matrix is the shape of the target-matrix. The barrier form of the shape metric in 2D or 3D is:

$$\mu_{SB,2D}(T) = \frac{\|T\|_F^2}{2 \det(T)} - 1 \quad (2.1)$$

$$\mu_{SB,3D}(T) = \frac{\|T\|_F^3}{3 \sqrt{3} \det(T)} - 1 \quad (2.2)$$

Note that $\|\cdot\|_F$ refers to the Frobenius norm (the square root of the sum of the squares of all elements in a matrix). The second metric of interest is the Size (Sz) metric, given in both 2D and 3D by:

$$\mu_{Sz}(T) = (\det(T) - 1)^2 \quad (2.3)$$

This metric obeys $\mu_{Sz} \geq 0$ for all T and $\mu_{Sz} = 0$ if and only if $\det(T) = 1$, i.e., $\det(A) = \det(W)$. Thus, at the minimum, the local sizes of the active and target matrices agree. Because this metric has no barrier, it can potentially untangle an inverted mesh as well as improve relative size. It does not, however, encourage the shape of the active element to be close to the shape of the target element. Thus, the target is chosen so that its primary use in the present application is to encourage mesh untangling because $\det(W) > 0$.

In [9], many issues in measuring sample point quality are investigated. A label invariance property is also discussed, which is important since there are many possible mappings from a logical element to another elements based on the correspondence between logical and physical vertices. Conditions are derived for the metric formulation, sample point selection, and target element selection to guarantee label invariance of the local quality metric. It was shown that the metrics/element types used in the present study are label invariant.

The selection of sample point locations is an important issue and can affect the label invariance of the metric and the ability to detect inverted elements. Since, the current study does not attempt to provide robust untangling, it is the label invariance that dominates this selection. Specifically, we use sample points at all the vertex positions (corner, mid-side, mid-face, mid-element, etc.) that are used to define the element for all elements.¹ Further experiments into robust untangling are left for future work.

The local qualities, μ_k , at all the sample points, k , are combined into an objective function to be used in our numerical mesh optimization. Specifically, we use a power-mean, giving the following objective function:

$$OF = \frac{1}{N} \sum_{\text{Sample Point } k} \mu_k \quad (2.4)$$

3. Algorithm. This work extends the work of the previous summer [12] in which it was found that, for quadratic triangular elements, a two stage algorithm in which the first stage untangles the mesh using a size metric and the second stage maintains the untangled property of the mesh while improving shape quality by using a shape barrier metric. We extend the algorithm to scale the target matrices and scale the and work with quadratic tetrahedral, quadrilateral, and hexahedral elements. This gives the algorithm depicted in figure 3.1. We motivate the choices made in this algorithm, then give the proposed algorithm and present pseudocode.

The algorithm is currently structured to properly handle cases of non-inverted meshes, inverted meshes, and uninvertible meshes. The area optimization is the primary untangling step, and is protected by checks from running on non-inverted meshes to avoid unnecessary computation and because it has been found to tangle meshes with element size heterogeneity. The shape optimization was added because the untangling stage often produces meshes of poor shape quality. It is protected by checks to prevent operating on tangled meshes and also is run on input non-inverted meshes of poor quality in an attempt to improve quality. This is different from the previous structure of two stages, one after the other, in that an initially non-inverted mesh does not go through the area optimization (which may potentially invert the mesh), also a mesh that doesn't untangle in the area optimization avoids the shape

¹Note that a mesh vertex can have multiple associated sample points.

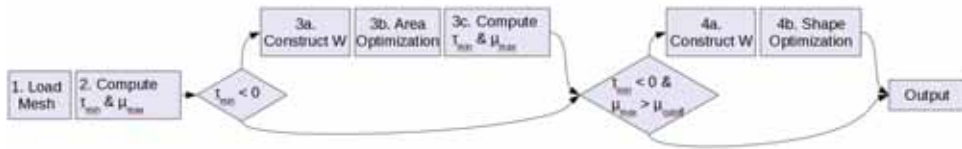


FIG. 3.1. *Quadratic element untangling and shape improvement algorithm.* This is a flow depiction of the algorithm, note there are two branches. The first branch is decided based on whether the mesh is inverted (top is inverted). The second branch is decided based on whether the mesh is poor quality and not inverted (taking the top, otherwise taking the bottom).

optimization, whereas before it would go through the shape optimization enough to produce an error.

The target matrices, W , used in the optimization steps of the proposed algorithm are intended to target the solvers towards ideal elements. here, we use the notation from the $LVQD$ matrix decomposition [10]. In this case Q_{ideal} corresponds to the Jacobian matrix for an ideal element of standard size, skew, and orientation. This suffices for the shape optimization since it is using a shape barrier metric that is sensitive only to the shape of the elements. The size optimization however also uses $\bar{\Lambda}$ so that the target matrix corresponds to an element of appropriate size. Specifically $\bar{\Lambda}$ is the mean over all the sample points of the Λ size values from the $LVQD$ decomposition of the Jacobian matrix of the mapping at that sample point. This setting of $\bar{\Lambda}$ corresponds well to experiments (not presented due to space) that confirmed that a setting significantly larger or smaller tended to hinder the ability of the area optimization to untangle the mesh.

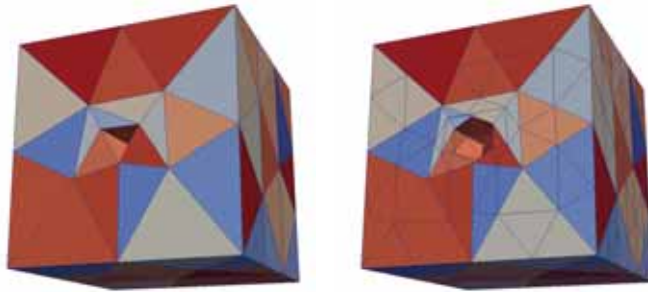


FIG. 3.2. *Boundary Tangling.* Both figures depict a mesh for a cube with a hemisphere cut out. Left is the generated mesh of linear tetrahedra, whereas the right figure depicts the same topology with quadratic tetrahedra. Note that the projection of the mid-side nodes around the edge of the hemispherical cut causes the triangular boundary of one tetrahedron to become tangled. This cannot be untangled without moving vertices on the boundary of the mesh. Coloring is based on element ID since ParaView draws extra edges to connect mid-nodes to the rest of the mesh.

As has been described before, high order meshes tend to have problems on the boundaries. this can cause some trouble if the boundaries are held fixed. One potential problem is depicted in figure 3.2, in which converting the linear mesh to a quadratic one causes a boundary triangle of at least one tetrahedron to become inverted. Thus fixing boundary vertices will mean that this element can not be untangled by the area optimization. This problem is why the boundary geometric constraints were added to this algorithm. Since the inverted boundary triangle can now be untangled by moving the vertices within their relevant boundary geometries.

To be precise, the algorithm starts out by loading the mesh and determining, for each ver-

tex, which geometry the vertex belongs to. This is used to constrain boundary vertices to the appropriate boundary geometry (eg. plane, cylindrical surface, line, circle, point). The second step is to detect whether the mesh contains inverted or poor quality elements (or neither). This is accomplished by looping over all the sample points and detecting the minimal $\det(T)$ value, τ_{min} , and the maximal μ_{SB} value, $(\mu_{SB})_{min}$. The third step then applies only if some inverted sample point is found (i.e. if $\tau_{min} \leq 0$). In the third step, an area optimization attempts to untangle the mesh. Step 3a constructs the target matrices, in this case it corresponds to an ideally shaped element of size $\bar{\Lambda}$ (which is the average of the sizes detected at all the sample points). The area optimization (step 3b) then runs to convergence (400 iterations is more than enough for the meshes investigated) using the geometric constraints, target matrices, and the size metric. It does not use $\bar{\Lambda}$ since the μ_{SB} is size invariant. Afterward it is unknown whether the area optimization was successful, thus step 3c detects if the mesh is inverted or has poor quality. Step four then operates to improve the quality of the mesh without inverting it, thus it should only be run if the mesh has poor quality elements and is not inverted (i.e. $\tau_{min} > 0$ and $(\mu_{SB})_{max} < \mu_{cutoff}$). In our experiments, we use $\mu_{cutoff} = 2$, but other values could be used. Step four starts by defining the target matrices (step 4a) based on an ideal element of unit size, then the shape optimization procedure (step 4b) is run to convergence using the target matrices and geometric constraints previously defined. This shape optimization uses the shape barrier metric in order to prevent the mesh from inverting.

Quadratic Element Untangling and Shape Improvement Algorithm

1. Load mesh and determine geometric constraint
2. Compute minimum area, τ_{min} , and worst quality, $(\mu_{SB})_{max}$
3. If $\tau_{min} \leq 0$
 - (a) Construct Target Matrices, W
 - i. Compute size parameter, $\bar{\Lambda}$
 - ii. $W = \bar{\Lambda} Q_{ideal}$ at all sample points.
 - (b) Area Optimization
 - i. Boundaries: Constrained to geometry.
 - ii. Metric: $\mu_{Sz}(T)$ (Size metric, no barrier).
 - iii. Termination Criterion: 1 iteration outer, 400 inner (converged).
 - iv. Solver: SteepestDescent.
 - (c) Compute τ_{min} and $(\mu_{SB})_{max}$
4. If $\tau_{min} > 0$ AND $(\mu_{SB})_{max} < \mu_{cutoff}$
 - (a) Construct Target Matrices, $W = Q_{ideal}$ at all sample points.
 - (b) Shape Optimization
 - i. Boundaries: Constrained to geometry.
 - ii. Metric: $\mu_{SB}(T)$ The Shape barrier metric.
 - iii. Termination Criterion: 1 iteration outer, 400 inner (converged).
 - iv. Solver: SteepestDescent.

The treatment of the boundary vertices in theory is that they should be constrained to the appropriate geometric entity (corner, curve or surface). This is implemented by setting all the vertices to free and placing the boundary vertices under the ownership of the relevant surface geometric entity in step 1. This ownership is respected in the two optimization stages where the solver optimizes vertex positions using unconstrained optimization techniques, then all vertices are snapped back to their owning geometry.

4. Numerical Examples. In this section we present results of this algorithm applied to various meshes composed of four types of quadratic elements; triangles, quadrilaterals, tetrahedra, or hexahedra. Details of these different element types are presented in appendix

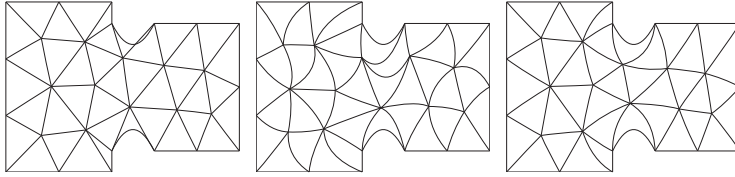


FIG. 4.1. *Smoothing the triangular Part mesh. Input is left, output of the area optimization is center, and algorithm output is right. Note that the area optimization produces poorly shape elements in an untangled mesh.*

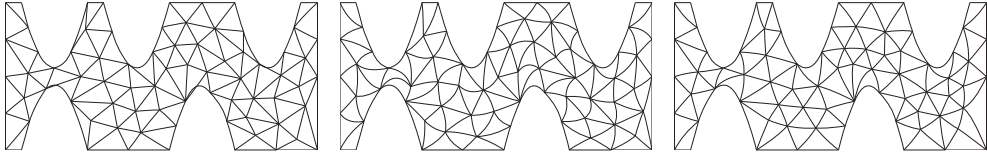


FIG. 4.2. *Smoothing the unperturbed Homogeneous mesh. Initial inverted elements are at the top of the two cuts into the lower edge and at the bottom of the rightmost cut into the top. Input is left, output of the area optimization is center, and algorithm output is right.*

A. Most examples were created using Cubit [19] or Triangle [20] by first creating a mesh of linear elements and then converting it to quadratic elements and projecting boundary mid-side and mid-face nodes to the geometry. Most of the presented examples are cases in which this procedure created tangled meshes.

For the 2D meshes, we present a quadrilateral result (figure 4.3) and two triangle results (figures 4.1 and 4.2). Figure 4.2 was created using triangle, and the rest were created using Cubit. All three of the 2D figures (4.1, 4.2, and 4.3) were created such that the conversion to quadratic elements tangled the mesh.

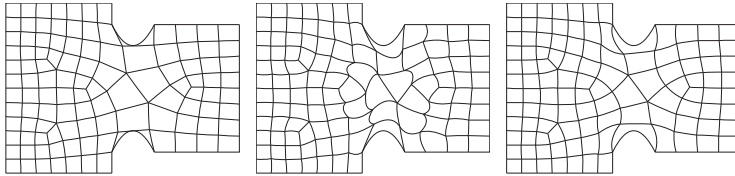


FIG. 4.3. *Smoothing the quadrilateral Part mesh. Input is left, output of the area optimization is center, and algorithm output is right. Note that some interior edges are needlessly curved by the area optimization.*

One can see from figures 4.1, 4.2, and 4.3 that the proposed algorithm is capable of untangling and smoothing some 2D quadratic elements (both tri6 and quad9). In this case one can also see that some of the interior regions far from the tangled elements are significantly affected by the size optimization. Also, many of such elements are moved back to nearly their initial position by the shape optimization. This computational inefficiency is one issue we hope to address in future work.

For the 3D meshes depicted in figures 4.4, 4.5, and 4.6; all were created with Cubit. However the hex27 mesh (figure 4.6) was not tangled after creation, so selected boundary vertices were manually moved to create the tangled elements on the boundary of the mesh.

The figures show that the proposed algorithm is capable of untangling various 3D tangled meshes. Due to the difficulty of visualizing 3D meshes it is difficult to assess the amount that some interior elements may be needlessly changed by the size optimization.

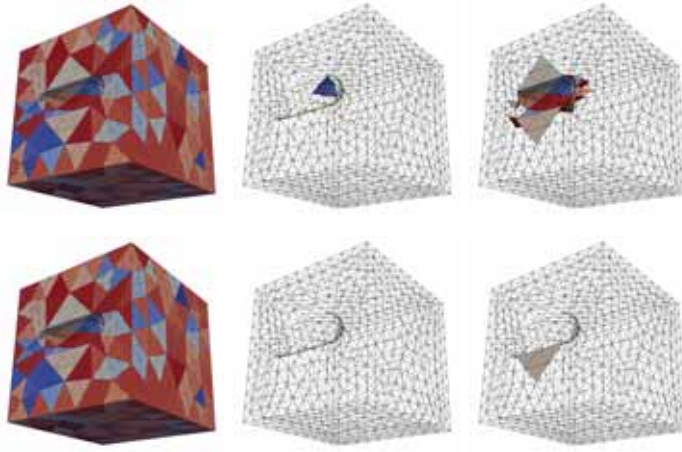


FIG. 4.4. *Untangling and Smoothing Tetrahedral Cut Cube Mesh.* The mesh is tangled as generated by Cubit, without any perturbations. The upper row of the figure has the input and the bottom row has the output. The leftmost figures shows the elements depicted in different colors. The center figures show the inverted elements (elements having a sample point where $\det(T) \leq 0$) along with a wireframe of the entire mesh. The rightmost figures show the poor quality elements (elements having a sample point where $\mu_{SB}(T) \geq 2$) along with a wireframe of the entire mesh. Note that the algorithm untangles the mesh and reduces the amount of poor quality elements.

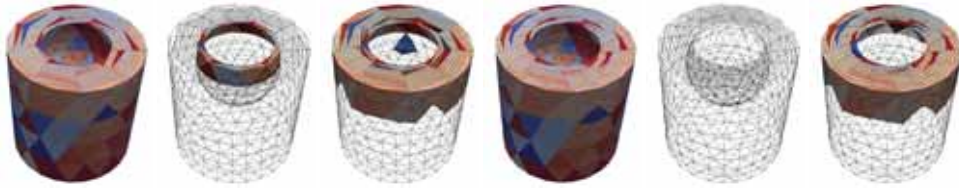


FIG. 4.5. *Untangling and Smoothing Tetrahedral Cut Cylinder Mesh.* This figure shows the results of the proposed algorithm on a mesh created by taking a spherical cut out of a cylinder with a refined top. The idea for this is based on some untangling present in the SLAC mesh. Furthermore, the mesh is tangled as generated by Cubit, without any perturbations. The left three figures depict the input and the right three figures depict the output. The leftmost figures of each side show the elements depicted in different colors. The center figures of each side show the inverted elements (elements having a sample point where $\det(T) \leq 0$) along with a wireframe of the entire mesh. The rightmost figures of each side show the poor quality elements (elements having a sample point where $\mu_{SB}(T) \geq 2$) along with a wireframe of the entire mesh. Note that the algorithm untangles the mesh and reduces the amount of poor quality elements.

5. Conclusions and Future Work. We have given a proposed algorithm extending our previous work [12], to untangle and improve the shape quality of meshes containing quadratic elements. Various aspects of this algorithm were motivated, and results were shown with 6 point triangular elements, 9 point quadratic elements, 10 point tetrahedral elements, and 27 point hexahedral elements. Plans for future work include investigating other and more robust untangling methods into the algorithm. We also plan to look into methods to improve the efficiency of the algorithm, specifically we plan to investigate constraining some of the interior vertices to prevent them from moving in a manner that doesn't hinder the untangling.

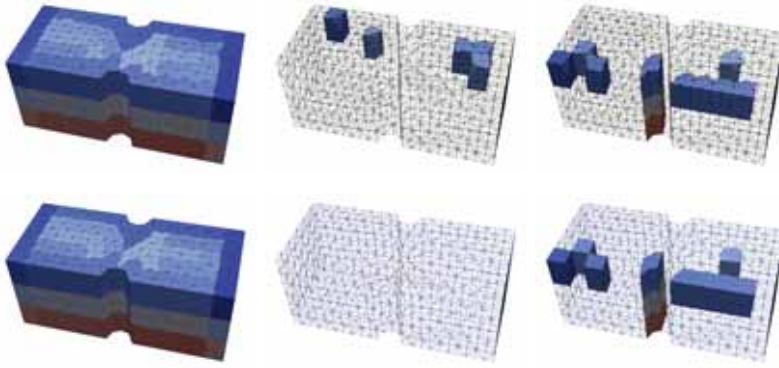


FIG. 4.6. *Untangling and Smoothing Hexahedra.* In this figure, we show the results of untangling and smoothing the 3Dpart Hex27 mesh that has been generated by paving one side, sweeping through the volume and perturbing some of the corner vertices on the boundary of the mesh. The upper row of the figure has the input and the bottom row has the output. The leftmost figures shows the elements depicted in different colors. The center figures show the inverted elements (elements having a sample point where $\det(T) \leq 0$) along with a wireframe of the entire mesh. The rightmost figures show the poor quality elements (elements having a sample point where $\mu_{S_B}(T) \geq 2$) along with a wireframe of the entire mesh. Note that the algorithm untangles the mesh, it also improves the quality of the poorest quality elements, though that is difficult to see.

REFERENCES

- [1] M. BREWER, L. DIACHIN, P. KNUPP, T. LEURENT, AND D. MELANDER, *The Mesquite mesh quality improvement toolkit*, Proceedings of the 12th International Meshing Roundtable, (2003), pp. 239–250.
- [2] Z. CHEN, J. TRISTANO, AND W. KWOK, *Combined Laplacian and optimization-based smoothing for quadratic mixed surface meshes*, Proceedings of the 12th International Meshing Roundtable, (2003), pp. 361–370.
- [3] ———, *Construction of an objective function for optimization-based smoothing*, Engr. w/Cmptns., 20 (2004), pp. 184–192.
- [4] J. ESCOBAR, E. RODRIGUEZ, R. MONTENEGRO, G. MONTERO, AND J. GONZALEZ-YUSTE, *Simultaneous untangling and smoothing of tetrahedral meshes*, Computer Methods in Applied Mathematics and Engineering, 192 (2003), pp. 2775–2787.
- [5] L. FREITAG AND P. PLASSMANN, *Local optimization-based untangling algorithms for quadrilateral meshes*, (2001).
- [6] P. KNUPP, *Hexahedral mesh untangling & algebraic mesh quality metrics*, Proceedings of the 9th International Meshing Roundtable, (2000), pp. 173–183.
- [7] ———, *Local 2D metrics for mesh optimization in the target-matrix paradigm*, Sandia National Laboratories, (2006), pp. SAND2006–7382J.
- [8] ———, *Analysis of 2D, rotation-invariant, non-barrier metrics in the target-matrix paradigm*, Sandia National Laboratories, (2008), pp. SAND2008–8219P.
- [9] ———, *Label-invariant mesh quality metrics*, Proceedings of the 18th International Meshing Roundtable, (2009), pp. 139–155.
- [10] ———, *Target-matrix construction algorithms*, Sandia National Laboratories, (2009), pp. SAND2009–7003P.
- [11] ———, *Introducing the target-matrix paradigm for mesh optimization via node-movement*, to appear in Proceedings of the 19th International Meshing Roundtable, (2010).
- [12] P. KNUPP, N. VOSHELL, AND J. KRAFTCHECK, *Quadratic triangle mesh untangling and optimization via the target-matrix paradigm*, CSRI Summer Proceedings, (2009), pp. SAND2010–3083P.
- [13] X. LUO, M. SHEPHARD, L. LEE, L. GE, AND C. NG, *Moving curved mesh adaptation for higher order finite element simulations*, Engineering with Computers, submitted., (2006), pp. 42–142.
- [14] X. LUO, M. SHEPHARD, L. LEE, C. NG, AND L. GE, *Tracking adaptive moving mesh refinements in 3d curved domains for large-scale higher order finite element simulations*, Proceedings of the 17th International Meshing Roundtable, (2008), pp. 585–601.
- [15] S. ROBERT, R. O’BARA, AND M. SHEPHARD, *Curvilinear mesh generation in 3D*, Proceedings of the 8th International Meshing Roundtable, (1999), pp. 407–417.
- [16] A. SALEM, S. CANANN, AND S. SAIGAL, *Robust distortion metric for quadratic triangular 2D finite elements*, Trends in Unstructured Mesh Generation, ASME, AMD-Vol. 220 (1997), pp. 73–80.
- [17] ———, *Mid-node admissible spaces for quadratic triangular arbitrarily curved 2D finite elements*, Int. J.

- Numerical Methods in Engineering, 50(2) (2001), pp. 253–272.
- [18] A. SALEM, S. SAIGAL, AND S. CANANN, *Mid-node admissible space for 3D quadratic tetrahedral finite elements*, Engineering with Computers, 17 (2001), pp. 39–54.
- [19] SANDIA, *The cubit website*, <http://www.cubit.sandia.gov/>, (2010).
- [20] J. SHEWCHUK, *Triangle: Engineering a 2d quality mesh generator and delaunay triangulator*, Applied Computational Geometry: Towards Geometric Engineering, 1148 (1996), pp. 203–222.
- [21] P. VACHAL, R. V. GARIMELLA, AND M. J. SHASHKOV, *Untangling of 2d meshes in ale simulations*, Journal of Computational Physics, 196 (2004), pp. 627–644.

Appendix

A. Quadratic Element Maps and Target Matrices. The four shapes focused on in this work are depicted in figure A.1, which shows the location and ordering of the points. Using the standard basis vectors for \mathbb{R}^2 one can define a logical triangle which serves as the master element for the transformations used by quadratic elements.

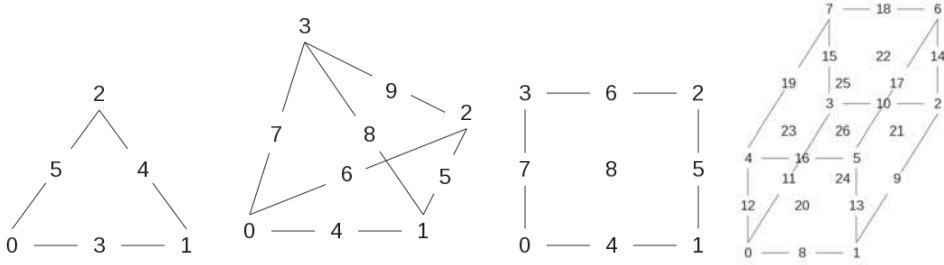


FIG. A.1. *Element Shapes.* Depicted are the 6 point triangle (left-most), 9 point quadrilateral (center left), 10 point tetrahedron (center right), and 27 point hexahedron (right-most).

Based on the vertex ordering depicted in A.1, the quadratic triangle has the following mapping:

$$\vec{x}_{tri6}(r, s) = \sum_{i=0}^{n-1} N_i(r, s) \vec{x}_i \quad (\text{A.1})$$

$$u = 1 - r - s \quad (\text{A.2})$$

$$N_0 = u(2u - 1) \quad (\text{A.3})$$

$$N_1 = r(2r - 1) \quad (\text{A.4})$$

$$N_2 = s(2s - 1) \quad (\text{A.5})$$

$$N_3 = 4ru \quad (\text{A.6})$$

$$N_4 = rs \quad (\text{A.7})$$

$$N_5 = 4su \quad (\text{A.8})$$

The 10 point tetrahedral mapping is as follows:

$$\vec{x}_{tet10}(r, s, t) = \sum_{i=0}^{n-1} N_i(r, s, t) \vec{x}_i \quad (\text{A.9})$$

$$\begin{aligned} u &= 1 - r - s - t \\ N_1 &= u(2u - 1) & N_4 &= t(2t - 1) & N_8 &= 4tu \\ N_2 &= r(2r - 1) & N_5 &= 4ru & N_9 &= 4rt \\ N_3 &= s(2s - 1) & N_6 &= 4rs & N_{10} &= 4st \\ & & N_7 &= 4su & & \end{aligned} \quad (\text{A.10})$$

The mapping and target matrices are different for the other element types. The Quad9 (nine point quadrilateral) and Hex27 (twenty seven point hexahedron) elements are closely related and can be expressed using the following basic approaches:

$$\vec{x}_{quad9}(r, s) = \sum_{i=0}^{n-1} N_i(r, s) \vec{x}_i \quad (\text{A.11})$$

$$\begin{aligned} N_0 &= (r-1)(2r-1)(s-1)(2s-1) & N_5 &= r(2r-1)4s(1-s) \\ N_1 &= r(2r-1)(s-1)(2s-1) & N_6 &= 4r(1-r)s(2s-1) \\ N_2 &= r(2r-1)s(2s-1) & N_7 &= (r-1)(2r-1)4s(1-s) \\ N_3 &= (r-1)(2r-1)s(2s-1) & N_8 &= 4r(1-r)4s(1-s) \\ N_4 &= 4r(1-r)(s-1)(2s-1) & & \end{aligned} \quad (\text{A.12})$$

$$\vec{x}_{hex27}(r, s, t) = \sum_{i=0}^{n-1} N_i(r, s, t) \vec{x}_i \quad (\text{A.13})$$

$$\begin{aligned} l_1(\xi) &= (\xi-1)(2\xi-1) & l_2(\xi) &= 4\xi(1-\xi) & l_3(\xi) &= \xi(2\xi-1) \\ N_0 &= l_1(r)l_1(s)l_1(t) & N_9 &= l_3(r)l_2(s)l_1(t) & N_{18} &= l_2(r)l_3(s)l_3(t) \\ N_1 &= l_3(r)l_1(s)l_1(t) & N_{10} &= l_2(r)l_3(s)l_1(t) & N_{19} &= l_1(r)l_2(s)l_3(t) \\ N_2 &= l_3(r)l_3(s)l_1(t) & N_{11} &= l_1(r)l_2(s)l_1(t) & N_{20} &= l_2(r)l_1(s)l_2(t) \\ N_3 &= l_1(r)l_3(s)l_1(t) & N_{12} &= l_1(r)l_1(s)l_2(t) & N_{21} &= l_3(r)l_2(s)l_2(t) \\ N_4 &= l_1(r)l_1(s)l_3(t) & N_{13} &= l_3(r)l_1(s)l_2(t) & N_{22} &= l_2(r)l_3(s)l_2(t) \\ N_5 &= l_3(r)l_1(s)l_3(t) & N_{14} &= l_3(r)l_3(s)l_2(t) & N_{23} &= l_1(r)l_2(s)l_2(t) \\ N_6 &= l_3(r)l_3(s)l_3(t) & N_{15} &= l_1(r)l_3(s)l_2(t) & N_{24} &= l_2(r)l_2(s)l_1(t) \\ N_7 &= l_1(r)l_3(s)l_3(t) & N_{16} &= l_2(r)l_1(s)l_3(t) & N_{25} &= l_2(r)l_2(s)l_3(t) \\ N_8 &= l_2(r)l_1(s)l_1(t) & N_{17} &= l_3(r)l_2(s)l_3(t) & N_{26} &= l_2(r)l_2(s)l_2(t) \end{aligned} \quad (\text{A.14})$$

For an isotropic domain, there is no reason for the target element to break symmetry, so the equilateral triangle is a good choice for the target. For theoretical reasons explained in [10], we choose the area of the target equilateral triangle to be one. Then the Jacobian of the logical to target mapping is as follows:

$$W_{ideal,tri6} = \frac{1}{\sqrt{2}\sqrt{3}} \begin{bmatrix} 2 & 1 \\ 0 & \sqrt{3} \end{bmatrix} \quad (\text{A.15})$$

The tetrahedral ideal element also corresponds to an equilateral element, and has the following W_{ideal} :

$$W_{ideal,tet10} = 2^{-5/6} \begin{bmatrix} 2 & 1 & 1 \\ 0 & \sqrt{3} & \sqrt{\frac{1}{3}} \\ 0 & 0 & 2\sqrt{\frac{2}{3}} \end{bmatrix} \quad (\text{A.16})$$

For the quadrilateral and hexahedral elements, the W_{ideal} from equilateral elements of unit size is an identity matrix (2x2 for quadrilateral and 3x3 for hexahedral).

$$W_{ideal,quad9} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (\text{A.17})$$

$$W_{ideal,hex27} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (\text{A.18})$$

A NEW STRATEGY FOR UNTANGLING 2D MESHES VIA NODE-MOVEMENT

JASON W. FRANKS[†] AND PATRICK M. KNUPP[‡]

Abstract. A new mesh optimization strategy for untangling quadrilateral meshes, based on node-movement, is investigated. The strategy relies on a set of Propositions which show that, for certain quality metrics $0 \leq \mu < \infty$ within the Target-matrix paradigm, if $\mu < 1$ then the local area is positive. The Propositions are exploited in devising a new strategy for simultaneous mesh untangling and quality improvement. Numerical results confirm the expected behavior of the new strategy.

1. Introduction. A common problem in mesh generation is the creation of meshes having inverted, invalid, or tangled elements. In general, such meshes cannot be used in computer simulations. Many methods for repairing or untangling inverted meshes exist, including re-meshing, local mesh modification, and mesh optimization and smoothing via node-movement techniques. Re-meshing tends to be non-automatic. Local mesh modification can be very effective but, for the most part, is limited to simplicial meshes. Mesh optimization can address meshes containing a wider variety of element types, but current methods do not guarantee that the optimized mesh will be untangled. While we recognize the value of re-meshing and local mesh modification methods in mesh untangling, in this paper we set them aside in order to focus on node-movement methods.

There exists a wide variety of node movement methods and the majority of them do not specifically address the mesh untangling problem. For example, Laplace smoothing is widely used to create 'smooth' meshes consisting of well-shaped elements. The fact that Laplace smoothing can sometimes untangle an inverted mesh is somewhat incidental in that it is not specifically designed to do so. In fact, it is well known that Laplace smoothing can create a tangled mesh from one that initially is untangled. Many other smoothing and optimization methods behave similarly. The common approach to avoiding tangled meshes has been to devise node-movement strategies with 'invertibility guarantees', i.e., they guarantee that the result of the node-movement will be a non-inverted mesh. Winslow smoothing is perhaps the first node-movement method which provided such a guarantee [11]. However, the guarantee is limited to 2D meshes only. Moreover, the method has proved somewhat difficult to extend to unstructured meshes.

Winslow smoothing is based on the numerical solution of a partial differential equation. Other node-movement strategies are based on numerical optimization of objective functions which are defined in terms of discrete geometric entities such as edge-lengths, angles, areas, and volumes. These methods are more easily applied to unstructured meshes and, moreover, provide node-movement methods similar to Winslow. The main example would be the discrete objective function based on a summation of the squares of edge-lengths, divided by the local area or volume:

$$F = \sum \frac{\ell_1^2 + \ell_2^2}{V} \quad (1.1)$$

These methods belong to a set of methods known as *barrier* methods. In this example, elements having volumes close to zero are penalized because the objective function tends to increase rapidly as the volume approaches zero. If all the elements of the initial mesh have positive volume, then optimization of this objective function will (when properly implemented)

[†]jasonwfranks@gmail.com

[‡]Sandia National Laboratories, pknupp@sandia.gov

guarantee that the minimizing mesh is untangled. A drawback to the barrier methods, however, is that they cannot be used when the initial mesh is tangled (because there is no way to cross the barrier to the non-inverted region).

Two approaches have been proposed to overcome the limitation that barrier methods cannot be applied to tangled initial meshes. In the first, the barrier is moved by replacing V in the objective function by $V - V_{min}$, where V_{min} is the minimum volume over the initial mesh. A homotopy method is applied to gradually increase the minimum volume until it is positive (see [1] for details). While this method is often effective, it is relatively expensive and does not guarantee that the final result will be non-inverted. In the second approach, the barrier is removed entirely, with V being replaced by a blending function that is approximately equal to $(V + |V|)/2$. As V goes from positive to negative, the blending function transitions from V to zero, so that elements having negative areas are heavily penalized (see [2] for details). While effective, the method does not provide an invertibility guarantee.

As an alternative to node-movement methods that provide an invertibility guarantee, there exist methods which directly try to untangle an inverted initial mesh. We refer to those methods as *pure* untanglers because they are specifically designed to untangle meshes and generally neglect other aspects of mesh quality such as element Shape. The intended use of pure untangle methods is to first untangle a mesh and then to subsequently apply a barrier method to improve Shape or other mesh qualities.

Freitag and Plassman developed a pure untangler based on a system of linear programs with the intent to maximize the minimum area/volume of elements within each local patch of the mesh [3, 4]. When optimized, the system is guaranteed to converge, but not necessarily to an untangled mesh. This method proves to be very effective for untangling but counter-productive for mesh improvement, because ‘a small but perfectly shaped element is likely to be distorted in an effort to maximize its area’ [10].

Shashkov et. al. used a computational geometry approach to compute feasible regions in which a vertex could be placed to produce positive areas [6, 7, 10]. The feasibility region for a given free vertex is the space in which that vertex can be shifted such that all of its incident elements become or remain uninverted. By constructing halfspaces parallel to the edges of each element adjacent to the vertex in question, the feasibility region can be constructed from the intersection of these halfspaces (note that the feasible region can be empty for a given free vertex). The free vertex is then placed in the center of this feasible set. The feasible region approach to untangling proves very effective in removing large quantities of invalid elements from meshes, and is useful in multi-stage algorithm approaches to untangling. But disadvantages do exist when using this method. First, in local submeshes where elements take extreme shapes or very small size, the feasibility region is naturally very small. When a large cluster of vertices and elements exist with minuscule feasibility regions it often occurs that passes over the submesh will yield null or oscillatory results, as moving one vertex to untangle one element will tangle and invert another element. The second weakness is that the feasible regions are more difficult to calculate for elements having non-planar faces (e.g. hexahedra). The commonly practiced solution in this event is to employ the Simplex method. By optimizing just four linear functions, $f = x$, $f = -x$, $f = y$, $f = -y$, as opposed to one for every edge, and if the feasible set is not empty, a subset can be found [10].

Knupp gave a method of untangling that used a local non-barrier metric that penalized inverted elements [7]. Global optimization of the objection function proved to be more robust in terms of untangling larger and more difficult meshes, as every element is considered only once and simultaneously via a global sweep. This ‘Untangle-Beta’ metric contained a global parameter β which was needed to ensure that elements of exactly zero area were not created. Proper selection of the β value was sometimes problematic, particularly if the area of elements

in the mesh was highly variable. The choice of the value of β sometimes made the difference as to whether the optimal mesh was tangled or not. Another problem with the metric was that it was non-differentiable at certain points. R. Liska showed that this problem could be fixed by squaring the metric, thus making the method susceptible to standard optimization methods. As with the other pure untanglers, there is no guarantee of obtaining an untangled mesh with this method.

The focus in the present paper is on a new strategy for untangling 2D meshes via the Target-matrix Optimization Paradigm (TMOP) [9]. TMOP is an algebraic approach rooted in matrix analysis, allowing for the development of metrics with the property to be able to improve or preserve specific mesh qualities such as element size, shape, and orientation. This is accomplished by establishing a collection of *sample points* within the elements of the mesh, as well as a set of Target-matrices that describe the desired optimal mesh. For linear quadrilateral mesh elements, one sample point is normally assigned per corner, and that is the choice used in this work. Local metrics in TMOP are functions from a set of real, square matrices $T_{d \times d}$, with $d = 2$ for 2D meshes. In turn, $T = AW^{-1}$, where A is the Jacobian matrix of the element map and W the target matrix. Target matrices are constructed prior to the optimization phase and describe the Jacobian of the map in the optimal mesh; targets are constructed such that $\det(W) > 0$. The determinant of A gives the local size at a sample point in the active mesh and $\tau = \det(T) = \det(A)/\det(W)$ measures Sizes in the active mesh relative to the desired optimal mesh. In order to use TMOP metrics, one must specify the targets. In this study, targets of the form

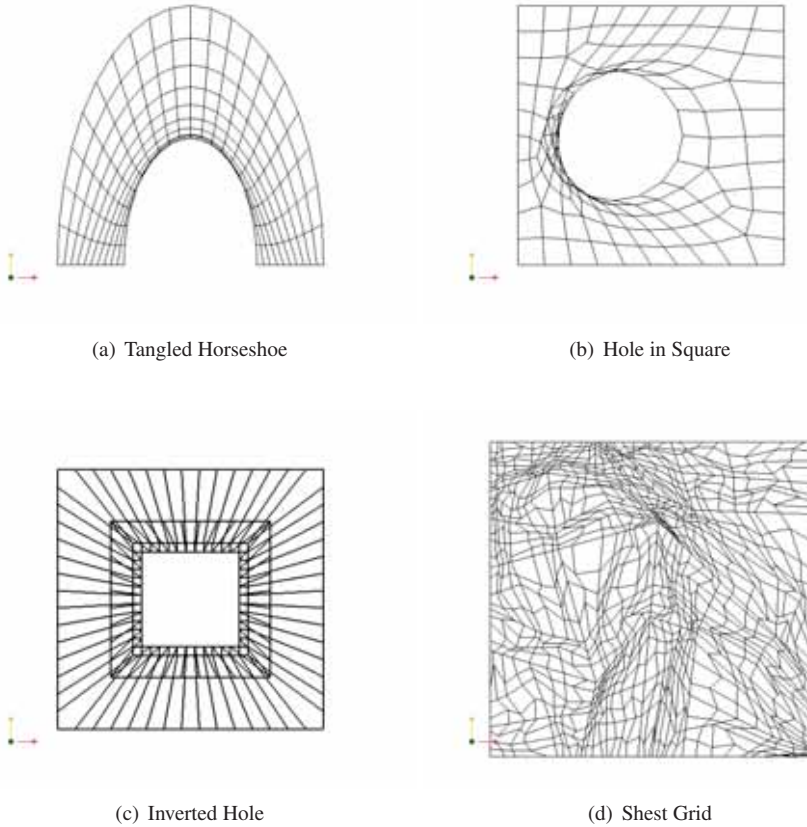
$$W = \bar{\Lambda} S_{ideal} \quad (1.2)$$

are used. The first term in this product is a scalar and is equal to the average edge length in the mesh. The second term in the product is a matrix which describes the ideal shape of the element; for quadrilateral elements, S_{ideal} is the Identity matrix. This particular form of the Target-matrices says that the desired Jacobian matrices in the optimal mesh should be squares whose edge lengths are given by $\bar{\Lambda}$. Although in general the set of Target-matrices can vary from one sample point to another, in this study a constant Target suffices. If a different target form were chosen, the results in this study might change.

To clarify the terminology used in this paper, we first define a mesh to be tangled or inverted if it contains an inverted element. In the Finite Element Method, an inverted element corresponds to an element for which there is a point within the master element for which the map is non-invertible. Unfortunately, it can be rather difficult to tell whether or not an element is inverted based on this definition, particularly for linear hexahedral elements and quadratic or higher-order elements. For convenience, we substitute for this definition the following: within TMOP, an element is inverted if it contains an invalid sample point. An invalid sample point is a sample point for which the determinant of the matrix A (or T) at the sample point is less than or equal to zero. An element containing an invalid sample point is always inverted, but elements can also be inverted (according to the strict Finite Element definition) without containing an invalid sample point. In practice, this substitute definition for inverted elements is quite useful.

2. Establishing a Baseline. In this section we describe four tangled initial meshes and show the results of applying either Laplace smoothing or the Untangle-Beta metric to them. This is useful when later comparing results to the new untangle methods described later. The four tangled meshes in Figure 2.1 constitute the set of 2D test meshes we use to study the behavior of untangling and other algorithms' behavior.

All of our 2D test meshes are composed of quadrilateral elements because experience has shown that 2D meshes with triangle elements are relatively easy to untangle. The tangled

FIG. 2.1. *Four tangled test meshes.*

Horseshoe was created by applying the Laplace smoother to an untangled Horseshoe mesh. The Hole-in-Square mesh is a result of a paved mesh subjected to Laplacian smoothing. The Inverted Hole mesh was produced by a sweeping algorithm. The Shest grid mesh was produced by an ALE calculation. It is known a priori that untangled meshes having the same mesh topology in all four of these test meshes exist.¹ More difficult test meshes than these four exist, but the four are adequate for a preliminary investigation of our strategy.

The degree of inversion or tangling varies from one initial mesh to another, with as little as 9 inverted elements (Shest Grid) to 40 (Inverted Hole). As was mentioned in Section 1, some smoothers that improve mesh quality (e.g., Laplace smoothing) can incidentally untangle meshes without having been designed with this goal as their primary function. For comparison to our new algorithms, Fig. 2.2 shows how the Laplacian smoother performed on our test meshes.² The Horseshoe and Hole-in-Square meshes are not shown because the Laplacian smoother (run to convergence) was used to generate them and therefore applying Laplace smoothing again will fail to untangle them.

¹In general, given a tangled mesh, there is no way to determine a priori if an untangled mesh with the same connectivity exists.

²All numerical results in this paper kept the boundary vertex coordinates unchanged so that only interior vertices were moved as a result of smoothing or optimization.

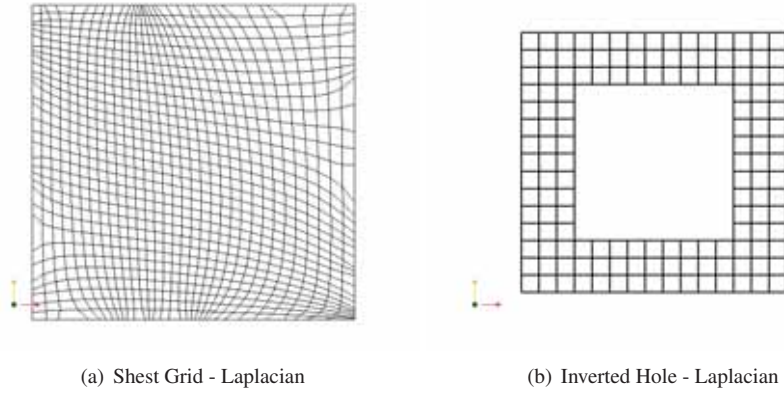


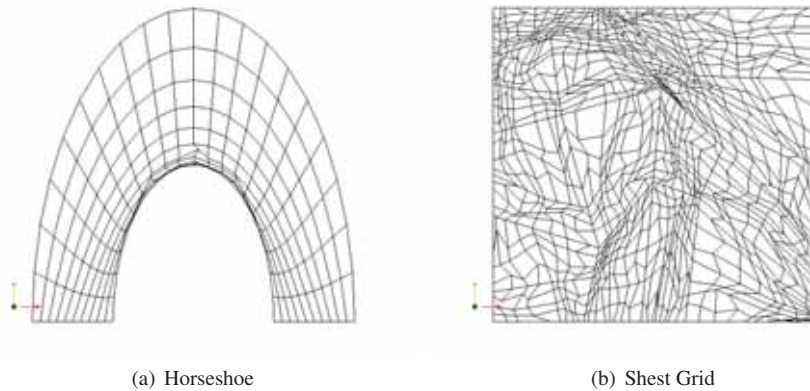
FIG. 2.2. Laplacian smoothing applied to two initially tangled meshes.

As can be seen in Fig 2.2, the Laplacian smoother completely untangled the Shest and Inverted Hole meshes when driven to convergence. This illustrates that some smoothers can untangle a mesh, even though that is not their main purpose. Of course, we see that the Shest grid's original character was destroyed during the untangling, so that could make the use of Laplace inappropriate in this instance. If the Laplacian smoother is not driven to convergence, results may differ in terms of number of inverted elements and invalid points.

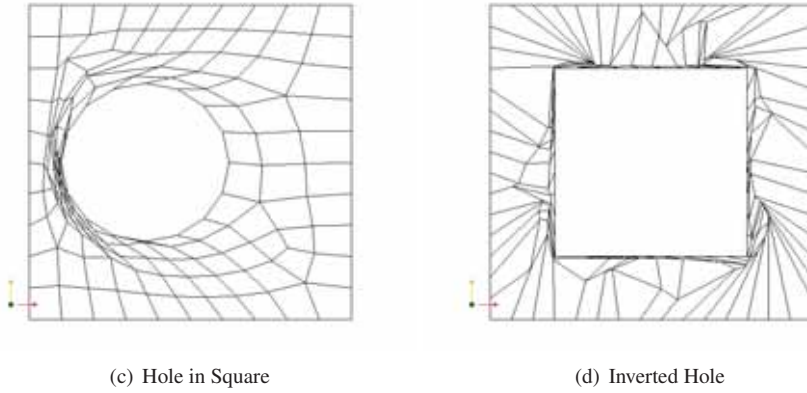
As an additional comparison, we apply the Untangle-Beta untangler to the four tangled test meshes. The Untangle-Beta metric has the form:

$$\mu_{\beta}(T) = \{|\tau - \beta| - (\tau - \beta)\}^2 \quad (2.1)$$

with $\beta > 0$ an adjustable parameter. This metric is non-negative, with $\mu_{\beta} = 0$ if and only if $\tau \geq \beta$ [6]. Unlike the Laplacian smoother, the Untangle-Beta metric was specifically designed to untangle meshes, i.e. it is a 'pure' untangler that does not guarantee Shape or Size quality. Figure 2.3 shows the results.³ With β set to $\bar{\Lambda}^2/20$ (approximately 1/20th of the average cell area) for each mesh, all four meshes were untangled. As expected for pure untanglers, the Shape quality in these meshes is poor.



³The Laplace, Untangle-Beta, and newer algorithms were implemented in the Mesquite code.

FIG. 2.3. All four meshes after optimization with the μ_β metric.

	Horseshoe	Shest	Hole-in-Square	Inverted Hole
Initial	-0.178098	-0.315243	-0.957579	-1.47294
Laplace	-0.178098	0.106715	-0.957579	0.707527
μ_β	0.00480571	0.00121174	0.0202237	0.014298

TABLE 2.1

 τ_{min} for each mesh before and after Laplace and μ_β .

		Number of Inverted Elements							
		Horseshoe		Shest		HoleSq		Inv Hole	
		I	O	I	O	I	O	I	O
Laplacian		12	12	9	0	23	23	40	0
μ_β		12	0	9	0	23	0	40	0

TABLE 2.2

Initial Mesh (I) and Optimized Mesh (O).

		Number of Invalid Sample Points							
		Horseshoe		Shest		HoleSq		Inv Hole	
		I	O	I	O	I	O	I	O
Laplacian		44	44	9	0	70	70	156	0
μ_β		44	0	9	0	70	0	156	0

TABLE 2.3

Initial Mesh (I) and Optimized Mesh (O).

Define τ_{min} to be the minimum value of τ over all the sample points in a given mesh. A non-positive value for τ_{min} signifies a tangled mesh. Entries in Table 2.1 give the minimum values for each of the four initial meshes and the results after Laplace smoothing and the Untangle-Beta were applied. Because $\tau_{min} > 0$ after optimization with μ_β , we conclude that the Untangle-Beta metric is able to untangle all four test meshes. Notice also that when Laplacian smoothing is able to untangle a mesh, the corresponding value of τ_{min} is often larger than what one gets when using Untangle-Beta. This is because Laplacian smoothing is not a pure untangler.

Tables 2.2 and 2.3 report the number of inverted elements and the number of invalid

sample points in each of the four meshes, both before and after optimization.

Except for Laplace, all numerical optimization results reported in Sections 2 and 3 used the Block Coordinate Descent method (local patches with a global objective function), a tight termination criterion of 'absolute vertex movement' less than $\bar{\Lambda}/1000$, and Mesquite's Steepest Descent Solver. The Objective Function was simply the linear average of the metric over all sample points in the mesh.⁴

3. A New Strategy for Mesh Untangling. Previous untangling strategies rely on computational geometry, simplex methods, or on 'untangling' metrics. In this section we extend the latter strategy though the use of hybrid metrics which improve Size or Shape+Size, while at the same time encouraging untangling. Our strategy is particularly interesting in that the approach is based on a series of newly proved Propositions. First we briefly discuss previously-studied Size and Shape+Size metrics and how they behave with respect to untangling. Second, we prove certain Propositions about these metrics. Third, the Propositions are exploited to devise hybrid metrics that encourage untangling, as well as improve Size and Shape+Size. Fourth, numerical results for the new strategy are shown, based on the four tangled initial meshes.

3.1. Size and ShapeSize. In this section we consider two previously proposed TMOP quality metrics [8] because they are key components in our new strategy for untangling. The first metric is the so-called relative Size metric, given by

$$\mu_{Sz}(T) = (\tau - 1)^2 \quad (3.1)$$

It is easy to see that $0 \leq \mu_{Sz}$ with $\mu_{Sz} = 0$ if and only if $\tau = 1$. The primary purpose of this metric is thus to make the local Size in the optimal mesh as close as possible to the Size $\bar{\Lambda}$ specified by the Target-matrices. Although the metric is not designed to be a pure untangle metric, it is clear that this metric would tend to untangle meshes since $\mu_{Sz} = 0$ means that $\det(A) = \det(W) > 0$.

The second metric is designed to improve element Shape and Size:

$$\mu_{ShSz}(T) = |T|^2 - 2\psi(T) + 2 \quad (3.2)$$

with

$$\psi(T) = \sqrt{|T|^2 + 2\tau} \quad (3.3)$$

In previous work it was shown that $0 \leq \mu_{ShSz}$ with $\mu_{ShSz} = 0$ if and only if $T = R$, where R is any rotation matrix. That means the metric is invariant to orientation, but sensitive to Shape and Size. Although the metric is not designed to be a pure untangle metric, it is clear that this metric would tend to untangle meshes since $\mu_{Sz} = 0$ means that $\det(T) = \det(R) = 1$ and thus $\det(A) > 0$.

So, much like Laplace smoothing, both the Size and the ShapeSize metric, with the Target-matrix given in equation (1.2), will tend to create untangled meshes when optimized; they do not guarantee it, however. The two metrics are applied to the four tangled test meshes to illustrate the expected behavior..

⁴One point on which we elaborate is the use of the Block Coordinate Descent (BCD) algorithm. An alternative provided in Mesquite is the Global Patch solver which moves all the 'free' mesh vertices simultaneously. Unfortunately, it was found that the Global Patch did not prove as effective as BCD in terms of speed or robustness. The Shrest Grid eventually converged to the same results as BCD, but took a substantially longer time to terminate with Global Patch, and the Hole-in-Square mesh always terminated, but not always to the same final mesh (suggesting that there may be tangled local minima in the global objective function). Problems also arose when Global Patch was utilized with the SizeUntangle metric because convergence to the final untangled mesh for this metric took the longest amount of computer time.

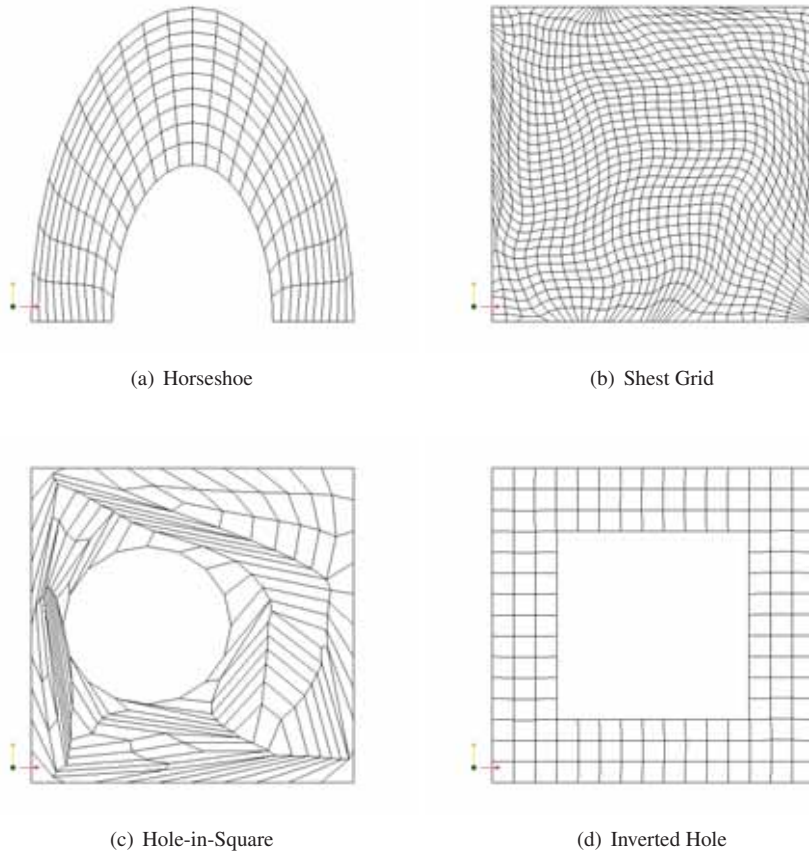


FIG. 3.1. All four meshes after optimization with the μ_{Sz} metric.

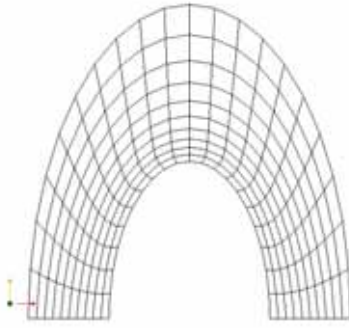
Figure 3.1 gives results for optimization with the Size metric. As this figure demonstrates, the Size metric created near-equal area elements and somewhat non-Smooth meshes; this is the expected behavior of this metric. More to the point, it was able to untangle three of the four meshes. It failed to untangle the Hole-in-Square, but reduced the number of inverted elements from 23 to 6. We conjecture that, had we used a more sophisticated Target-matrix in which the Size factor is allowed to vary with the sample point, the mesh might have untangled with the Size metric.

Figure 3.2 shows how the ShapeSize metric performed. The same three meshes were untangled when using the ShapeSize metric. The Size metric reduced the Hole-in-Square mesh to 6 inverted elements compared to 12 for the ShapeSize metric. Comparing the results in Figure 3.2 to 3.1 one sees the expected characteristic differences between the Size and ShapeSize metrics. The results of ShapeSize bears a strong resemblance to the Laplacian results as well for the Shest Grid and the Inverted Hole meshes.

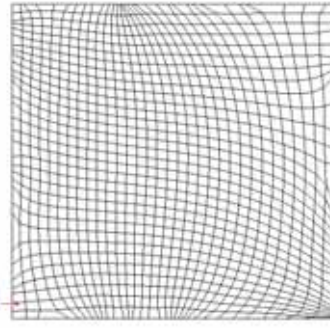
Now that we have some data on the effects of some basic TMOP metrics on tangled meshes, we give some Propositions that will allow us to develop new untangling metrics.

3.2. Propositions. In this section it will be shown that certain local metrics μ within TMOP enjoy the following property: for any $d \times d$ real matrix T , if $\mu(T) < 1$, then $\tau > 0$.

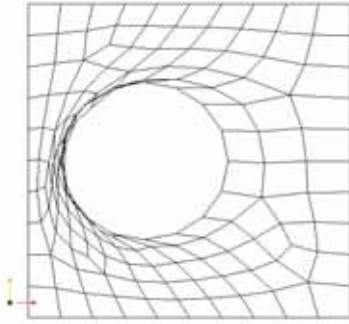
Proposition 3.2.1. Let T be a $d \times d$ matrix. If $\mu(T) = (\tau - 1)^2 < 1$, then $0 < \tau < 2$.



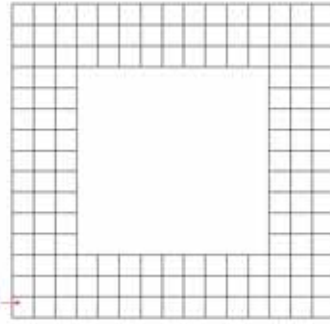
(a) Horseshoe



(b) Shest Grid



(c) Hole-in-Square



(d) Inverted Hole

FIG. 3.2. All four meshes after optimization with the μ_{ShSz} metric.

Proof. Suppose the metric is less than one. Then

$$(\tau - 1)^2 < 1$$

$$\tau^2 - 2\tau + 1 < 1$$

$$\tau^2 - 2\tau < 0$$

$$\tau(\tau - 2) < 0$$

So either: A) $\tau < 0$ and $\tau - 2 > 0$ or B) $\tau > 0$ and $\tau - 2 < 0$.

Case A is impossible, leaving B: $0 < \tau < 2$. \square

Thus, when the Size metric is less than 1, the determinant is positive. On the other hand, having the determinant positive does not necessarily mean that the Size metric will be less than 1.

Here is another metric which enjoys a similar property:

Proposition 3.2.2. Let T be a $d \times d$ real matrix. If $\mu(T) = |T|^2 - 2\psi(T) + 2 < 1$, then $\tau > 0$.

Proof. Suppose the metric is less than one. Then

$$|T|^2 - 2\psi + 2 < 1$$

$$|T|^2 - 2\psi + 1 < 0$$

$$|T|^2 + 1 < 2\psi$$

But $(|T| - 1)^2 \geq 0$ leads to $2|T| \leq |T|^2 + 1$. Combining these results gives

$$2|T| \leq |T|^2 + 1 < 2\psi$$

and thus $|T| < \psi$. Therefore

$$|T| < \psi = \sqrt{|T|^2 + 2\tau}$$

$$|T|^2 < |T|^2 + 2\tau$$

and so we have $0 < \tau$. \square

Although Proposition 3.1.2 is proved for $d \times d$ matrices, the metric itself is not a Shape metric unless $d = 2$, and thus the Proposition is only useful for 2D meshes. For 3D meshes, the Shape metric has a different form [5].

The converse to this Proposition is only true if T is such that $\psi(T) = 1$. To see this, suppose $0 < \tau$. Then

$$\begin{aligned} 0 &< 2\tau \\ |T|^2 &< |T|^2 + 2\tau \\ |T|^2 &< \psi^2 \\ |T|^2 - 2\psi + 2 &< \psi^2 - 2\psi + 2 \\ \mu(T) &< 1 + (\psi - 1)^2 \end{aligned} \tag{3.4}$$

As a final result, we have the following Proposition for the 2D non-barrier Shape metric $\mu_{Sh}(T) = |T|^2 - 2\tau$:

Proposition 3.2.3 Let T be a 2×2 matrix. If $\mu_{Sh}(T) = 0$, then $\tau \geq 0$.

Proof. In [8] it was shown that $\mu_{Sh} = 0$ if and only if T is a scaled rotation, i.e., T has the form $T = sR$. In that case, $\tau = \det(sR) = s^2 \geq 0$. \square

3.3. Exploiting the Propositions. Both the Size and ShapeSize metrics have the potential to untangle meshes because their target matrices are non-inverted. Based on the Propositions of the previous section, these metrics can further be used to devise new metrics that encourage untangling even more. Let μ be a TMOP metric such that $\mu < 1$ guarantees that $0 < \tau$. Then consider the following function $\tilde{f}(T) = f(\mu(T))$, which can be considered to be a TMOP untangle metric:

$$f(\mu) = \{ |[1 - \epsilon] - \mu| - ([1 - \epsilon] - \mu) \}^2 \tag{3.5}$$

with $0 < \epsilon$ a number that is small compared to 1. It is easy to see that $\tilde{f}(T) \geq 0$. Further, $\tilde{f}(T) = 0$ if and only if $\mu \leq 1 - \epsilon < 1$. Therefore, if $\tilde{f} = 0$, then $\tau > 0$ (note the similarity with μ_β).

Propositions 3.2.1 and 3.2.2 thus allow us to create two new metrics having an increased potential to untangle meshes:

$$f_{SzU} = \{ |[1 - \epsilon] - \mu_{Sz}| - ([1 - \epsilon] - \mu_{Sz}) \}^2 \quad (3.6)$$

$$f_{ShSzU} = \{ |[1 - \epsilon] - \mu_{ShSz}| - ([1 - \epsilon] - \mu_{ShSz}) \}^2 \quad (3.7)$$

Note that $f_{SzU} = 0$ gives $\mu_{Sz} < 1$ and, therefore, $0 < \tau < 2$. Thus the metric tends to both untangle and to encourage $\tau = 1$. In contrast, the Untangle-Beta metric only tends to untangle. Similarly, the second metric should tend to untangle and to encourage well-shaped elements. Proposition 3.2.3 shows that the use of the Shape metric μ_{Sh} in $f(\mu)$ would be unproductive.

3.4. Numerical Experiments with the New Untangle Metrics. First, the SizeUntangle quality metric (f_{SzU}) was applied to the four tangled test meshes, with a constant value of epsilon = 0.01. The optimized meshes are shown in Figure 3.3. Not surprisingly, the results are similar to the results obtained by the Size metric shown in Figure 3.1. Results for the Hole-in-Square differ the most, with the SizeUntangle results looking less close to being inverted. In fact, the SizeUntangle metric was able to untangle all four meshes.

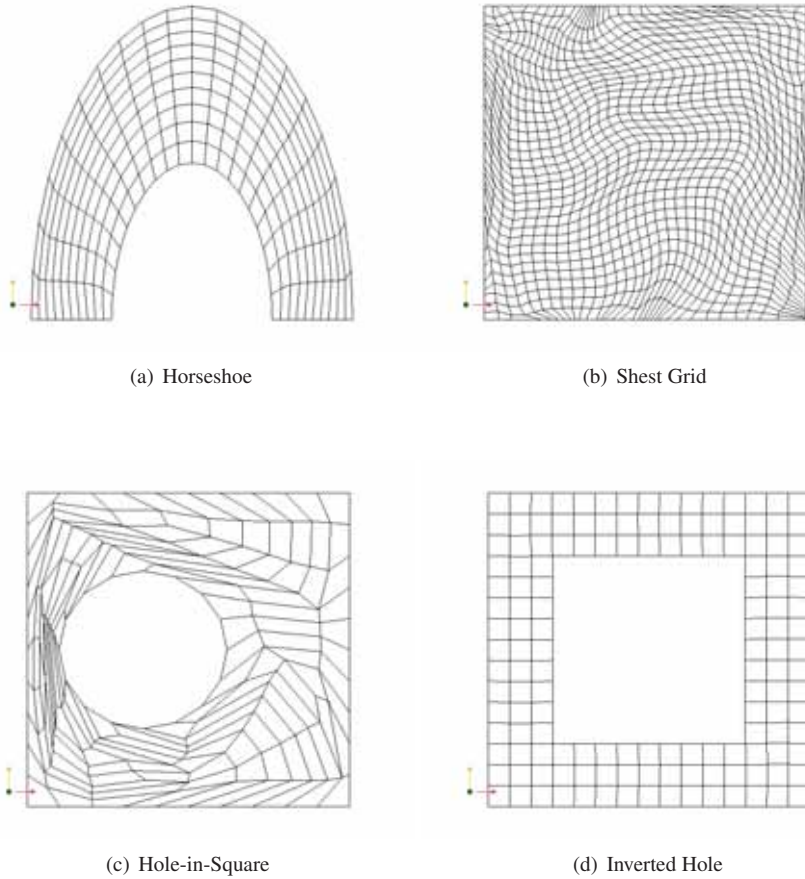


FIG. 3.3. All four meshes after optimization with f_{SzU} .

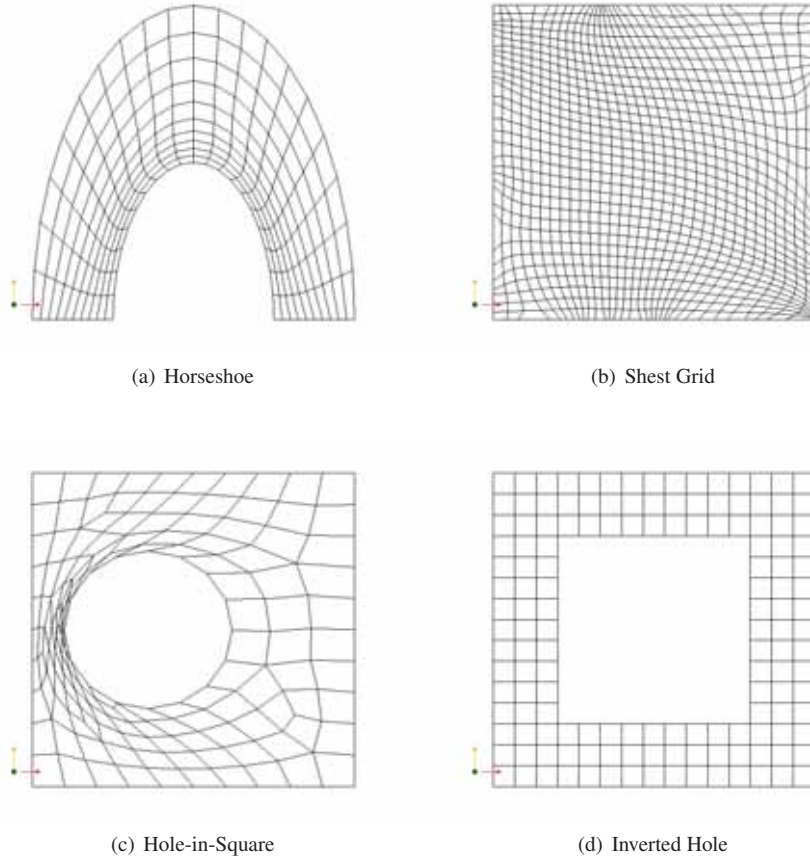


FIG. 3.4. All four meshes after optimization with f_{ShSzU} .

In Figure 3.4, the optimal meshes resulting from the ShapeSizeUntangle metric (f_{ShSzU}) are shown. These meshes look very similar to those obtained by applying the μ_{ShSz} metric (Figure 3.2). The one noticeable difference, as the following data will further detail, is that our new metric reduced the Hole-in-Square mesh to 4 of its original 23 inverted elements, while the ShapeSize metric only obtained a result of 12. Overall, due to the untangling of the Hole-in-Square mesh, the SizeUntangle metric performed better - as an untangler - than the ShapeSizeUntangle metric. As was noted in Section 3.1, Size was able to reduce Hole-in-Square to 6 inverted elements, while ShapeSize only managed 12. Perhaps this provides insight as to why the SizeUntangle metric was able to untangle the mesh when the Shape-SizeUntangle metric could not.

Several items to note from Table 3.1: The Shape metric performed the same, in terms of untangling, as Laplace smoothing. Comparing the results of the two μ vs. their $f(\mu)$ counterparts, we see that the latter did prove somewhat more effective in untangling meshes. The original μ_β metrics remains highly competitive. Better shaped elements are produced by the SizeShape and SizeShapeUntangle metrics when compared to the Size and SizeUntangle metrics, although the latter are perhaps superior in terms of ability to untangle.

For completeness we also optimized the four meshes using $f_{ShUntangle}$, even though this is not supported by a Proposition similar to 3.1.1 and 3.1.2. Results were the same as for μ_{Sh}

TABLE 3.1

Summary of Whether a Given Method Was Able to Untangle a Given Mesh
T means 'not able to untangle', *U* means 'able to untangle'

	Horseshoe	Shest	Hole-in-Sq	Inv. Hole	#T
Initial Mesh	T	T	T	T	4
Laplace	T	U	T	U	2
μ_β	U	U	U	U	0
μ_{Sz}	U	U	T	U	1
f_{SzU}	U	U	U	U	0
μ_{ShSz}	U	U	T	U	1
f_{ShSzU}	U	U	T	U	1
μ_{Sh}	T	U	T	U	2
#T	2	0	5	0	

in terms of ability to untangle.

Table 3.2 gives the values of τ_{min} resulting from the various optimizations. As one can see, the Hole-in-Square has the smallest τ_{min} values, and shows the relative success of the various methods in untangling it.

TABLE 3.2

Minimum τ value for each mesh before and after optimization with the TMOP metrics.

	Horseshoe	Shest	Hole-in-Square	Inverted Hole
Initial	-0.178098	-0.315243	-0.957579	-1.47294
μ_{Sz}	0.804821	0.253687	-0.188222	0.672569
f_{SzU}	0.790504	0.270406	0.0570835	0.717553
μ_{ShSz}	0.354348	0.192078	-0.901212	0.736087
f_{ShSzU}	0.295828	0.169121	-0.312364	0.735972
μ_{Sh}	-0.178098	0.193576	-0.957579	0.735989

TABLE 3.3

Initial Mesh (I) and Optimized Mesh (O)

	Number of Inverted Elements							
	Horseshoe		Shest		HoleSq		Inv Hole	
	I	O	I	O	I	O	I	O
Size	12	0	9	0	23	6	40	0
SizeUntangle	12	0	9	0	23	0	40	0
ShapeSize	12	0	9	0	23	12	40	0
ShapeSizeUntangle	12	0	9	0	23	4	40	0
Shape	12	12	9	0	23	23	40	0

4. Conclusions. A new mesh optimization strategy for untangling 2D meshes, based on node-movement, has been investigated. The strategy relies on a set of Propositions which show that, for certain quality metrics $0 \leq \mu < \infty$ within the Target-matrix paradigm, if $\mu < 1$ then the local area is positive. The Propositions were exploited in devising a new strategy for mesh untangling that can also incorporate other goals such as Size or ShapeSize improvement (i.e., the new metrics are not pure untanglers). In this way, the new strategy is similar in purpose (but not in approach) to the method in [2]. Numerical results showed that the new

TABLE 3.4
Initial Mesh (O) and Optimized Mesh (O)

	Number of Invalid Points							
	Horseshoe		Shest		HoleSq		Inv Hole	
	I	O	I	O	I	O	I	O
Size	44	0	9	0	70	6	156	0
SizeUntangle	44	0	9	0	70	0	156	0
ShapeSize	44	0	9	0	70	31	156	0
ShapeSizeUntangle	44	0	9	0	70	4	156	0
Shape	44	44	9	0	70	70	156	0

metrics (especially SizeUntangle) performed reasonably well when compared to the authors' previous Untangle-Beta metric (additional tangled meshes are needed to confirm this conclusion). Significantly, the new metrics do not require that one specify a parameter similar to the awkward β parameter.⁵ It is unfortunate that a useful Proposition for the Shape metric does not exist since if it did, one could avoid having to construct proper values for Λ in the Target-matrices, because the Shape metric is Size-invariant. Future work within this new strategy might include more sophisticated models for the Size parameter Λ in the Target-matrix to allow untangling of meshes having heterogeneously-sized elements. Work on extending this approach to 3D meshes has already commenced. Finally, like all the other untangling algorithms appearing in the literature, the new metrics do not guarantee an untangled result.

Acknowledgments The initial impetus for this paper is due to Evan van der Zee, who first proved that the ShapeSizeOrientation metric, $|T - I|^2$, obeys a Proposition very similar to those given in Section 3.2. This metric was not used in this study because it requires one to include Orientation information within the Target-matrix construction step, and it is not obvious what that information should be in mesh untangling.

REFERENCES

- [1] P. BARRERA-SANCHEZ AND J. TINOCO-RUIZ, *Smooth and Complex Grid Generation over General Plane Regions*, Mathematics and Computers in Simulation, 46 (1998), pp. 87–102.
- [2] J. ESCOBAR, E. RODRIGUEZ, R. MONTENEGRO, G. MONTERO, AND J. GONZALEZ-YUSTE, *Simultaneous Untangling and Smoothing of Tetrahedral Meshes*, Computer Methods in Applied Mathematics and Engineering, 192 (2003), pp. 2775–2787.
- [3] L. FREITAG AND P. PLASSMANN, *Local Optimization-Based Simplicial Mesh Untangling and Improvement*, Intl. Journal of Numerical Methods in Engineering, 49, pp. 109–125.
- [4] ———, *Local Optimization-Based Untangling Algorithms for Quadrilateral Meshes*, (2001).
- [5] P. KNUPP, *Local 3d metrics for mesh optimization in the target-matrix paradigm*, manuscript.
- [6] ———, *Hexahedral Mesh Untangling and Algebraic Mesh Quality Metrics*, 9th International Meshing Roundtable, (2000), pp. 173–183.
- [7] ———, *Hexahedral and Tetrahedral Mesh Untangling*, Engineering with Computers, 17 (2001), pp. 261–268.
- [8] ———, *Local 2d Metrics for Mesh Optimization in the Target-matrix Paradigm*, SAND2006-7382J, Sandia National Laboratories, (2006).
- [9] ———, *Introducing the target-matrix paradigm for mesh optimization via node-movement*, Proceedings of the 19th International Meshing Roundtable, (2010).
- [10] P. VACHAL, R. V. GARIMELLA, AND M. J. SHASHKOV, *Mesh Untangling*, Los Alamos National Laboratory - Summer report, (2002).
- [11] A. WINSLOW, *Numerical solution of the quasilinear poisson equations in a nonuniform triangle mesh*, J. Comp. Phys., 2 (1967), pp. 149–172.

⁵The parameter ϵ in $f(\mu)$ is easier to choose than β since it only needs to be small compared to 1, whereas β must be small compared to τ . Moreover, Λ can be constant over the mesh even if the mesh is Size-heterogeneous.

STATIC VERTEX REORDERING SCHEMES FOR LOCAL MESH QUALITY IMPROVEMENT

JEONGHYUNG PARK^{||}, PATRICK KNUPP^{**}, AND SUZANNE M. SHONTZ^{††}

Abstract. Numerical experiments were conducted to investigate how timings in a local mesh Laplacian-smoothing algorithm vary as a function of the ordering of the vertices (and patches) in the mesh. Timings varied by roughly a factor of 2, depending which of the 20 orderings was used. Sensitivity of these results as a function of mesh size, mesh quality, and termination criteria was investigated. No particular ordering was best in all situations, although certain orderings appear to be viable candidates for an all-purpose ordering.

1. Introduction. The mesh and its quality can greatly impact: (1) the accuracy of the PDE solution and (2) the conditioning, stability, and efficiency of the associated PDE solver. [11, 7]. The quality of the mesh can be improved by adaptivity [1, 6], smoothing [8, 2], or swapping [4, 5]. Mathematically rigorous mesh smoothing is performed via optimization of an objective function which measures the mesh quality. In mesh smoothing, vertex movement strategies are applied in order to change the mesh vertex coordinates, while maintaining the connectivity of the initial mesh. In this paper, we focus on static vertex reordering schemes within the context of local mesh optimization in order to improve mesh quality.

1.1. Related Literature. Shontz and Knupp previously performed research on vertex reordering schemes within the context of local mesh optimization [12]. Key findings from their paper [12] are as follows:

1. Vertex reorderings were most helpful when the initial mesh is far from optimal.
2. Vertex reorderings were most helpful when the initial vertex ordering is poor.
3. Dynamic vertex reordering schemes were too expensive relative to static vertex reordering schemes.

These results form the motivation for the current study on static vertex reordering schemes within the context of local mesh optimization.

Other researchers have also performed research vertex reordering within the context of mesh smoothing performed via optimization. However, it should be noted that the papers to be discussed below develop results on vertex reordering schemes applied to improving the efficiency of the mesh optimization via improvements in the cache performance on a specific computer architecture. In contrast, we are developing general-purpose techniques, i.e., schemes which are independent of the computer architecture.

For example, Munson [9] and Munson and Hovland [10] developed a Feasible Newton mesh optimization algorithm and benchmark. Their algorithm and benchmark employed both data and iteration reordering in order to improve cache performance. One finding of their research was that reordering of the input data can increase or decrease the number of iterations taken by the inexact Newton method and can affect its success or failure [9]. The reordering applied was a reordering of the vertices and elements in the mesh by applying a breadth-first search and reversing the order in which the vertices were visited. When data and iteration ordering were performed on the relevant hypergraphs, the reorderings were found to significantly decrease the number of cache misses in all phases of code execution and resulted in significantly faster code [10].

Strout *et al.* [13] investigated six data and iteration reordering schemes and applied them to tetrahedral meshes before they were optimized using FeasNewt. With each reordering

^{||}The Pennsylvania State University, jxp975@cse.psu.edu

^{**}Sandia National Laboratories, pknupp@sandia.gov

^{††}The Pennsylvania State University, shontz@cse.psu.edu

scheme, the vertices and elements of the mesh were reordered using a hypergraph model. The six orderings considered in their paper were: the (1) null ordering, (2) Hyper-BFS (a breadth first search on hypergraph), (3) Hyper-CPack (consecutive packing on hypergraph), (4) Hyper-Part (hypergraph partitioning), (5) HierBFS (Hierarchical BFS) and (6) HierCPack (Hierarchical consecutive packing). They saw up to 40% better performance than the original ordering. Their results show that hierarchical data reordering improves over the performance of local reordering strategies. Performance metrics of interest were overall execution time as well as various types of cache misses (L1, L2, and TLB). This paper lists two possibilities for future research: (1) show that a particular ordering will never slow down a particular algorithm if applied and (2) automatic determination of the best ordering.

1.2. Vertex Reordering Study. For the current study, we introduce vertex reordering of the interior vertices within the context of the local Laplacian smoothing procedure. Many other methods for mesh smoothing exist, but Laplacian was selected for this study because it is simple and well-understood. Two classes of reordering techniques were suggested in [12]: static vertex reordering and dynamic vertex reordering. In the static case, the initial ordering of the vertex list provided to the mesh optimization algorithm is reordered by using some criteria. The reordered list is used and kept fixed during the optimization. In the dynamic case, the reordered list is updated at the end of each iteration within the optimization, thus creating a sequence of vertex lists. In this paper, we focus on static reordering of the vertices. The quality of the mesh is measured according to a normalized Laplacian mesh quality metric. The principal goal of this study is to determine the key variables that affect the success of the reordering methods as determined by the performance of the underlying Laplacian mesh smoothing routine.

We specify the pseudocode for the local mesh optimization procedure which identifies when vertex reordering is performed and also the details of the timing. Further details on the timing metrics of interest will be described in the Numerical Experiments section.

```

1. Initialization: vertex list, tolerance values,
    termination criterion, objective function,
    control parameters.
    -- CPU Timer_1
2. While tolerance not satisfied and iteration
    count not exceeded
    -- CPU Timer_2
    Loop over list of free vertices
        - update free vertex coordinates
          (via optimization)
    End loop over free vertices
    -- CPU Timer_3
    Reorder vertex list on first iteration
    -- CPU Timer_4
End while loop
-- CPU Timer_5

```

1.3. Vertex Reordering Study Design. For our initial study, we fixed the following variables: quality metric, optimization template (linear averaging), optimization solver (Laplace smoothing), sorter (Quicksort), mesh connectivity, serial computation, architecture and vary the mesh size, element type, element heterogeneity, element isotropy, solver convergence tolerance, and vertex reordering scheme. Mesquite, the Mesh Quality Improvement Toolkit [3], was used for the study.

2. Mesh Quality Improvement Problem. Let q denote the objective function as computed by the Laplacian metric. Then $q = \frac{1}{n} \sum_{i=1}^n l_i^2$, where l_i is the length of the i^{th} interior edge and n is the total number of interior edges in the mesh. Define the normalized Laplace objective function by $q_N = \frac{q_{final} - q_{cur}}{q_{final} - q_{init}}$, where q_{init} is the initial value of the objective function, q_{cur} is the current value of the objective function during the optimization, and q_{final} is the value of the objective function when the optimization is converged. The range of the normalized objective function is from 0 to 1, with 0 being the optimal value. Minimizing q_N is equivalent to minimizing q_{cur} . The value of q_{final} , which establishes convergence, was based on observing when q_{cur} became nearly constant as the optimization progressed.

3. Vertex Reordering Schemes. We apply vertex reordering in the above mesh optimization algorithm. Most of the schemes reorder the vertices at the beginning of the optimization process. However, in the case of the vertex movement-related schemes, the vertex reordering is executed after one optimization step is complete so that the distance between the previous and the current vertex coordinates can be computed. In this paper, static vertex reordering is considered, i.e., reordering is performed only once. The twenty vertex reordering schemes explored are as follows (taken from [12]):

- **Scheme (N): Null ordering.** Do not reorder the vertex list.
- **Scheme (R): Random Ordering.** Generate a random integer from 1 to N, with N being the total number of local patches in the mesh. Place the vertex assigned the value 1 first in the list and N last.
- **Scheme (WQP): Ordering by Worst Quality Patch.** Evaluate objective function on a local patch to measure local patch quality. Sort by putting the worst quality patch first in the list, and so on.
- **Scheme (GAVM): Ordering by Greatest Absolute Vertex Movement.** Evaluate the absolute distance moved by the free vertex in the local patch before and after the local optimization. Sort by putting the patch with the greatest absolute distance moved first in the list, and so on.
- **Scheme (GRVM): Ordering by Greatest Relative Vertex Movement.** Evaluate the relative distance moved by the free vertex in the local patch before and after the local optimization. Relative distance was measured by dividing the absolute distance-moved by the initial absolute distance-moved; the normalizing quantity thus varies from patch to patch. Sort by putting the patch with the greatest relative distance moved first in the list, and so on.
- **Scheme (LNG): Ordering by Largest Norm of Gradient.** Evaluate the ℓ_2 -norm of the local gradient of the objective function. Sort by putting the largest norm first in the list, and so on.
- **Schemes (D-WQP, D-GAVM, D-GRVM, and D-LNG): Distance schemes.** Created by measuring the distance of the free vertex in each patch from the position of the free vertex which had the (i) worst patch quality, (ii) greatest average vertex movement, (iii) greatest relative vertex movement, or (iv) largest norm of the gradient, respectively. Then sort by putting the patch with the smallest distance first in the list, and so on.
- **Schemes (-N, -R, -WQP, -D-WQP, -GAVM, -D-GAVM, -GRVM, -D-GRVM, -LNG, -D-LNG): Reverse schemes.** For each of the schemes above there is a corresponding ordering scheme produced by reversing the ordering. For example, scheme -WQP orders the vertex patches from best quality first to worst quality last.

4. Numerical Experiments. To determine the efficiency of the static vertex reordering schemes, four numerical experiments were designed. Each experiment was designed to illus-

trate the impact of the following factors on the efficiencies of the mesh optimization algorithm when used in conjunction with the various vertex reordering schemes: the final mesh quality desired, the mesh size, the initial mesh quality, and the initial vertex ordering. The following subsections will explain these experiments and the associated results in more detail.

A tetrahedral hook geometry was used to create the initial meshes for the experiments (Figure 4.1).

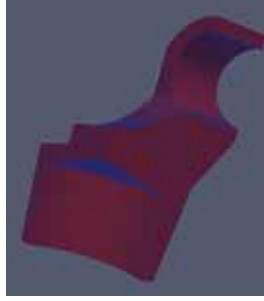


FIG. 4.1. *The tetrahedral hook geometry used to create the initial meshes for the experiments*

Our mesh quality optimization procedure minimized the objective function q , which corresponds to the overall mesh quality. To evaluate the normalized metric q_{NLM} , timing results are reported for when the mesh optimization algorithm reached $q_{final} = 2.237975$.

The timing results for each scheme were unstable due to the background tasks of the machine. Multiple runs were required in order to obtain stable relative rankings of the reordering schemes. Because the average time results after five runs were close to stable, five runs of each scheme for each experiment were performed, and their times were averaged for each scheme. The average total time (over the 5 runs) of a given scheme is defined as TT and all timing results for each experiment used the TT values of each reordering scheme.

To characterize the variation in total times over the 20 schemes, compared to the total time of a typical scheme, results are reported as $\frac{TT_{max}+TT_{min}}{2} \pm \frac{TT_{max}-TT_{min}}{2}$, where TT_{min} and TT_{max} are the minimum and maximum total times of all the reordering schemes. Also, the percent variation in the total time was computed by dividing $\frac{TT_{max}-TT_{min}}{2}$ by $\frac{TT_{max}+TT_{min}}{2}$.

To effectively compare the timing results of each scheme, we defined β as follows:

$$\beta = \frac{TT - TT_{avg}}{TT_{avg}}, \quad (4.1)$$

where TT_{avg} is the average total time of all the reordering schemes. Note that if $\beta > 0$ for a particular scheme, the scheme is slower than average. Similarly, if $\beta < 0$, the scheme is better than average. The deviation in β used for each experiment represented the value farthest from $\beta = 0$.

All four experiments were performed with a multicore machine equipped with two processors: a 4 quad-core AMD Shanghai processor running at 2.7 GHz with 96 GB of RAM and a 2 quad-core Intel Nehalem processor running at 2.66 GHz with 24 GB of RAM. The performance of each run was very sensitive, because the machine was multicore; thus, there was memory contention between processors. Although five runs of each scheme for each experiment were performed to obtain stable timing results, the timing results might be different if the experiment environment is changed.

4.1. Experiment 1: Sensitivity to Outer Termination Criterion Tightness. The purpose of this experiment was to observe the behavior of the mesh optimization algorithm when used

in conjunction with the various reordering schemes when different outer termination values were set. The mesh optimization algorithm was terminated based on the normalized Laplacian metric (NLM). In this experiment, four NLM values were considered, i.e., 0.75, 0.5, 0.25, and 0, as termination values. The first three cases were used to observe the behavior of each scheme when a relatively inaccurate solution is pursued, whereas the last case was used to observe the behavior when an accurate solution is desired. After the mesh quality reached the desired NLM value, the mesh optimization algorithm terminated, and the corresponding timing information was recorded.

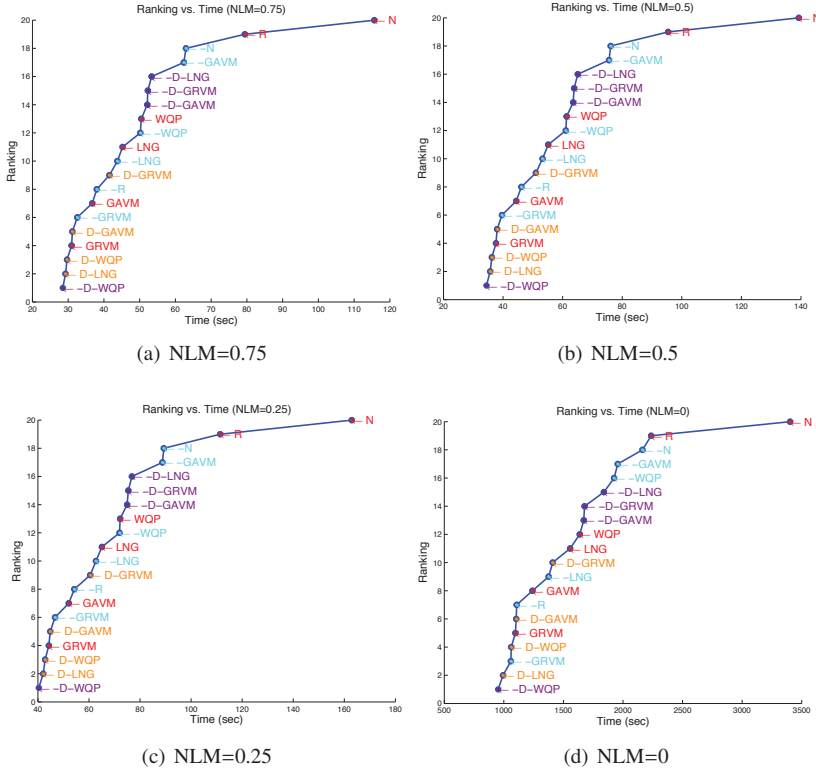


FIG. 4.2. Ranking vs. total time for the mesh optimization algorithm when used in conjunction with the various vertex reordering schemes with various NLM values in Experiment 1.

Figure 4.2 shows the ranking results for each scheme in Experiment 1. The overall scheme rankings were relatively stable although the rankings for some schemes, such as GRVM, -LNG, and -D-GAVM, changed slightly for $NLM=0$. The mesh optimization algorithm converged faster, independent of the NLM value, when most of the schemes other than N were used for reordering. Thus, vertex reordering improves the efficiency of the mesh optimization algorithm.

In this experiment, the particular choice of reordering scheme is important. It was observed that the time difference between the best and the worst ranked schemes was reasonably large. The timing results were: 72.07 ± 43.57 seconds for $NLM=0.75$, 86.86 ± 52.42 seconds for $NLM=0.5$, 101.64 ± 61.27 seconds for $NLM=0.25$, and 2177.24 ± 1225.05 seconds for $NLM=0$. That is, for $NLM=0.75$, 0.5, 0.25, and 0, the percent variations in total time were 60.45%, 60.26%, 60.29%, and 56.27%, respectively. The reason for the large difference in the variation in the total time was the relatively poor performance

of the scheme N. The total time of scheme N (3100 seconds) was approximately twice that of the other reordering schemes (1520 seconds). Excluding this outlier, the percent variation in the total time was 35% which was still significant. Therefore, appropriate selection of the reordering scheme was required to converge quickly.

Similarly, the deviation in β (4.1) was relatively noticeable. Figure 4.3 shows the values of β for each scheme in Experiment 1.

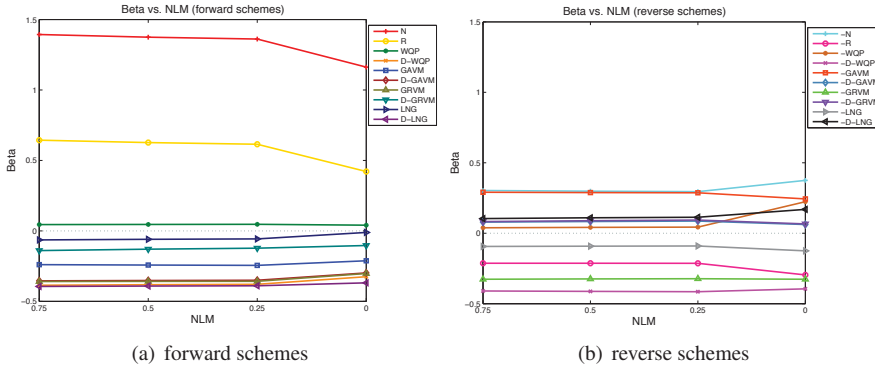


FIG. 4.3. β vs. NLM for the 20 vertex reordering schemes in Experiment 1. Results for the 10 forward schemes: N , R , WQP , $D-WQP$, $GAVM$, $D-GAVM$, $GRVM$, $D-GRVM$, LNG , and $D-LNG$ are shown in (a). The results for the 10 reverse schemes: $-N$, $-R$, $-WQP$, $-D-WQP$, $-GAVM$, $-D-GAVM$, $-GRVM$, $-D-GRVM$, $-LNG$, and $-D-LNG$ are shown in (b).

In all cases, the deviations in β were greater than 1. In particular, the deviations were: 1.39 for $NLM=0.75$, 1.37 for $NLM=0.5$, 1.36 for $NLM=0.25$, and 1.16 for $NLM=0$. This was because the value of β for scheme N was always greater than 1. In addition, the total time for scheme N was twice that of the other schemes. Again, this demonstrated the importance of the choice of vertex reordering scheme.

The $-D-WQP$ scheme ranked the best in this experiment for all NLM values. $GRVM$, $D-LNG$, and $D-GAVM$ schemes also showed relatively good performance in this experiment. Although small changes occurred in the relative rankings as the NLM value decreased, the total time to converge was reduced through adequate choice of reordering schemes, especially the $-D-WQP$ scheme for this experiment.

4.2. Experiment 2: Mesh Size. The purpose of this experiment was to determine the effect that mesh size has on the efficiency of the mesh optimization algorithm when used in conjunction with various vertex reordering schemes. In this experiment, five different sizes of meshes were used: 50K, 75K, 100K, 125K, and 150K. Appropriate mesh sizes were determined through a preliminary experiment. Figure 4.4 shows the ranking vs. total time results for each of the five mesh sizes in this experiment.

The rankings for each scheme were very sensitive to an increase in mesh size. Schemes WQP and $-WQP$ ranked very highly until the mesh size increased to 150K. However, when the mesh size reached 150K, these schemes ranked the worst. The time difference between the best and the worst ranked schemes was relatively noticeable. For mesh sizes 50K, 75K, 100K, 125K, and 150K, the timing results were: 72 ± 20 seconds, 170 ± 78 seconds, 600 ± 305 seconds, 1622 ± 853 seconds, and 2532 ± 804 seconds, respectively. The percent variations in the total time were: 27% for 50K, 47% for 75K, 51% for 100K, 53% for 125K, and 32% for 150K. Excluding the outlier, i.e., the 150K mesh, the time variations increased as the mesh size increased. In this experiment, quality-related reordering schemes WQP and

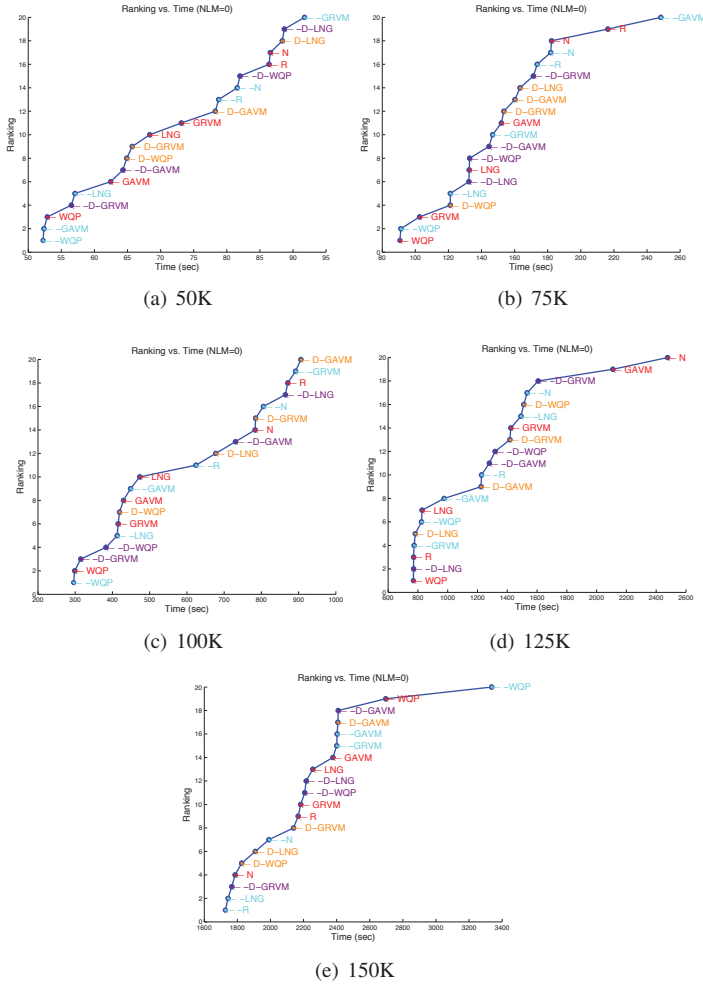


FIG. 4.4. Ranking vs. total time for the mesh optimization algorithm when used in conjunction with the various vertex reordering schemes with various mesh sizes in Experiment 2.

-WQP showed good performance when combined with the mesh optimization algorithm. The deviation in β (4.1) also showed the sensitivity of the ranking results as the mesh size increased (Figure 4.5).

The deviations in β were: 0.28, 0.65, 0.53, 0.97, and 0.51 for mesh sizes 50K through 150K, respectively. Although the deviation in β for mesh size 50K was small compared to the deviation for the other mesh sizes, the overall β deviation from one mesh size to the next was discernible.

4.3. Experiment 3: Sensitivity to Initial Mesh Vertex Coordinates. The purpose of this experiment was to investigate the behavior of the mesh optimization algorithm when used in conjunction with various reordering schemes when various meshes with different initial mesh qualities were used. The input meshes differed from the original mesh used in Experiment 1 in their vertex coordinates. Four meshes of various qualities were obtained by randomly

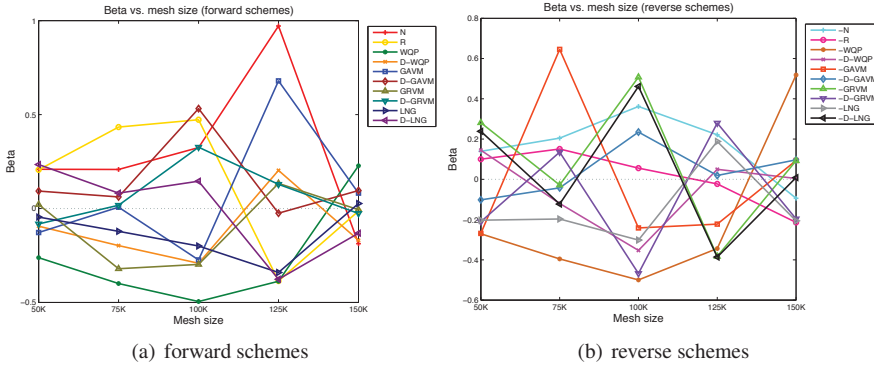


FIG. 4.5. β vs. mesh size for the 20 vertex reordering schemes in Experiment 2. Results for the 10 forward schemes N , R , WQP , $D-WQP$, $GAVM$, $D-GAVM$, $GRVM$, $D-GRVM$, LNG , and $D-LNG$ are shown in (a). The results for the 10 reverse schemes $-N$, $-R$, $-WQP$, $-D-WQP$, $-GAVM$, $-D-GAVM$, $-GRVM$, $-D-GRVM$, $-LNG$, and $-D-LNG$ are shown in (b).

perturbing the vertex coordinates of the original mesh as follows:

$$X_{new} = X_{orig} + fL\mathbf{u}, \quad (4.2)$$

where \mathbf{u} is a random unit vector, L is the average of all the edge lengths in the mesh, and f is the amount of perturbation. In this paper, $f = 0, 0.1, 0.3$, and 0.9 . Note that $f = 0$ means no perturbation, and $f = 0.9$ is a large perturbation. The perturbations are applied only to the interior vertices in the mesh. Figure 4.6 shows the ranking results for each scheme in Experiment 3.

As the perturbation size increased, the mesh optimization algorithm required more time to converge for nearly all the reordering schemes. The time difference between the best and the worst ranked schemes was noticeable. The timing results were: 1940 ± 1148 seconds for $f = 0$, 1440 ± 712 seconds for $f = 0.1$, 2013 ± 985 seconds for $f = 0.3$, and 3775 ± 2270 seconds for $f = 0.9$. That is, for $f = 0, 0.1, 0.3$, and 0.9 , the percent variations in total time were 59%, 50%, 49%, and 60%, respectively. Hence vertex reordering can reduce the amount of time for the mesh optimization algorithm to converge. In this experiment, schemes $D-GAVM$ for $f = 0$, WQP for $f = 0.1$, $-N$ for $f = 0.3$, and $-D-GRVM$ for $f = 0.9$ ranked the best. The mesh optimization algorithms with these schemes took less than half of the total time for the mesh optimization algorithm with scheme N .

The plots of the values of β vs. perturbation size for each scheme are shown in Figure 4.7. The β deviations (4.1) were: 0.85 for $f = 0$, 0.51 for $f = 0.1$, 0.74 for $f = 0.3$, and 0.7 for $f = 0.9$. The values of β for each scheme deviate from $\beta = 0$ more than 0.5. The ranking of each scheme was very sensitive with respect to the different perturbation size.

4.4. Experiment 4: Sensitivity to Initial Mesh Vertex Ordering. The purpose of this experiment was to explore the effect that the initial vertex ordering had on the efficiency of the mesh optimization algorithm when used in conjunction with the vertex reordering schemes. The input meshes differed from the original mesh used in Experiment 1 in their initial vertex orderings. Four different reordering schemes, i.e., LNG , R , N , and $D-LNG$ were applied to the original mesh to obtain four initial meshes. The results for this experiment are shown in Figure 4.8.

When schemes LNG and R were used for creating the initial vertex ordering, the mesh optimization algorithm with scheme $-D-WQP$ required the least amount of time to complete.

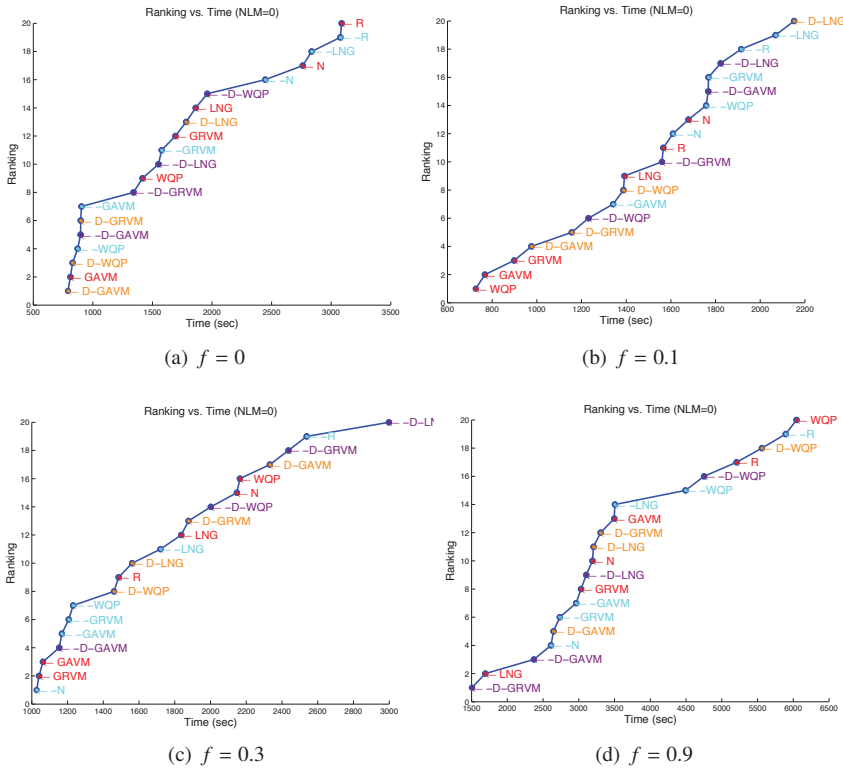


FIG. 4.6. Ranking vs. total time for the mesh optimization algorithm when used in conjunction with the various vertex reordering schemes with various perturbation sizes in Experiment 3.

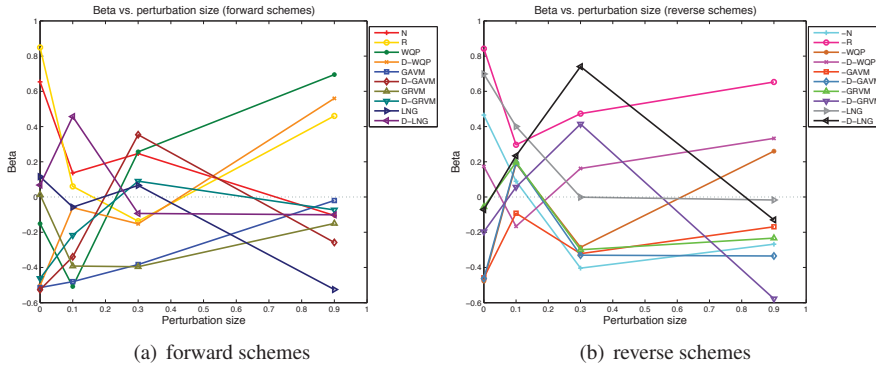


FIG. 4.7. β vs. perturbation size for the 20 vertex reordering schemes in Experiment 3. Results for the 10 forward schemes N, R, WQP, D-WQP, GAVM, D-GAVM, GRVM, D-GRVM, LNG, and D-LNG are shown in (a). The results for 10 reverse schemes -N, -R, -WQP, -D-WQP, -GAVM, -D-GAVM, -GRVM, -D-GRVM, -LNG, and -D-LNG are shown in (b).

However, the rankings for all of the schemes were very sensitive with respect to the initial vertex ordering of the initial mesh. In this experiment, the time difference between the best and the worst ranked schemes was relatively large, i.e., over 2000 seconds. For the initial vertex orderings created using schemes LNG, R, N, and D-LNG, the timing results were:

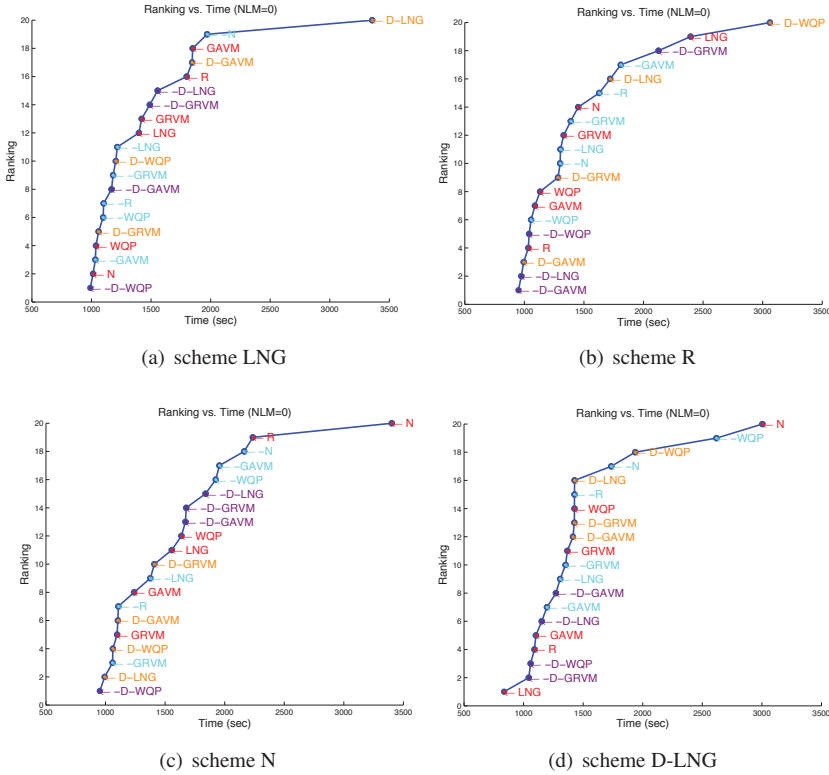


FIG. 4.8. Ranking vs. total time for the mesh optimization algorithm when used in conjunction with the various vertex reordering schemes applied to meshes with different initial vertex orderings in Experiment 4.

2173 \pm 1183 seconds, 2006 \pm 1055 seconds, 2177 \pm 1255 seconds, and 1920 \pm 1084 seconds, respectively. That is, the percent variations in total time were: 54% for the LNG initial ordering, 53% for the R initial ordering, 56% for the N initial ordering, and 56% for the D-LNG initial ordering. Figure 4.9 also shows the sensitivity of the ranking results for this experiment.

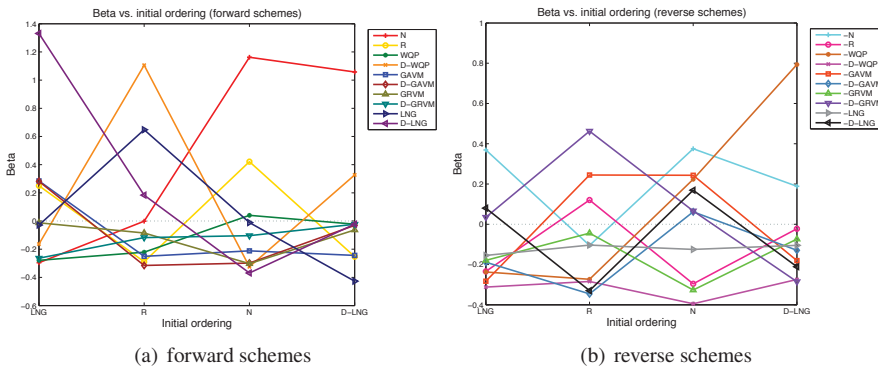


FIG. 4.9. β vs. the initial vertex ordering for the 20 vertex reordering schemes in Experiment 4. Results for the 10 forward schemes N, R, WQP, D-WQP, GAVM, D-GAVM, GRVM, D-GRVM, LNG, and D-LNG are shown in (a). The results for the 10 reverse schemes -N, -R, -WQP, -D-WQP, -GAVM, -D-GAVM, -GRVM, -D-GRVM, -LNG, and -D-LNG are shown in (b).

The deviation in β (4.1) as a function of initial vertex ordering was very significant. The maximum value of β was: 1.33 for the LNG initial ordering, 1.11 for the R initial ordering, 1.16 for the N initial ordering, and 1.06 for the D-LNG initial ordering. Even scheme N had the maximum value of β for the N and D-LNG initial vertex orderings.

5. Conclusions. Numerical experiments were conducted on a local-patch Laplacian-smoothing algorithm to assess the effect of the ordering of the vertices in a tetrahedral mesh on the CPU time to solution. Twenty orderings, based on mesh quality, gradients, and movement, were created from the initial mesh for use in a set of four experiments. The experiments were designed to explore the sensitivity of the timing results to (1) whether or not an 'accurate' solution was found, (2) the mesh size, (3) the 'distance' between the initial and optimal meshes, and (4) the initial vertex orderings. The results reveal that the time-to-solution within each experiment varied significantly (about 50%) over the twenty orderings, showing that it does matter which ordering is used. Scheme N results in the initial ordering that is supplied by the mesh generator; that scheme ranked 20th (worst) in Experiment 1, no greater than 14th in Experiment 2 (except for the 150K mesh), and no greater than 10th in Experiment 3, leading to the conclusion that re-ordering is often better than using the initial ordering from the mesh generator.

In the previous paper [12], one of the conclusions was that the farther the initial guess is from optimal, the larger the variation in the total time of the mesh optimization algorithm, and thus the more significant the choice of reordering scheme becomes. A similar trend was obtained in the results of Experiment 3. When the perturbation size $f = 0.9$ was applied, the timing results showed more variation than other smaller perturbation size cases. However, one could not see that vertex reorderings were most useful when the initial vertex ordering was poor through Experiment 4.

In Experiment 1, the rankings of the schemes were remarkably stable as the 'accuracy' of the solution was varied, with -D-WQP giving the smallest timings. Rankings were notably less stable in the other experiments. Scheme WQP ranked well in the mesh-size experiments (from 50K to 125K vertices); however, scheme -D-WQP, which ranked well in Experiment 1, did not stand out in Experiment 2. Scheme GAVM ranked well in Experiment 3 for $f < 0.9$. Scheme -D-WQP again ranked well in Experiment 4. Consequently, although no particular scheme did well under all circumstances, the after-mentioned schemes might be worth considering if static re-ordering is to be applied prior to local-patch Laplacian-smoothing.

6. Future Work. Reducing the variability in the multicore timing results is planned in the future. Furthermore, additional experiments will be performed to examine the performance of the reordering schemes within other contexts such as aspect ratio and boundary vertex reordering.

7. Acknowledgement. This work was supported in part through instrumentation funded by the National Science Foundation through grant OCI-0821527. Also, this work was funded in part by NSF grant CNS 0720749 and an Institute for CyberScience grant from The Pennsylvania State University.

REFERENCES

- [1] R. BANK AND R. SMITH, *Mesh smoothing using a posteriori error estimates*, SIAM J. Numer. Anal., 34 (1997), pp. 979–997.
- [2] L. BRANETS AND G. CAREY, *Smoothing and adaptive redistribution for grids with irregular valence and hanging nodes*, in Proc. of the 13th International Meshing Roundtable, 2004, pp. 333–344.

- [3] M. BREWER, L. A. FREITAG, P. KNUPP, T. LEURENT, AND D. MELANDER, *The Mesquite Mesh Quality Improvement Toolkit*, in Proceedings of the 12th International Meshing Roundtable, Sandia National Laboratories, 2003, pp. 239–250.
- [4] L. FREITAG AND C. OLLIVIER-GOOCH, *Tetrahedral mesh improvement using swapping and smoothing*, Int. J. Num. Meth. Engr., 40 (1997), pp. 3979–4002.
- [5] B. KLINGNER AND J. SHEWCHUK, *Aggressive tetrahedral mesh improvement*, Springer-Verlag, 2008, pp. 3–23.
- [6] J. LAGUE AND F. HECHT, *Optimal mesh for P_1 interpolation in H^1 seminorm*, in Proc. of the 15th international Meshing Roundtable, Springer-Verlag, 2006, pp. 259–270.
- [7] I. M. BATDORF, *Computational study of the effect of unstructured mesh quality on solution efficiency*, in Proc. of the 13th Annual AIAA Computational Fluid Dynamics Conference, 1997.
- [8] R. MONTENEGRO, J. ESCOBAR, G. MONTERO, AND E. RODRIGUEZ, *Quality improvement of surface triangulations*, in Proc. of the 14th International Meshing Roundtable, Springer-Verlag, 2005, pp. 469–484.
- [9] T. MUNSON, *Mesh shape-quality optimization using the inverse mean ratio*, Mathematical Programming, 110 (2007), pp. 561–590.
- [10] T. MUNSON AND P. HOVLAND, *The feasNewt benchmark*, in Proc. of The Second ACM SIGPLAN Workshop on Memory System Performance (MSP), pp. 23–24.
- [11] J. SHEWCHUK, *What is a good linear finite element?* Unpublished preprint, 2002.
- [12] S. SHONTZ AND P.KNUPP, *The effect of vertex reordering on 2D local mesh optimization efficiency*, in Proc. of the 17th International Meshing Roundtable, Springer, 2008, pp. 107–124.
- [13] M. STROUT, N. OSHEIM, D. ROSTRON, P. HOVLAND, AND A. POTHEN, *Evaluation of hierarchical mesh reorderings*, in Proc. of the 9th International Conference on Computational Science, pp. 540–549.

MULTIFRACTAL DIMENSIONS USING MAXIMAL SIMPLICES AND PYTHON EXTENSIONS TO TEVA-SPOT

JESSE BERWALD*, DAVID DAY†, SCOTT MITCHELL‡, AND AFRA ZOMORODIAN§

Abstract. This article briefly summarizes two projects, the first on nonlinear dynamical systems and the second on discrete optimization software.

Nonlinear dynamical systems, which are integral to a wide range of fields from biological modeling to climate studies, often exhibit chaotic behavior. The attractors of such systems give a window into the global behavior of the systems themselves. We introduce a new method for determining the multifractal spectrum and dimension of the attractor of a dynamical system. We show that the tools developed for efficient analysis in the field of computational topology are well-suited for the task of estimating the local density of points in the attractor.

Discrete optimization problems are found in a great variety of fields, such as architecture (e.g., long term site plans over many sites) to water planning and management to nuclear weapon stewardship. We describe below an extension to the TEVA-SPOT suite developed by SNL and the EPA. We focus on the Pareto front of optimal solutions to a pair of objectives and consider the near-optimal solutions dominated by these.

1. Introduction: Simplicial Measures. The discovery of *strange attractors*, such as the Lorenz attractor and other chaotic systems, illuminated an incredibly rich structure in relatively simply physical systems. Subsequent study showed that the attractors of such systems are complex enough to have a continuum of scaling exponents [6, 12, 18]. This is in contrast to the Cantor set, for instance, which has a single scaling exponent [6].

The concept of dimension can be generalized past the usual notions of length, area, and volume. Many methods have been developed to approximate the dimension of the attractor of a dynamical system. The two that we will focus on in this work are the box-counting dimension, also known as the Minkowski-Bouligand dimension, and the Hausdorff dimension, which is estimated using the partition function and methods derived from thermodynamics [6, 14, 17].

In order to approximate the dimension of an attractor computationally, one needs a way to tease out the invariant measure on the attractor from a finite set of data points [5]. These data are assumed to have been sampled from the distribution of points on the attractor (determined by the invariant measure), either numerically by running a computer simulation for a “long time” or through suitable measurement of a real physical system. Understanding the invariant measure is difficult because it often has a highly irregular distribution on the attractor and can only be studied using the finite data set.

We describe a new *simplicial measure* that gives an accurate approximation of the mass distribution on the attractor and allows one to compute the fractal dimension as well as the *multifractal spectrum* of the attractor. A simplicial measure is defined on a subset of simplices in a Vietoris-Rips simplicial complex that has been constructed from a finite data set sampled from the attractor. This subset is composed of the maximal simplices in the complex. These structures provide detailed information about the density of points in the data set, allowing one to approximate the invariant measure on the attractor itself.

1.1. Dynamical Systems. In this section we recall the definition of a dynamical system as well as that of an attractor. A dynamical system specifies a rule that describes how one state of a system evolves into another over time. We describe this situation for discrete-time systems on a topological space X , the *state space*. (For a more thorough review including

*Department of Mathematical Sciences, Montana State University, berwald@math.montana.edu

†Sandia National Laboratories, dmday@sandia.gov

‡Sandia National Laboratories, samitch@sandia.gov

§Department of Computer Science, Dartmouth College, afra@cs.dartmouth.edu

continuous time systems see [10, 18].) Given a map $f : X \rightarrow X$ the evolution of the system is defined by $x_{t+1} = f(x_t)$, where $x_t := f^t(x)$ and $t \in \mathbb{N}$ or \mathbb{Z} . An attractor is a set $A \subset X$ to which a dynamical system evolves over time. We state this precisely as follows: A compact set $A \subset X$ is an *attractor* for f if there exists a neighborhood V of A and $N \in \mathbb{N}$ such that $f^N(V) \subset V$ and $A = \bigcap_{n \in \mathbb{N}} f^n(V)$. We illustrate the above with an example.

1.1.1. Example (Cantor set). The Cantor set, F , is the prototypical example of an attractor of a dynamical system that is also fractal. We describe a construction of the Cantor set for which F is the attractor of an iterated function system (IFS) [6] defined in terms of a simple dynamical system. Define the map $f : \mathbb{R} \rightarrow \mathbb{R}$ by

$$f(x) = \frac{3}{2}(1 - |2x - 1|). \quad (1.1)$$

This is often referred to as a *tent map*, see Figure 1.1.

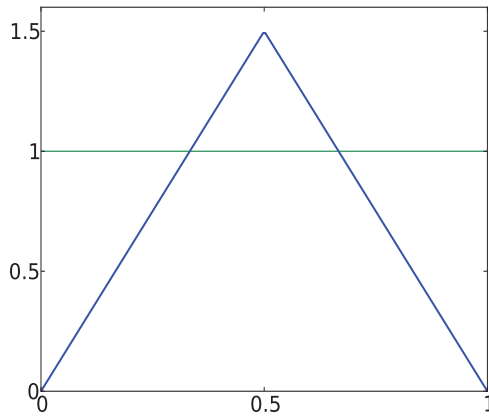


FIG. 1.1. The tent map defined by f . Intersection of the iterations of the branches of the inverse, f_0 and f_1 , give F .

Now define an IFS by the contractions $f_0, f_1 : [0, 1] \rightarrow [0, 1]$ such that

$$f_0(x) = \frac{x}{3} \quad f_1(x) = 1 - \frac{x}{3}. \quad (1.2)$$

The equations (1.2) are the two branches of f^{-1} since

$$f(f_0(x)) = x = f(f_1(x)) \quad (1.3)$$

Let $E = [0, 1]$ and define $g(V) := f_0(V) \cup f_1(V)$, where V is a nonempty compact subset of E . Using the Contraction Mapping Theorem we can conclude that there exists a unique, nonempty, compact attractor in E corresponding to the Cantor set. We can define F in terms of the IFS such that $F = \bigcap_{k=0}^{\infty} g^k(E)$. From (1.3) we see that $f(F) = F$.

We detail some of f 's behavior on F . If $x \in \mathbb{R} \setminus E$, then $f^k(x)$ diverges to $-\infty$ as $k \rightarrow \infty$. Similarly, if $x \in E \setminus F$, then there exists some $N > 0$ such that $x \notin \bigcup \{f_{i_0} \circ f_{i_1} \circ \cdots \circ f_{i_N}(E)\}$, where $i_j \in \{0, 1\}$. Therefore, running the system forwards in time we see that $f^N(x) \notin F$. This shows that F is a repeller.

To aid in the discussion in Section 1.3 we summarize the symbolic dynamics viewpoint of f . The sequence of inverses provide a coding of the points in F as follows: Let $x \in F$, then

$$x = x_{i_1 i_2 \dots} = \bigcap_{j=0}^{\infty} f_{i_1} \circ f_{i_2} \circ \dots \circ f_{i_j}(E). \quad (1.4)$$

Note that this provides an alternative way to write F , namely $F = \bigcup \{x_{i_1 i_2 \dots}\}$. Now, it follows from (1.4) that $|x_{i_1 i_2 \dots} - x_{i'_1 i'_2 \dots}| < 3^{-k}$ whenever $i_i = i'_i, \dots, i_k = i'_k$. Applying f to a point in F we see that $f(x_{i_1 i_2 \dots}) = x_{i_2 i_3 \dots}$. To see that orbits in F are dense under f , consider that for any $x = x_{i_1 i_2 \dots} \in F$, there is a k such that $i_1 = i_{k+1}, i_2 = i_{k+2}, \dots, i_l = i_{k+l}$, from which it follows that $|x - f^k(x)| < 3^{-l}$. From this it follows that periodic points are also dense in F as well.

Points in F also exhibit sensitivity to initial conditions, showing that F is a chaotic repeller for f . Alternatively, it is a chaotic attractor for the IFS defined by g . To see the former, consider $x = x_{i_1 i_2 \dots i_k 0}$ and $x' = x_{i_1 i_2 \dots i_k 1}$ which are within 3^{-k} of one another. Then $f^k(x) = x_0 \in [0, 1/3]$ and $f^k(x') = x_1 \in [2/3, 1]$. Hence, points that begin near one another can diverge under the action of f .



FIG. 1.2. The ternary Cantor set showing the concept of self-similar, or fractal, structure. Shown is the third “level”, i.e. F was iterated 3 times.

We compute the first few iterations and intersections of g , namely $\bigcap_{k=0}^3 g^k([0, 1])$, in Figure 1.2. This is the “third level” of the Cantor set. We will return to the Cantor set below.

1.2. Simplices, Simplicial Complexes, and Simplicial Sets. The simplicial measure that we develop has its origins in computational topology. We therefore continue by describing some structures basic to topology and homology in particular. A more thorough description can be found in [13].

A *simplex* is a geometric object that generalizes the notion of a triangle. A triangle is formed from three vertices all of which are connected to form a 2-dimensional convex hull. In general, an n -dimensional simplex is a polytope formed by the convex hull of $n+1$ vertices. We denote a simplex σ by its ordered set of vertices $[v_0, v_1, \dots, v_n]$ or simply $v_0 v_1 \dots v_n$ for brevity. The dimension of σ we denote by $\dim(\sigma)$. A *face* τ of an n -dimensional simplex σ is also a simplex and is defined as the convex hull formed by a (non-empty) subset of the $n+1$ vertices of σ . A *simplicial complex* is a collection K of simplices such that for each $\sigma_1, \sigma_2 \in K$, $\sigma_1 \cap \sigma_2$ is a face of both or empty; and any face of a simplex in K is also a simplex in K .

Given τ , a face of σ , σ is a *coface* of τ . A *maximal simplex* is a simplex with no proper coface in K .

We now describe the *Vietoris-Rips (VR) complex*. We begin with the Vietoris-Rips neighborhood graph. Given a finite set of points $Y \subset \mathbb{R}^m$ and a real value $\epsilon > 0$, the VR neighborhood graph, $\mathcal{N}_\epsilon = \mathcal{N}_\epsilon(Y)$, is the neighborhood graph composed of nodes (points) from Y , and edges whenever $d(x, y) \leq \epsilon$ for $x, y \in Y$, where d is the Euclidean metric. We can expand a neighborhood graph to a VR complex as follows. Let $\mathcal{N}_\epsilon = \langle V, E \rangle$, where V and E are the

nodes and edges of \mathcal{N}_ϵ , respectively. When all edges of σ , denoted by $e(\sigma)$, are in \mathcal{N}_ϵ we append σ to \mathcal{N}_ϵ to obtain the VR complex

$$\mathcal{R}_\epsilon(Y) := V \cup E \cup \{\sigma \mid e(\sigma) \subset E\}, \quad (1.5)$$

The last term simply states that if all edges of σ are in \mathcal{N}_ϵ then σ belongs to $\mathcal{R}_\epsilon(Y)$. In other words, $\mathcal{R}_\epsilon(Y)$ is composed of simplices whose vertices are each within ϵ of every other vertex in the simplex.

The VR complex is a critical tool for computational topologists as it represents the topology of a point set and is relatively fast to compute [19]. Hence, in the computational setting we utilize VR complexes exclusively. Nevertheless, to obtain the theoretical results below the Čech complex is more suitable. It is constructed similarly to a VR complex [3, 9]. Given ϵ , the Čech complex $C_\epsilon(Y)$ contains a simplex for every subset of balls of radius ϵ with nonempty intersection. It is easy to see that given ϵ' and $\epsilon = 2\epsilon'$, $\mathcal{R}_{\epsilon'}(Y) \subset C_\epsilon(Y)$. The relationship between ϵ and ϵ' can be made tighter, a fact which we formalize in Section 1.3.

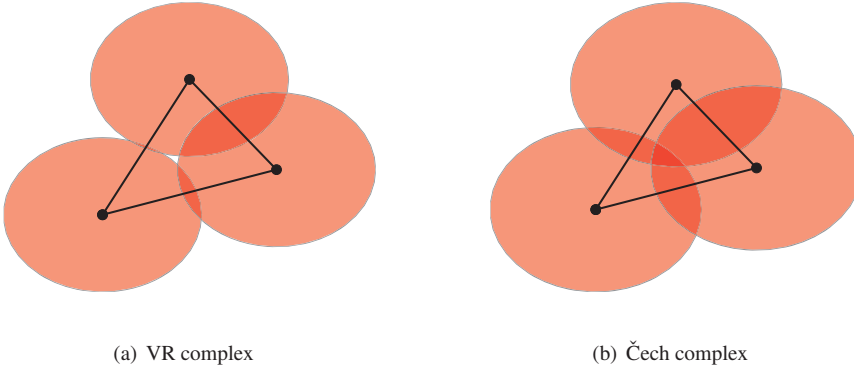


FIG. 1.3. (a) The balls forming the VR complex have radius ϵ . Since all edges are in the graph, the 2-simplex (the triangle) is an element of the VR complex. (b) The Čech complex results from balls of radius $\frac{2}{\sqrt{3}}\epsilon$. In this case the balls have a nonempty intersection, so the 2-simplex defined by the three nodes is included in the Čech complex.

Lastly, a k -skeleton of a simplicial complex K is the subcomplex of K having faces of dimension no larger than k . The 1-skeleton of $\mathcal{R}_\epsilon(X)$ is just the neighborhood graph with a threshold of ϵ consisting of nodes (vertices) and edges between nodes that are less than ϵ apart. Recall that a *clique* is a set of nodes in a graph that include a complete subgraph. If a clique cannot be made any larger then it is termed *maximal*. The maximal cliques in a VR complex are the maximal simplices of the complex.

1.3. Simplicial Measures. Recall that we are interested in using the maximal simplices in a VR complex to understand the density of points in an attractor. We first describe the fundamental notions behind the definition of the simplicial measure. We then define the simplicial measure.

Let f be a diffeomorphism of the manifold X . Suppose U is a neighborhood of the attractor A of f , which satisfies Smale's *Axiom A*. Namely, we require i) that the non-wandering points of f , $\Omega(f)$, form a hyperbolic set; and ii) that the periodic points of f are dense in $\Omega(f)$. Given these assumptions, then for all x in U there exists a probability measure μ , called the

SRB measure, such that [16],

$$\lim_{n \rightarrow \infty} \frac{1}{n} \sum_{k=0}^{n-1} \delta_{f^k x} = \mu, \quad (1.6)$$

where

$$\delta_y(B) = \begin{cases} 1, & \text{if } y \in B \\ 0, & \text{if } y \notin B. \end{cases} \quad (1.7)$$

The limit in (1.6) is taken on the set $\mathcal{B}(X)$ of all Borel probability measures on X which is given the weak* topology. Let $\mu_n = \frac{1}{n} \sum_{k=0}^{n-1} \delta_{f^k x}$ be a measure in $\mathcal{B}(X)$ and let ϕ be a continuous function on X . In the weak* topology $\mu_n \rightarrow \mu$ iff $\int \phi d\mu_n \rightarrow \int \phi d\mu$.

We point out that general tent maps for which $|f'| > 1$, such as the one used to construct the Cantor set in Section 1.1.1, satisfy the requirements of Axiom A systems. The nonwandering set $\Omega(f) = F$ is hyperbolic. And as was shown, the orbits of f , and in particular the periodic points, are dense in F .

In summary, the existence of the SRB measure in (1.6) holds for almost every x in some neighborhood U of A with positive Lebesgue measure. The measure μ is concentrated on A and is typically singular with respect to Lebesgue measure. In other words, the Lebesgue measure of A is zero while $\mu(A) = 1$. And the ergodic averages on the left hand side of (1.6) converge to μ .

In practice, we assume that we have a set of data points $Y \subset X$ generated by the observation of a physical system or by a computer model. By (1.6), the ergodic averages

$$\frac{1}{n} \sum_{k=0}^{n-1} \delta_{f^k x} \quad (1.8)$$

approximate the measure μ on the attractor of the system. The length of the transient can pose issues, and is system-dependent. Methods related to (1.8) have historically been used to analyze both physical systems as well as computer-generated models. Most assume something about the attractor—either its existence or that points in Y tend toward it. Thus, while the SRB measure theoretically provides a strong method of approximating the measure on an attractor, in what follows we are forced to be less rigorous with respect to the actual measure that we use. In other words, we do not assume that the measure we approximate is the unique SRB measure.

Given $\epsilon > 0$, we construct a VR complex on the points in Y . It is convenient for the theoretical development to use Čech complexes formed from balls of radius ϵ . A theorem of de Silva and Ghrist [3] provides the necessary relationship between VR complexes and Čech complexes:

THEOREM 1.1. *For a set of points Y in \mathbb{R}^d , the Čech complex $C_\epsilon(Y)$ is bounded by VR complexes as follows:*

$$\mathcal{R}_{\epsilon'}(Y) \subset C_\epsilon(Y) \subset \mathcal{R}_\epsilon(Y) \quad (1.9)$$

whenever $\epsilon \geq \epsilon' \sqrt{\frac{2d}{d+1}}$.

We conclude from Theorem 1.1 that for each $\sigma \in \mathcal{R}_{\epsilon'}(Y)$, there exists a point $y \in A$ such that $B_\epsilon(y)$ circumscribes the vertices of σ , where $\epsilon = \epsilon' \sqrt{\frac{2d}{d+1}}$. Figure 1.3 illustrates the non-extremal case, in which the vertices of the VR complex are each less than a distance ϵ' from

one another. The centroid falls inside the open neighborhood determined by the intersecting balls in the Čech complex, as in Figure 1.3(b). In the extremal case, where the vertices of $\mathcal{R}_{\epsilon'}(Y)$ are exactly pairwise ϵ' apart, the intersection of the balls in a Čech complex is a single point, y .

In fact, y is unique in both the extremal and non-extremal cases. To see this, define a map from $C_{\epsilon}(Y)$ that chooses the centroid of each simplex,

$$\phi_{\epsilon} = \phi : C_{\epsilon}(Y) \rightarrow \mathbb{R}^d \quad (1.10)$$

given by $\phi(\sigma) = y$.

CLAIM 1.1. ϕ is one-to-one.

Proof. Let $\dim(\sigma) = n$. Define

$$f(x) := \max_{v \in [v_0, \dots, v_n]} \|x - v\|, \quad (1.11)$$

which determines the minimal radius of a ball circumscribing the vertices of σ . If ϕ is not injective, then there exist points $y \neq y'$ such that $f(y)^2 = f(y')^2 = \|v_i - y\|^2 = \|v_i - y'\|^2$ for each vertex v_i . Relabeling $y' = y + \lambda v$, $\lambda > 0$ and $v \in \mathbb{R}^d$, we have that $\|v_i - y\|^2 = \|v_i - (y + \lambda v)\|^2$, which holds iff $\lambda = 0$. \square

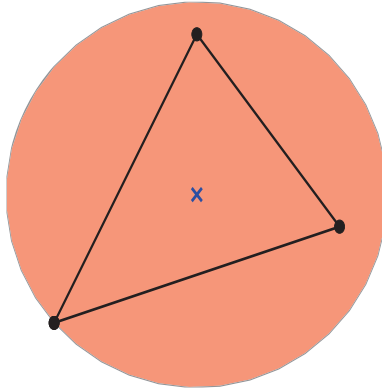


FIG. 1.4. A simplex with an associated circumscribing ball B . The centroid is denoted with an \times . The ball has a residence measure of $3/N$.

Therefore, for each simplex $\sigma \in \mathcal{R}_{\epsilon'}(Y)$ there exists a unique ball centered at $y = \phi(\sigma)$ with radius in $[\epsilon', \epsilon]$ that contains all of the vertices in σ . The simplices in $\mathcal{R}_{\epsilon'}(Y)$ determine the points $y \in A$ which approximate the local density of points in A .

To bring this into the sphere of measure and dimension theory, we define the *residence (probability) measure* on a set $U \subset X$ similarly to (1.6) by

$$\mu(U) = \lim_{m \rightarrow \infty} \frac{1}{m} \#\{k \mid 1 \leq k \leq m, f^k(x) \in U\} \quad (1.12)$$

for any $x \in X$. The measure computes the amount of time iterates of f spend in U . When using the finite set of data points, $Y = \{x_i\}_{i=1}^N$, we approximate μ in Equation (1.12) by

removing the limit. Thus, for a subset $U \subset Y$, let

$$\nu(U) = \frac{1}{N} \#\{k \mid 1 \leq k \leq N, f^k(x) \in U\} \quad (1.13)$$

$$= \frac{1}{N} \#\{x \mid x \in U\} \quad (1.14)$$

Note that $\nu(Y) = 1$. The $\frac{1}{N}$ term will be left off to avoid clutter.

Let $\mathcal{M}_\epsilon(Y)$ be a collection of points in \mathbb{R}^d determined by the centroids in $C_\epsilon(Y)$ so that

$$\mathcal{M}_\epsilon(Y) := \{y \in X \mid \phi(\sigma) = y, \text{ for some } \sigma \in C_\epsilon(Y)\}. \quad (1.15)$$

For each y in $\mathcal{M}_\epsilon(Y)$ there is a radius $\delta \in [\epsilon', \epsilon]$ such that $\nu(B_\delta(y)) = k + 1$ where $k = \dim(\sigma)$ and $\phi(\sigma) = y$. Therefore, by abuse of notation, we can define the *simplicial measure* on simplices $\mathcal{R}_{\epsilon'}(Y)$ as

$$\nu(\sigma) = \nu[B_\delta(\phi(\sigma))] = \dim(\sigma) + 1, \quad (1.16)$$

where $\sigma \in \mathcal{R}_{\epsilon'}(Y)$.

We now formulate a well-defined map between VR complexes $\mathcal{R}_{\epsilon'}(Y)$ and centroids in $\mathcal{M}_\epsilon(Y)$. From the first inclusion in Theorem 1.1, if $\sigma_n = [v_0, \dots, v_n]$ is a n -simplex in $\mathcal{R}_{\epsilon'}(Y)$, then $[v_0, \dots, v_n]$ is a n -simplex in $C_\epsilon(Y)$, where $\epsilon' \left(\sqrt{\frac{2d}{d+1}} \right) \leq \epsilon$. Hence, via the inclusion $i : \mathcal{R}_{\epsilon'}(Y) \hookrightarrow C_\epsilon(Y)$ we associate to each $\sigma_n \in \mathcal{R}_{\epsilon'}(Y)$ an n -simplex in $C_\epsilon(Y)$. Thus, we define

$$g := \phi \circ i. \quad (1.17)$$

This identifies every n -simplex in $\mathcal{R}_{\epsilon'}(Y)$ with a point in A for which there is a ball of radius ϵ with measure $n + 1$.

Since computation and simplex generation occurs at the level of the VR complexes, the diagram in (1.18) serves to connect VR complexes to centroids in $\mathcal{M}_\epsilon(Y)$. The analysis above shows how ϕ allows for the definition of a well-defined map from the VR complex $\mathcal{R}_{\epsilon'}$ to the centroid set, and hence to balls of radius at most ϵ which have the measure of the dimension of each simplex in the complex. We summarize the above discussion in the following diagram:

$$\begin{array}{ccc} & \mathcal{M}_\epsilon & \\ \phi \nearrow & & \nwarrow g \\ C_\epsilon & \xleftarrow{i} & \mathcal{R}_{\epsilon'} \end{array} \quad (1.18)$$

We have dealt above with simplicial complexes containing all possible simplices. Since every simplex contains many faces, and these faces all contain their own centroids, measures of balls around points in $\mathcal{M}_\epsilon(Y)$ would grossly overestimate the density of a region of space according to the simplicial measure. Yet, computing the dimension of A requires above all else accuracy in the estimation of the density of a region. In the next section we briefly discuss the reasons for this in the context of dimension theory.

1.4. Dimension. Fundamentally, determining the dimension of a set relies on “measuring” how much space a set occupies at various scales. Thus, it is imperative that we are able to faithfully estimate the density of a region of space using ν . Thence, we must negate any significant overlap amongst the simplices in $\mathcal{R}_{\epsilon'}(Y)$. We detail in this section the heuristic

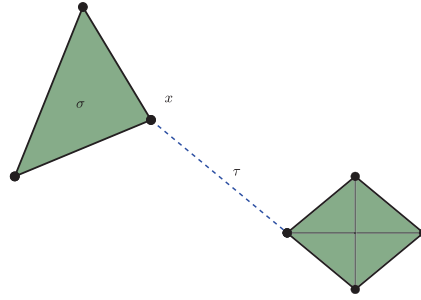


FIG. 1.5. In this small simplicial complex, the two maximal simplices are the filled (green) 2-simplex and 3-simplex. Since $\dim(\sigma) > \dim(\tau)$, $T_x = \{\sigma\}$. Symmetric reasoning holds for the 3-simplex on the right.

notions of “fractal dimension” in order to motivate our subsequent focus on the minimal set of maximal simplices.

Suppose $M_\epsilon(Y)$ is some measurement of the set Y at the scale ϵ . As $\epsilon \rightarrow 0$ we consider how $M_\epsilon(Y)$ behaves. A spacial dimension for A can be approximated using the power law relationship

$$M_\epsilon(Y) \sim \epsilon^{-\alpha}. \quad (1.19)$$

The scaling exponent $\alpha > 0$ is the dimension we seek. Taking the limit of ϵ we get,

$$\alpha = \lim_{\epsilon \rightarrow 0} \frac{\log M_\epsilon(Y)}{-\log \epsilon}, \quad (1.20)$$

which lends itself nicely to numerical estimation of α by finding the slope of the linear regression line to the log-log plot of ϵ vs. M_ϵ . For example, suppose M_ϵ measures ordinary Lebesgue area in \mathbb{R}^2 , then α must be 2.

Sometimes, α depends on the location at which a measurement is taken. In this case one obtains an entire spectrum of scaling exponents leading to the *multifractal spectrum* of a set. We discuss these ideas in more detail in Section 1.7.

1.5. Pruning the Family of Maximal Simplices. A crucial step in utilizing the simplicial measure to determine the dimension of a set is to make sure that the simplicial measure accurately approximates the mass distribution of points at scale ϵ . By considering the simplices of maximal dimension associated to each $x \in Y$, we can exclude the measurement of faces contained in these maximal simplices. We first show how to obtain this from $\mathcal{R}_\epsilon(Y)$ before we describe the computational aspects of this process.

For each $x \in Y$ let

$$S_x = \{\sigma \in \mathcal{R}_\epsilon(Y) \mid x \in \sigma\}. \quad (1.21)$$

By considering only the maximal simplices in S_x we associate to each x the set

$$T_x = \{\sigma' \mid \dim(\sigma') \geq \dim(\sigma), \forall \sigma \in S_x\} \quad (1.22)$$

Figure 1.5 shows an example for a small simplicial complex composed of a 1-,2-,and 3-simplices. The vertex x belongs to both σ and τ , so $S_x = \{\sigma, \tau\}$. Yet $\dim(\sigma) > \dim(\tau)$ so only σ is included in T_x .

Let \mathcal{T} be the family $\{T_x\}_{x \in Y}$ of maximal simplices for each point in Y . In practice, the formation of \mathcal{T} is rather involved. Zomorodian has implemented fast algorithms for the construction of the VR complex as well as the selection of maximal simplices from such a complex (see [19, 20]). We utilize these to obtain \mathcal{T} for a VR complex $\mathcal{R}_\epsilon(Y)$.

For the example in Section 1.6.1, \mathcal{T} is composed of all sets in Figure 1.6. In general, the generation of maximal simplices does not create disjoint simplices. Thus, even with \mathcal{T} in hand, we must implement a second round of pruning.

Algorithm 5 DISJOINT-COVER(\mathcal{T})

```

1:  $C \leftarrow \emptyset$  //  $C$ : disjoint simplicial set
2:  $P \leftarrow \emptyset$  //  $P$ : candidates for filling gaps
3:  $R \leftarrow \emptyset$  //  $R$ : non-disjoint sets
4: while  $\mathcal{T} \neq \emptyset$  do
5:    $\sigma \leftarrow \mathcal{T}.\text{next}()$  // remove largest simplex from  $\mathcal{T}$ 
6:    $L \leftarrow |\sigma \setminus C|$  // number of non-intersecting vertices in  $\sigma$ 
7:   if [ $\sigma$  is disjoint from  $C$ ]  $L = |\sigma|$ 
8:      $C \leftarrow C \cup \sigma$  // add  $\sigma$  to cover
9:      $C.\text{vertices} \leftarrow C.\text{vertices} \cup \sigma.\text{vertices}$ 
10:  else
11:     $R \leftarrow \sigma$ 
12:  end if
13: end while
14:  $\text{make-hash}(R, P)$  return  $C, P$ 

```

1.6. The Tidy Cover Algorithm. Recall that a collection of sets $\{V_\alpha\}_{\alpha \in \mathcal{J}}$ is said to cover a set Z if

$$\bigcup_{\alpha} V_\alpha \supset Z, \quad (1.23)$$

where \mathcal{J} is an index set. We define a *simplicial cover* to be a cover of the set Y by a collection of simplices from $\mathcal{R}_\epsilon(Y)$. We use the algorithms DISJOINT-COVER and GAP-COVER to pare down \mathcal{T} to a subcollection of simplices that is close to the minimal number necessary to cover Y . We term this collection a *tidy cover*. Together we call DISJOINT-COVER and GAP-COVER the TIDY-COVER algorithm.

Algorithm 6 MAKE-HASH(R, P)

```

1: for  $\sigma \in R$  do
2:    $m \leftarrow |L|$ 
3:    $\sigma.\text{complement} \leftarrow L$  // add complement vertices as simplex attribute
4:    $P[m] \leftarrow P[m] \cup \sigma$  // add  $\sigma$  to  $P$ , indexed by  $m$ 
5: end for

```

Before calling DISJOINT-COVER we sort the simplices in \mathcal{T} in decreasing order. The primary purpose of DISJOINT-COVER is to cover Y as completely as possible with a collection of the largest, disjoint simplices from \mathcal{T} . While it creates this collection an additional data structure is constructed and returned along with C . These objects aid GAP-COVER in efficiently filling in the gaps in C . We use the “.” notation from C++ to denote attributes and methods in the algorithms.

As mentioned, DISJOINT-COVER builds a disjoint collection from the largest non-intersecting simplices in \mathcal{T} (line 7). While building C , if it finds a simplex that intersects one of the members of C (line 11), this simplex is stored in R . When a disjoint cover is obtained, the hash table P is constructed in MAKE-HASH. Keyed by the complement sizes of simplices left out of C , P maps these complement sizes to lists of simplices having that size complement. By sorting P by its keys in decreasing order, this data structure allows GAP-COVER to short circuit earlier by adding the simplices that most efficiently cover the gaps of points left uncovered during the construction of C .

Algorithm 7 GAP-COVER($C, P, N = |Y|$)

Require: P sorted by keys in decreasing order

```

1: for  $m \in P$  do
2:   while  $P[m] \neq \emptyset$  do
3:      $\sigma \leftarrow P[m].\text{next}()$  // remove  $\sigma$  with (possibly) most uncovered vertices
4:      $m \leftarrow |\sigma.\text{complement} \cap C|$ 
5:     if [ $\sigma$  has the most uncovered vertices]  $m = 0$ 
6:        $C \leftarrow C \cup \sigma$  // add to cover
7:        $C.\text{vertices} \leftarrow C.\text{vertices} \cup \sigma.\text{vertices}$  // mark vertices as covered
8:     else [earlier addition to cover altered complement]
9:        $\sigma.\text{complement} \leftarrow \sigma \setminus C$ 
10:       $m' \leftarrow |\sigma.\text{complement}|$ 
11:       $P[m'] \leftarrow P[m'] \cup \sigma$  // update  $P$  according to  $\sigma$ 's uncovered vertices
12:     end if
13:     if  $|C.\text{vertices}| = N$  then return  $C$ 
14:     end if
15:   end while
16: end for

```

Now consider the GAP-COVER algorithm. As noted, P is sorted by keys; let m be the largest key, or complement size. Assuming that C is not a cover of Y , GAP-COVER starts by adding the first simplex, say σ , in the list of simplices pointed to by $P[m]$ (line 5). Thus, the most possible new points are added to C . Once σ is added, it is possible that the complement sizes of some of the remaining simplices in P have changed. We determine this in a “lazy” way by waiting until a simplex τ is chosen from $P[m]$ for which $\tau.\text{complement} \cap C$ is non-empty. Then m (line 3) will be different from zero, triggering GAP-COVER to update τ 's position in P (lines 7-10). Note that elements in $P[m]$ are removed from the front of the list and not returned. If $P[m]$ is empty, m is incremented to the next largest key (complement size). During this process, if the number of vertices covered by simplices in C equals the number of data points, GAP-COVER terminates.

An important aspect of the above algorithms is that only one set of simplices is constructed and used. While a number of data structures are constructed, only pointers are manipulated as simplices are “moved”. So the memory footprint is of size $O(\mathcal{T})$.

By construction, the TIDY-COVER algorithm returns a tidy cover of Y . We compared the TIDY-COVER algorithm to a naive greedy cover algorithm that updates complements without the use of the adjacency graph. On a multifractal Cantor set with 10,000 points (see Section 1.6.1) TIDY-COVER returns a tidy cover approximately 250 times faster than an algorithm that fills the gaps but naively updates the complements without using the hash P .

With the tidy cover in hand we have an accurate notion of the density of points in an attractor A using the sampling of points Y . We can now employ the techniques of dimension

theory to approximate the fractal or multifractal properties of A .

We revisit the Cantor set to illustrate the above algorithms.

1.6.1. Example (Cantor set cover). Consider F , the Cantor set defined in Section 1.1.1. Let $p+q = 1$ and wlog assume $p < q$. We construct a measure μ on F . Consider the collection E_k of 2^k intervals of length 3^{-k} at each level of the construction of F . Then

$$F = \bigcap_{k=0}^{\infty} E_k. \quad (1.24)$$

The first level of the Cantor set, E_1 , contains left and right subintervals, E_{10} and E_{11} , respectively. For the left interval assign a mass of p and for the right a mass of q . Similarly, E_2 is composed of four subintervals: A left and a right subinterval within both of E_{10} and E_{11} . As before, to each of the left intervals assign a mass of p , and to each of the right assign a mass of q . Continuing to divide the mass amongst the subintervals in the ratio $p : q$ yields a mass distribution on F [6]. The resulting set with the associated measure is referred to as a multifractal Cantor set.

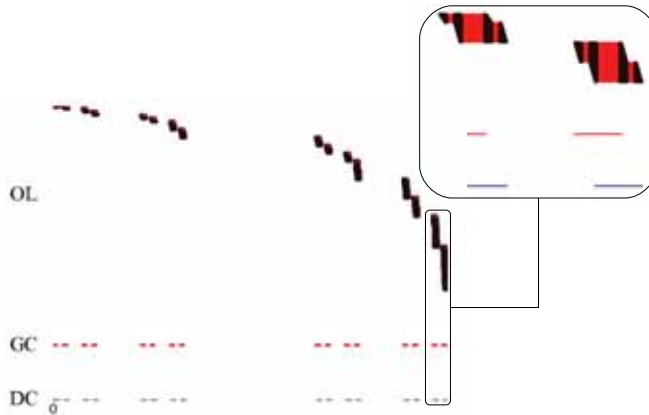


FIG. 1.6. The Cantor set constructed with uneven mass distribution, covered by a disjoint simplices (DC) and a set covering the gaps (GC). The tidy cover from Section 1.6 is the union of DC and GC. The stacks of sets (OL) above are the simplices that overlap DC and are not used in GC. The disjoint collection DC contains 16 simplices, the final cover contains 32 simplices, and there are 1645 sets discarded. The difference in sizes in OL is due to the uneven distribution of points. The blow-up shows in detail the overlap created by the sliding ϵ window mentioned in the text.

Given a radius ϵ we form a VR complex on the distribution of the N points at the k th level of the Cantor set. Intuitively, since the intervals are all the same length, points are more or less densely packed according which interval they fall into. For instance, the mass of an interval at the k th level depends on the number of 0's and 1's in the coding described in Section 1.1.1. An interval for which $i_j = 0$ n times has mass $p^n q^{k-n}$.

Since the Cantor set is 1-dimensional, the simplices are composed of points arranged along intervals within a ball, or window, of size $\epsilon/2$ of one another. In regions of high density, “sliding” this ϵ -window will create many maximal simplices and impressive amounts of overlap in a narrow region. The detail in Figure 1.6 shows the maximal simplices spread out vertically that result from this phenomenon. One can see how the density in a region affects the amount of overlap. The number of simplices in \mathcal{T} in a region, as well as the density of points in that region, is correlated to the height of the stacks OL in Figure 1.6. The

original arrangement of simplices can be recovered by projecting all of the simplicies in the figure onto the x -axis (the horizontal level of DC).

In Figure 1.6, DC is the set of simplices returned by DISJOINT-COVER and GC is the set returned by GAP-COVER. The tidy cover is the union of the simplices in DC and GC. OL is the remainder of maximal simplices from \mathcal{T} that were unused in the final cover.

1.7. Results for the Tidy Cover. The *box-counting dimension* is calculated by defining $M_\epsilon(Y)$ to be the number of sets of “size” ϵ necessary to cover Y . Computationally this is often done by “gridding” the space into squares of side length ϵ . Counting the number of grid elements occupied by at least one point from Y gives $M_\epsilon(Y)$. For example, consider the ordinary Cantor set described in Section 1.1.1. At the k th level there are 2^k sets of size 3^{-k} so the box-counting technique yields $\alpha = \log(2)/\log(3) \approx 0.631$, the fractal dimension of the Cantor set.

A numerical approximation of the box-counting dimension of the Cantor set using simplices in the tidy cover yields a similar result. We calculate a scaling exponent $\alpha \approx 0.656$, which is within 0.025 of the true dimension. M_ϵ is the number of simplices in the tidy cover at scale ϵ . The log-log plot of this behavior is shown in Figure 1.7. We considered a sequence of tidy covers for ϵ in the range $[3^{-10}, 3^{-3}]$.

Physical and mathematical systems often exhibit regions of varying density in their attractors [7, 8]. (Mandelbrot’s book contains many interesting examples and areas of real-world study [11].) Let μ be a measure on such an attractor A . Then this situation manifests itself when the set of points for which $\mu(B_\epsilon(x)) \sim \epsilon^{-\alpha}$ determines a different fractal for a range of α ’s. The box-counting dimension ignores the finer structure of a measure and so we must introduce a set of tools borrowed from statistical physics [5, 6, 17]. We briefly describe how these work.

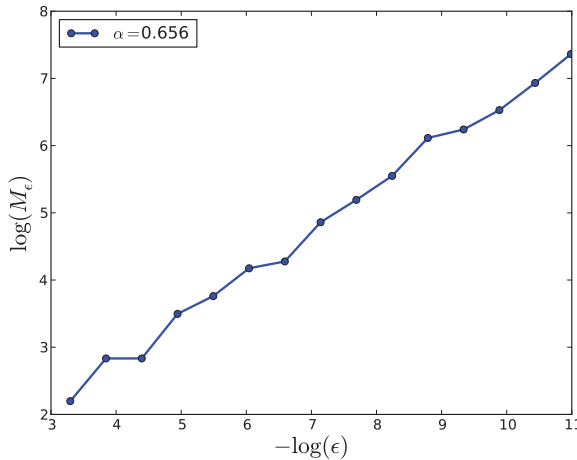


FIG. 1.7. Estimation of the box counting dimension of the Cantor set by the number of simplices contained in tidy covers at different ϵ ’s.

We introduce the partition function for $q \in \mathbb{R}$ and $\epsilon > 0$:

$$Z(q, \epsilon) = \sum_{\mathcal{B}} \mu(B_\epsilon)^q, \quad (1.25)$$

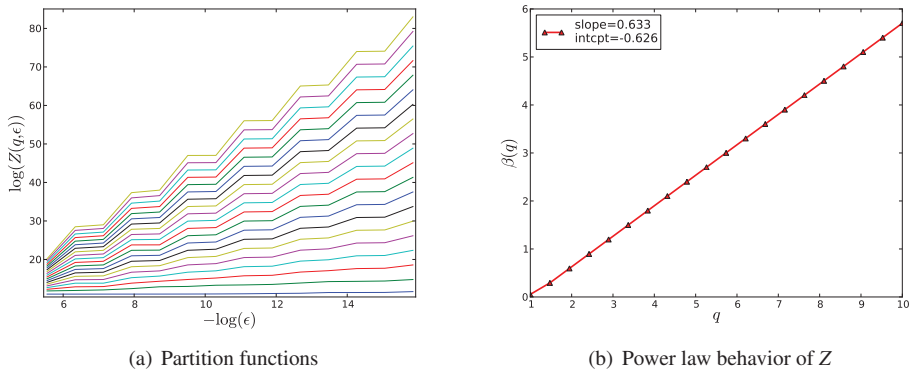


FIG. 1.8. In (a) each curve represents $Z(q, \epsilon)$ as a function of ϵ with a fixed value of q . The scaling behavior of the Cantor set is clearly visible in the curves. The $\beta(q)$ curve in (b) shows the scaling behavior of each $Z(q, \epsilon)$ curve across values of q .

where \mathcal{B} is the set of balls B of radius ϵ with $\mu(B) > 0$. In thermodynamics, q represents inverse temperature and μ measures a “state” of the system [14, 17]. We identify the power law behavior of Z for each q by [6]

$$\beta(q) = \lim_{\epsilon \rightarrow 0} \frac{\log Z(q, \epsilon)}{-\log(\epsilon)}. \quad (1.26)$$

We apply the partition function and Equation 1.26 to the ordinary Cantor set, C . In this case there is no change to the mass distribution on C , and therefore we expect a single scaling exponent. This forces the function $\beta(q)$ to be linear. For a measure we use the simplicial measure ν on each simplex in the tidy cover at scale ϵ . We use the same sequence of tidy covers as in the estimation of the box-counting dimension. The scaling structure of C can be seen in the partition functions in Figure 1.8(a). The behavior of Z over a range of q is seen to be linear in Figure 1.8(b) indicating a single fractal dimension. Let $f(\alpha)$ denote the dimension of the set of points for which the power law $\mu(B_\epsilon(x)) \sim \epsilon^{-\alpha}$ holds. It is obtained from β via the Legendre transform [1, 6, 14].

$$f(\alpha) = \sup_{\alpha \in \mathbb{R}} \{\alpha q - \beta(q)\}. \quad (1.27)$$

Since β in Figure 1.8(b), is linear the Legendre transform of β is a single point located at $(\alpha, f(\alpha))$. Note that $f(\alpha)$ is just the negative of the y-intercept of the line tangent to β with slope α . And so the dimension of the set for which $\mu(B_\epsilon(x)) \sim \epsilon^{-\alpha}$ holds is $f(\alpha)$. For C , we estimate from the functions shown in Figure 1.8 that the dimension is $f(\alpha) = f(0.633) = 0.626$, which is close to the exact value of $\log(2)/\log(3)$.

If β is not linear, an entire $f(\alpha)$ spectrum is derived via the Legendre transform of β . This yields the various fractal dimensions of the sets with scaling exponent α . We have studied this situation using both the multifractal Cantor set discussed in Section 1.6.1 as well as the 2-dimensional Henon attractor. Both of these dynamical systems have been shown in other work to have rich multifractal structures leading to a spectrum of $f(\alpha)$ dimensions. Our results are preliminary and unfortunately do not show the nonlinear behavior expected of β .

1.8. Conclusions. Attractors of relatively simple physical and dynamical systems often possess rich structure whose study often sheds light on the system itself.

Recent work in computational topology has led to efficient algorithms for computing simplicial structures on Y . We have introduced the notion of a simplicial measure using simplices from a VR complex constructed on Y . In order to utilize this measure to approximate the dimension of an attractor from the points in Y we must filter the simplices in the VR complex.

The first step uses the algorithms implemented by Zomorodian to create a subset of maximal simplices. We then developed the `TIDY-COVER` algorithm to deal with the overlap problem in the set of maximal simplices. The algorithm creates a cover in an efficient and greedy way, using an adjacency graph and complement sizes to keep updates to a minimum. We demonstrated the way in which this algorithm extracts a tidy cover from the VR complex on a multifractal Cantor set, yielding a “minimal” set of maximal simplices that cover the set by the simplices of highest dimension possible. By construction, these simplices have very little overlap.

We showed the usefulness of the tidy cover in computing dimension. On an ordinary Cantor set the tidy cover works very well for computing the dimension using box-counting techniques. Furthermore, we demonstrated that techniques used on multifractal sets, such as the partition function, behave well when the simplicial measure is used with the simplices in the tidy cover. In both cases we were able to approximate the fractal dimension of the Cantor set to within 0.025 of the actual dimension.

Our numerical methods have not yet performed well on multifractal systems, such as the multifractal Cantor set discussed in Section 1.6.1 or the Henon attractor [6]. We will address these issues in future work.

2. Extensions for the TEVA-SPOT Toolkit. The Threat Ensemble Vulnerability Assessment and Sensor Placement Optimization Tool (TEVA-SPOT) is a joint project of the U.S. Environmental Protection Agency, Sandia National Laboratories, Argonne National Laboratory, and the University of Cincinnati. TEVA-SPOT was designed to model a wide range of sensor placement problems to optimize a real-time, on-line contaminant warning system (CWS) [2].

This section details a number of recently developed Python extensions to the TEVA-SPOT Toolkit.

2.1. SP Module. The TEVA-SPOT sensor placement solvers are launched using the `spmodule`. The module takes as input one or more `IMPACT` files and returns a sensor placement. `sp` interfaces with various MIP solvers. (See [2] for more details.)

2.2. Goal of the Extensions. We briefly describe optimization using objectives defined in TEVA-SPOT. In this work optimization is synonymous with minimization of an objective. A *Pareto optimal* set in the solution space is the set of feasible solutions such that for each solution, there exists no feasible solution that is better than the reference solution in at least one objective and is no worse in the remaining objectives [4].

In the preliminary work reported in this paper we consider the set of Pareto optimal solutions associated with two objectives in a CWS, namely “expected contamination” (EC, measured in pipe feet) and “mass consumed” (MC, measured in mass of contaminant). If no bound is put on MC, then optimizing EC will yield a mean impact of approximately $EC \approx 8500$. But in this case MC ends up quite high with a mean impact of $MC \approx 43650$. If we insist on a better value of MC we must constrain it by passing in an upper bound to the solver. In doing so we necessarily drive EC up. A *Pareto front* or *Pareto frontier* is a set of discrete or continuous regions representing such a set of tradeoffs. In general, these regions form $d-1$ dimensional hypersurfaces in the solution space. The points on the Pareto front *dominate* the

other solutions. Figure 2.1 shows the Pareto front generated by restricting MC to values less than 43650. Infeasible solutions result for values of MC less than $\sim 22,000$.

The primary goal of the extensions is to study such Pareto fronts. There are two avenues that we plan to follow. First, we wish to investigate the structure of the near optimal solutions along the Pareto front. We expose `pico`'s `enumRelTol` option in `sp` to allow an optimality gap within a given percentage of an optimal solution. For instance, we constrain MC and require that `pico` return all solutions within (say) 10% of the optimal EC value at the given MC value. The Pareto front itself is discrete and forms a relatively sparse set in the solution space. The near-optimal solutions form a set with potentially interesting topological features. In addition, knowledge of near-optimal solutions allows more flexibility in real-world sensor placement decisions. See Figure 2.1.

Another route of analysis is the graphical analysis of the water network itself. Statistics such as the frequency of junction selection in solutions, distance between sensors (in linear pipe feet), and mean impact of the sensor placement are relatively easy to study as node and edge properties in a graph. See Figure 2.2.

2.3. SP Extensions for Near-Optimal Analysis. A number of Python modules were added to the SPOT-TEVA code this summer to facilitate data analysis. They fall in a few general categories. First, the most generically useful adds parsing and conversion utilities via the `RE_PARSER.PY` module. Extraction of data for analysis relies on this module. Secondly, the results can be analyzed in a traditional way by plotting one objective against a constrained objective. This facility is generalized to allow the user to specify the objectives to be plotted. Thirdly, the solutions can be considered on the network that produced the scenarios, with appropriate statistics computed for nodes in the network. All of the extensions in Table 2.1 are new, with the exception of `sp2` which is a copy of `sp` with `pico`'s `enumRelTol` option exposed. We describe their capabilities in more detail below.

1. `sp2`: Added option parsing ability that exposes `Pico` option `--enumRelTol` and associated value. This extension is in the `PICO()` class and appends the option onto the `PICO` call. Since this is the only addition to `sp` we will simply reference the `sp` module where it appears below.
2. `RE_PARSER`: This regular expression parser handles `sp`'s text output. This is quite useful in general since it creates Python dictionaries from `sp` output. We summarize its current abilities:
 - Handles regular `sp` output and enumerated output (only from `pico`'s raw text output at the moment).
 - Write solutions to `YAML` `SOLN` files. Also loads these `YAML` files.
 - Functions `PARSE_COORDINATE` and `PARSE_PIPES` handle line input from `INP` file. Respectively, return a geographic coordinate of each junction ID in the water network and the pipe connections (eg., edges in a graph of the water network).
3. `CONSTRAINED_OBJECTIVE`: This module interfaces with `sp` to run optimization problems with one constrained objective (see Figure 2.1). It contains two classes as well as all options for input parsing. One handles command processing; the other handles the subprocess call to `sp`:
 - Class `PROCESSOR()` contains the low level processing for input (creating command to pass to `sp`) and output (handling text solution returned or written to disk by `sp`). It primarily uses `RE_PARSER` for this.
 - Class `CONSTRAINEDOBJECTIVE()` inherits from `PROCESSOR()`. It sets up the call to `sp`, handles any enumeration parsing that must be done (assuming more than `pico` was operational via `sp`), and handles the returned solution with `PROCESSOR`'s

TABLE 2.1
TEVA-SPOT *Python Extensions*

Name of extension	Summary	Status
sp2	A version of sp exposing PICO's <code>enumRelTol</code> option.	Working. Calls PICO; sp's default output does not handle enumeration (handled via <code>RE_PARSER.PY</code>).
CONSTRAINED_OBJECTIVE	Formulates a command to pass to sp2 for calling a solver with a constrained objective.	Working. Currently works only with PICO's <code>enumRelTol</code> option now. Uses <code>RE_PARSER.PY</code> for solution parsing.
RE_PARSER	Parses output from sp and PICO enumerations. Writes results in dictionary format to YAML file.	Working. Should be easy to add new parsing tools by follow model of existing ones.
OPTIMA_ANALYZER	Contains tools for plotting Pareto fronts, plotting networks (using <code>NETWORK()</code> class), as well as exporting to Protovis format.	Working. NetworkX graphs, graph analyzers and IO utilities are all stable.
NETWORK_PROTOVIS	Converts node and edge data created in <code>NETWORK()</code> class and saved in .npy format to VTK format for use with Titan's Protovis interface[15]. This basically pipes the data to an interactive web browser format.	alpha. Draws the network, and node sizes. There are still issues with Protovis/-javascript.

methods.

4. `OPTIMA_ANALYZER`: This module has two main purposes (plus unfinished tools):

- Plot Pareto front by extracting solutions from SOLN files (Python dicts).
- Class `NETWORK()` inherits from `NETWORKX.GRAPH()` package. Nodes are water network junctions, with node data 'junctionID', 'size', 'longitude', 'latitude'. All data is extracted from appropriate INP file except 'size', which is determined by how often a junction occurs in all enumerated solutions (in SOLN files). Edges link nodes using Pipes field in INP files.
- An experimental `NETWORK` attribute related to nodes and edges is the 'sensor' attribute. This attempts to split 'size' into a more refined set of quantities according to how often a node is chosen 1st, 2nd, etc.
- All of the node, edge, and sensor attributes of a `NETWORK` object can be exported as a `NUMPY` array with custom data type. For instance, nodes \rightarrow `array((junctionID, longitude, latitude, size), dtype=(int, float, float, float))`. These are more easily read and parsed for conversion to VTK Tables in `NETWORK_PROTOVIS`.

5. `NETWORK_PROTOVIS`: *Note: This is written on top of Titan, which must be installed*

to run this module. This module is very basic at the moment. All file names are hardcoded—i.e. command line interaction has not been included yet. NETWORK_PROTOVIS reads NUMPY arrays stored on disk (see OPTIMA_ANALYZER), converts them to VTK Tables, and converts these to an appropriate format (JSON) for Protovis. VTK includes a wrapper for Protovis, so we embed Javascript code that is piped Protovis. Calls browser and renders network. Output is similar to the network rendered by OPTIMA_ANALYZER but with more interactive qualities. See Figure 2.2(b).

6. NEAR_OPTIMA_SCRIPT: This small module loops over a set of upper bounds, running CONSTRAINED_OBJECTIVE at each iteration.

2.4. Conclusions. We have developed extensions to the TEVA-SPOT software package that facilitate data analysis. Most of the tools developed and discussed above are fairly general and can be ported to other optimization problems with few modifications. Additionally, the network visualization tools provide novel ways to understand multi-objective optimization problems. Therefore, we are continuing to develop the visualization aspects of these modules.

REFERENCES

- [1] V. I. ARNOLD, *Mathematical Methods of Classical Mechanics*, Springer, 2nd ed., 1989.
- [2] J. BERRY, E. BOMAN, L. A. RIESEN, W. E. HART, C. A. PHILLIPS, AND J.-P. WATSON, *TEVA-SPOT Toolkit 2.3 User's Manual*, Sandia National Laboratories, Albuquerque, NM, October 2009.
- [3] V. DE SILVA AND R. GHRIST, *Coverage in sensor networks via persistent homology*, Algebraic & Geometric Topology, 7 (2007), pp. 339–358.
- [4] K. DEB, *Multi-objective optimization using evolutionary algorithms*, John Wiley & Sons, 2001.
- [5] J.-P. ECKMANN AND D. RUELLE, *Ergodic theory of chaos and strange attractors*, Reviews of modern physics, 57 (1985).
- [6] K. FALCONER, *Fractal Geometry*, 2003.
- [7] P. GRASSBERGER, *Generalized dimensions of strange attractors*, Physics Letters A, 97 (1983), pp. 227–230.
- [8] T. HALSEY, M. JENSEN, L. KADANOFF, I. PROCACCIA, AND B. SHRAIMAN, *Fractal measures and their singularities: the characterization of strange sets*, Physical Review A, 33 (1986), pp. 1141–1151.
- [9] J. G. HOCKING AND G. S. YOUNG, *Topology*, Dover, New York, 1961.
- [10] A. KATOK AND B. HASSELBLATT, *Introduction to the Modern Theory of Dynamical Systems*, Cambridge University Press, Cambridge, 1995.
- [11] B. MANDELBROT, *The Fractal Geometry of Nature*, W.H. Freeman, 1983.
- [12] K. MISCHAIKOW AND M. MROZEK, *Chaos in the Lorenz equations: a computer-assisted proof*, Bulletin of the American Mathematical Society, 32 (1995), pp. 66–72.
- [13] J. R. MUNKRES, *Elements of Algebraic Topology*, Addison-Wesley, Redwood City, CA, 1984.
- [14] Y. B. PESIN, *Dimension Theory in Dynamical Systems: Contemporary views and applications*, University of Chicago Press, Chicago, 1997.
- [15] PROTOVIS, <http://vis.stanford.edu/protovis/>. Protovis: A graphical approach to visualization.
- [16] D. RUELLE, *A Measure Associated with Axiom-A Attractors*, American Journal of Mathematics, 98 (1976), pp. 619–654.
- [17] ———, *Thermodynamic Formalism: The Mathematical Structures of Equilibrium Statistical Mechanics*, Cambridge University Press, 2nd ed., 2004.
- [18] S. STROGATZ, *Nonlinear dynamics and chaos: With applications to physics, biology, chemistry, and engineering*, Westview Pr, Cambridge, 2000.
- [19] A. ZOMORODIAN, *Fast construction of the Vietoris-Rips complex*, Computers & Graphics, (2010).
- [20] A. ZOMORODIAN, *The Tidy Set: A Minimal Simplicial Set for Computing Homology of Clique Complexes*, in Proc. ACM Symposium of Computational Geometry, 2010.

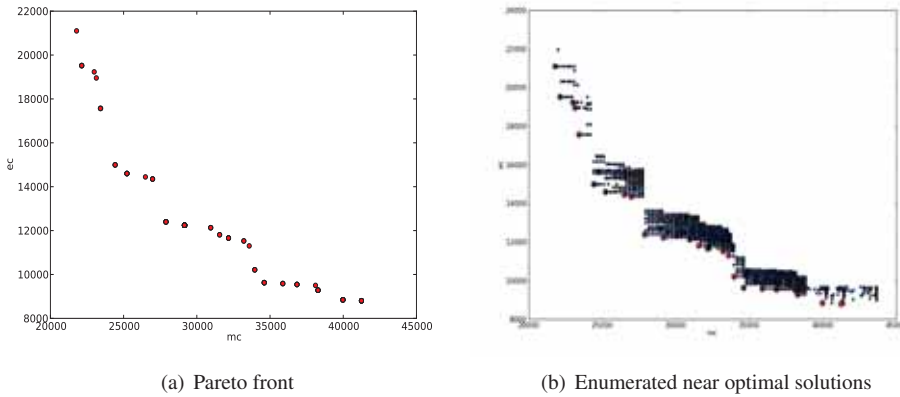


FIG. 2.1. (a) The Pareto front for objectives EC and MC, where a sequence of upper bounds have been imposed on MC (side-constraints). The true optimum for minimization of EC for the given scenario (5 sensors and given IMPACT files) is the point in the lower right. (b) Enumeration of solutions within 10% of Pareto optimal for the same sequence of upper bounds.

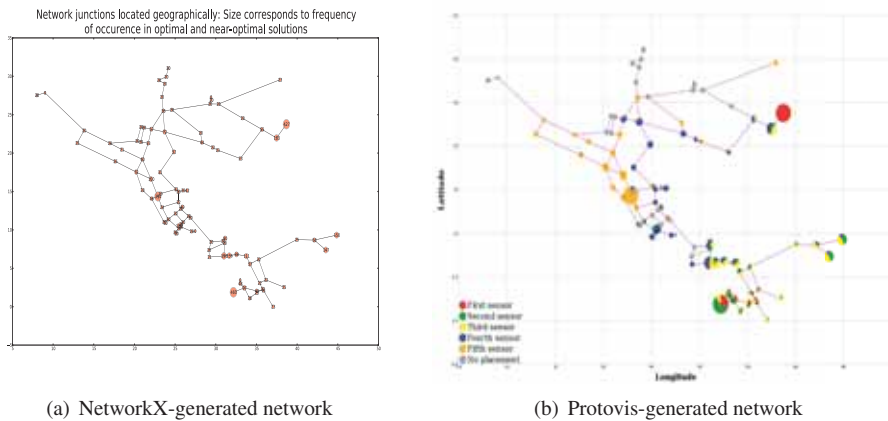


FIG. 2.2. (a) Static image of the Net3 sample network. Node size corresponds to occurrence of nodes in near-optimal solutions. (b) Titan/Protovis-generated graph. Node size is the same as in (a). Each node carries “mouseover” information. Many types of interactive statistics can be added using the Protovis package. We interface to it using Titan’s Python library.

BENDERS DECOMPOSITION IN PYOMO

PATRICK STEELE* AND JEAN-PAUL WATSON†

Abstract. Large scale linear programs often have computationally intractable numbers of variables. Decomposition methods such as Benders decomposition can reduce this number, but require very specific problem structures not always present in the original formulation. We present a tool that removes the need to explicitly construct a problem with such structure through the use of the algebraic modeling language Pyomo, a component of the Coop project.

1. Introduction.

1.1. Mathematical Programming. A mathematical programming or mathematical optimization problem is the problem of finding an element x in some set \mathcal{S} that minimizes an *objective function* $f : \mathcal{S} \rightarrow \mathbb{R}$. A specific class of such problems are *convex optimization problems* or *convex programs*, which can be expressed in the form

$$\begin{aligned} \min \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & g_i(\mathbf{x}) \leq b_i, i = 1, \dots, m, \end{aligned} \tag{1.1}$$

where $f, g_i : \mathbb{R}^n \rightarrow \mathbb{R}$, $i = 1, \dots, m$ are convex functions. Convex programs are a notable class of problems because convexity often makes finding a solution more computationally tractable than in a nonconvex program [5]. A class of convex programs that are particularly well-studied are *linear programs*, in which all functions are not only convex, but linear. Linear programs are most often expressed in matrix notation. If the variables in the vector $\mathbf{x} \in \mathbb{R}^n$ are *decision variables*, i.e., the variables manipulated to minimize the objective function, then a general linear program has the form

$$\begin{aligned} \min \quad & \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}, \end{aligned} \tag{1.2}$$

where the matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ and the vector $\mathbf{b} \in \mathbb{R}^m$ form m constraints on \mathbf{x} , and the elements c_i of the vector $\mathbf{c} \in \mathbb{R}^n$ are the cost coefficients of each variable x_i . We say a vector $\bar{\mathbf{x}} \in \mathbb{R}^n$ is *feasible* to a problem if $\bar{\mathbf{x}}$ satisfies all the constraints in the problem.

Although no closed-form solution to a general linear program is known to exist, several algorithms have been developed to solve linear programs in acceptable amounts of time. One commonly used algorithm is the simplex method, developed by George Dantzig, which at worst requires exponential time to solve but in practice is quite efficient [11]. Many interior point methods, such as that developed by Narendra Karmarkar, have polynomial runtime complexity in the worst case [10]. Linear programming techniques are used to solve problems ranging from portfolio management [12] to energy planning and forecasting problems [7] to shortest path and network flow problems [6, 2]

1.2. Algebraic Modeling Languages. Linear programming problems often have a very compact linear algebra formulation, such as (1.2). Other times, the form of each constraint in the problem varies enough that forming the correct matrix and vectors to represent the problem as in (1.2) would be quite tedious. In either case, explicitly writing out all the constraints in a problem is often unnecessary for human understanding, since many constraints have the

*The College of William and Mary, Department of Mathematics, prsteele@email.wm.edu

†Sandia National Laboratories, Discrete Math and Complex Systems, jwatson@sandia.gov

same form, differing only by coefficient values — in fact, many problems solved today involve millions of variables and constraints, making a hand-constructed problem impossible. To overcome this, linear programming problems are often expressed via *algebraic modeling languages*.

Algebraic modeling languages (AMLs) are computer languages designed to express mathematical ideas in a form that can be understood by a human. Although applicable to a wide range of mathematics, many AMLs are specifically designed to express mathematical programming problems. Both commercial AMLs, such as AMPL (A Math Programming Language), GAMS (the General Algebraic Modeling System), or OptimJ, as well as open source AMLs, such as FLOPC++ or MathProg, are commonly used to express linear programming models in a form that can be manipulated by a solver. Some AMLs such as AMPL, GAMS, and MathProg are stand-alone pieces of software with their own languages, while others are embedded in general programming languages, like OptimJ (Java) and FLOPC++ (C++).

Another embedded AML is Pyomo (Python Optimization Modeling Objects), a component of the CoopR (Common Optimization Python Repository) project [9, 1]. Pyomo is based in Python, an interpreted, object-oriented programming language with a well-established user- and code-base. In addition, Pyomo works cooperatively with other components of the CoopR project, such as PySP, a stochastic programming toolkit, and CoopROpt, a Python interface to many optimizers [8].

2. Benders Decomposition.

2.1. Motivation. Let us consider the problem of production planning. A company manufactures a number of products which they sell to their customers. They have already taken orders for the current year, and must fulfill these orders before the same time next year, at which time they will receive orders for the second year. The company must purchase raw materials to produce their products; the price of raw materials is expected to increase next year, so the company can decide to purchase excess raw materials and store them for a fee.

Formally, the decision problem can be stated as

$$\begin{aligned}
 \min \quad & \sum_{i \in \mathcal{P}\{1,2\}} (f_i(x_i) + f_i(w_i)) + \sum_{k \in \mathcal{R}} (g_{k1}(y_k) + g_{k2}(z_k)) + \sum_{k \in \mathcal{R}} h_k(s_k) \\
 \text{s.t.} \quad & x_i \geq D_{i1}, \quad \forall i \in \mathcal{P} \\
 & w_i \geq D_{i2}, \quad \forall i \in \mathcal{P} \\
 & \sum_{i \in \mathcal{P}} r_k(x_i) = y_k - s_k, \quad \forall k \in \mathcal{R} \\
 & \sum_{i \in \mathcal{P}} r_k(w_i) \leq z_k + s_k, \quad \forall k \in \mathcal{R} \\
 & x_i \geq 0, \quad \forall i \in \mathcal{P} \\
 & w_i \geq 0, \quad \forall i \in \mathcal{P} \\
 & y_k \geq 0, \quad \forall k \in \mathcal{R} \\
 & z_k \geq 0, \quad \forall k \in \mathcal{R} \\
 & s_k \geq 0, \quad \forall k \in \mathcal{R},
 \end{aligned} \tag{P1}$$

where \mathcal{P} is the set of products the company manufactures and \mathcal{R} is the set of raw materials used to produce the products; x_i and w_i are the decision variables representing the amount of product i to produce in year one and two, respectively, y_k and z_k are the decision variables representing the amount of raw material k to buy in year one and two, respectively, and s_k is

the amount of raw material k to store from year one to year two; f_i is the net cost associated with producing and selling product i , g_{kj} is the cost of purchasing raw material k in year j , r_k is the amount of raw material k needed to produce a material, and h_k is the cost of storing some amount of raw material k from year one to year two.¹ Additionally, D_{ij} is the demand for product i in year j . The objective is to minimize the total cost of production, purchasing, and storage. The first constraint ensures that demand is met for each product each year, and the second and third constraints ensure that there are sufficient supplies to meet production requirements in the first and second year, respectively. Note that the year one material constraint is a strict equality, since the company stores all unused materials, whereas the second year is an inequality, since too much material could have been stored in the first year.

An important omission from this model is the uncertainty of the year two demands; recall the first constraint from (P1),

$$x_{ij} \geq D_{ij}, \quad \forall (i, j) \in \mathcal{P} \times \{1, 2\}.$$

This constraint treats both year one and year two demands as known parameters; however, only year one demands are known with certainty. The values of the year two demands D_{i2} , $i \in \mathcal{P}$ lie in some uncertainty set \mathcal{D}_i . To model this uncertainty, we assume that customer demand for product i will be one of $D_{i2}^1, \dots, D_{i2}^M \in \mathcal{D}_i$, which occur with probability p_1, \dots, p_M , respectively. This results in M different *scenarios* that must be considered. Allow c_m to be the cost of purchasing materials given that scenario m has occurred, with the convention that an infeasible scenario has infinite cost; then we can consider minimizing our expected cost with the objective

$$\min \sum_{i \in \mathcal{P}} f_i(x_{i1}) + \sum_{k \in \mathcal{R}} g_k(y_{k1}) + \sum_{k \in \mathcal{R}} h_k(s_k) + \sum_{m \in M} p_m c_m. \quad (2.1)$$

Of course, this objective will obtain a value of positive infinity if at least one scenario is infeasible. However, to enforce feasibility constraints on each of the scenarios, we must replace the random variables D_{i2} , $i \in \mathcal{P}$ in (P1) with sets of variables representing the planned course of action for each scenario, as well as constraint ensuring feasibility in each scenario. This results in new formulation of (P1).

$$\begin{array}{llll} \min & \mathbf{f}^T \mathbf{x} + \mathbf{g}^T \mathbf{y} + \mathbf{h}^T \mathbf{s} & + p_1 c_1 & + \dots + p_M c_M \\ \text{s.t.} & \mathbf{x} & & \geq \mathbf{d} \\ & \mathbf{y} - \mathbf{s} & & = \mathbf{r}^T \mathbf{x} \\ & \mathbf{s} & + \mathbf{z}^1 - \mathbf{r}^T \mathbf{w}^1 & \geq 0 \\ & \vdots & \ddots & \vdots \\ & \mathbf{s} & + \mathbf{z}^M - \mathbf{r}^T \mathbf{w}^M & \geq 0 \end{array} \quad (2.2)$$

where

$$c_m = \sum_{i \in \mathcal{P}} f_i(w_i^m) + \sum_{k \in \mathcal{R}} g_k(z_k^m).$$

Notice that with this formulation, each scenario is only loosely coupled to the others via the year 1 variables; for this reason, we refer to year 1 variables as *complicating variables*.

¹The use of function notation here is for clarity only; all functions are assumed to be linear, and soon expressions such as $\sum_i f(x_i)$ will be replaced with the dot product $\mathbf{f}^T \mathbf{x}$.

Without them, each scenario could be solved independently. This is a common situation in *stochastic programming*, or programming under uncertainty, since random variables are often approximated as a number of discrete scenarios. PySP, a component of the Coopr library, offers a toolkit designed to solve such problems. Because each scenario introduces another set of variables, the number of variables in (2.2) often makes the problem computationally intractable. To overcome this, we utilize Benders decomposition [3].

2.2. Formulation. Benders decomposition is a method designed to reduce the number of variables in loosely-coupled problems such as (2.2). The primary goal of Benders decomposition is to create a new *master problem* containing the the constraints and objectives involving the complicating variables, and replacing each set of scenario variables with a single auxiliary variable. Constraints are then added to the master problem to ensure that the auxilliary variables ultimately attain the value of the original objectives. Consider the general linear program

$$\begin{aligned}
 \min \quad & \mathbf{c}^T \mathbf{x} + \mathbf{f}^T \mathbf{y}_1 + \dots + \mathbf{f}^T \mathbf{y}_M \\
 \text{s.t.} \quad & \mathbf{A} \mathbf{x} = \mathbf{b} \\
 & \mathbf{B}_1 \mathbf{x} + \mathbf{D} \mathbf{y}_1 = \mathbf{d}_1 \\
 & \vdots \\
 & \mathbf{B}_M \mathbf{x} + \mathbf{D} \mathbf{y}_M = \mathbf{d}_M \\
 & \mathbf{x} \geq \mathbf{0}
 \end{aligned} \tag{2.3}$$

Like in our motivating example, \mathbf{x} is a vector of complicating variables that loosely couple the various \mathbf{y}_i sets of variables and associated cost components and constraints. In general, the \mathbf{x} variables may be either continuous or discrete. To solve this problem via Benders decomposition, a relaxed master problem is created,

$$\begin{aligned}
 \min \quad & \mathbf{c}^T \mathbf{x} + \mathbf{1}^T \mathbf{z} \\
 \text{s.t.} \quad & \mathbf{A} \mathbf{x} = \mathbf{b},
 \end{aligned} \tag{2.4}$$

where $\mathbf{z} \in \mathbb{R}^m$, along with the subproblems

$$\begin{aligned}
 \min \quad & \mathbf{f}^T \mathbf{y}_m \\
 \text{s.t.} \quad & \mathbf{D} \mathbf{y} = \mathbf{d}_w - \mathbf{B}_w \mathbf{x}^* \\
 & \mathbf{y}_w \geq \mathbf{0},
 \end{aligned} \tag{2.5}$$

where $m = 1, \dots, M$. Note that in the master problem the \mathbf{z} variables are unconstrained; this is not a problem, because or constraint generating techniques will generate the relevant bounding constraints.

At each iteration of the algorithm, the relaxed master problem is solved, and the optimal value \mathbf{x}^* of variable \mathbf{x} is recorded. The optimal variable values of \mathbf{x} are then fixed in each subproblem, which are then solved. If any subproblem m is dual-unbounded² in direction³

²Every linear programming problem has a ‘dual,’ which is another linear programming problem closely related to the original, or ‘primal,’ problem. Briefly, each constraint in the primal corresponds to a variable in the dual, and each variable in the primal corresponds to a constraint in the dual. One can think of the dual variables as penalties applied to the primal for violating constraints. The most notable traits of a linear programming primal-dual pair is that the cost associated with any feasible dual solution acts as a lower bound on the optimal cost of the primal, and that the cost associated with any feasible primal solution acts as an upper bound on the optimal cost of the dual; this is known as *weak duality*. In fact, it can be shown that if either the primal or dual has an optimal solution, then both problems have an optimal solution and the associated costs are the same; this is known as *strong duality*. For a more complete treatment of duality theory, see [4].

³Given a problem with variables $\mathbf{x} \in \mathbb{R}^n$, a *feasible direction* is a vector $\mathbf{d} \in \mathbb{R}^n$ such that $\mathbf{x} + \theta \mathbf{d}$ satisfies the

$\mathbf{v}^{i(m)}$, the constraint

$$\left(\mathbf{v}^{i(m)}\right)^T (\mathbf{d}_w - \mathbf{B}_w \mathbf{x}) \leq 0$$

is added to the relaxed master problem, constraining the master problem to choose stage 1 variables that do not allow that direction of unboundedness to be feasible. If any subproblem m has a dual-optimal solution $\mathbf{p}^{j(m)}$ with an associated cost greater than the value of z_m , the constraint

$$\left(\mathbf{p}^{j(m)}\right)^T (\mathbf{d}_w - \mathbf{B}_w \mathbf{x}) \leq z_w$$

is added to the relaxed master problem, constraining the master problem to choose stage 1 variables that do not allow the optimal cost to be less than the optimal cost of the dual, enforcing the condition usually imposed by weak duality. If neither type of constraint is added during an iteration, the current master variable value \mathbf{x}^* is the optimal stage-1 variable value, and the solutions to each subproblem are the optimal actions to be taken should that scenario occur.

2.3. Representation in Pyomo. To solve a problem via Benders decomposition, knowledge of the duals of the subproblems is required. Specifically, we need to be able to determine both the values of the dual-optimal variables for each subproblem and any directions of dual unboundedness. Although some full-featured solvers, such as the commercial CPLEX solver, are capable of providing such information, the Pyomo modeling language is designed to be solver-independent, and so it cannot rely on this peripheral information to be available. Rather, Pyomo must form the duals of the subproblems and then determine the directions of unboundedness through the solution to an ordinary linear program.

The first step Pyomo must take is to form the duals of each subproblem. It is well known that the primal-dual pair related to problem (1.2) is given by

$$\begin{array}{ll} \min & \mathbf{c}^T \mathbf{x} \\ \text{s.t.} & \mathbf{A} \mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{array} \quad \begin{array}{ll} \max & \mathbf{p}^T \mathbf{b} \\ \text{s.t.} & \mathbf{p}^T \mathbf{A} \leq \mathbf{c}^T. \end{array} \quad (2.6)$$

Note that the vector \mathbf{p} contains a decision variable for each constraint in the matrix \mathbf{A} , characteristic of the dual. Thus, if Pyomo can express a problem in the form of (1.2), it can immediately transform the problem to its dual. Fortunately, transforming a general linear programming problem into the form of (1.2) is straightforward [4]. To accomplish this, the following transformations are made:

1. For each variable y not explicitly constrained to be nonnegative, replace y with $y^+ - y^-$, where $y^+, y^- \geq 0$.
2. For each less-than or equal-to constraint $\mathbf{A}_i \mathbf{x} \leq b_i$, replace the constraint with $\mathbf{A}_i \mathbf{x} + e_i = b_i$, where the excess variable satisfies $e_i \geq 0$.
3. For each greater-than or equal-to constraint $\mathbf{A}_i \mathbf{x} \geq b_i$, replace the constraint with $\mathbf{A}_i \mathbf{x} - s_i = b_i$, where the slack variable satisfies $s_i \geq 0$.
4. If the objective is of the form $\max \mathbf{c}^T \mathbf{x}$, replace it with $\min -\mathbf{c}^T \mathbf{x}$.

constraints of the problem for some positive θ and feasible \mathbf{x} . An *unbounded direction* is a direction \mathbf{d} such that $\mathbf{x} + \theta \mathbf{d}$ is feasible for all positive θ . A direction of improvement is a direction \mathbf{d} such that given a cost vector $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{c}^T(\mathbf{x} + \theta \mathbf{d}) < \mathbf{c}^T \mathbf{x}$ for some positive θ . Thus, an *unbounded direction of improvement* is a direction that can be chosen to produce an optimal cost that is unbounded below.

The next step is to be able to determine the directions of unboundedness in the dual subproblems. If a dual subproblem is solved and found to be unbounded, we solve the auxiliary subproblem (AUX) described in proposition 2.1 to determine the direction of unboundedness.

PROPOSITION 2.1. *Allow a linear programming problem to be given with a dual of the form*

$$\begin{aligned} \min \quad & \mathbf{p}^T \mathbf{b} \\ \text{s.t.} \quad & \mathbf{p}^T \mathbf{A} \leq \mathbf{c}^T \end{aligned} \tag{D}$$

If (D) is unbounded, a direction of unboundedness \mathbf{d}^ is given by an optimal solution to the auxiliary problem*

$$\begin{aligned} \min \quad & \mathbf{d}^T \mathbf{0} \\ \text{s.t.} \quad & \mathbf{d}^T \mathbf{A} \leq \mathbf{0}^T \\ & \mathbf{d}^T \mathbf{b} \geq 1. \end{aligned} \tag{AUX}$$

Thus, by solving these auxiliary subproblems when directions of dual unboundedness are required, Pyomo has all the tools necessary to solve a problem via Benders decomposition using the simplest of linear programming solvers.

3. Conclusions and Future Work. At present, the transformations required to solve a problem via Benders decomposition in Pyomo are memory intensive. As Pyomo is built on an interpreted language, Pyomo already consumes a large amount of computer memory and the addition of these transformations will likely cause Pyomo to be unable to solve large problems.

There are several immediate enhancements that can be made to conserve memory usage in the algorithm outlined above. The motivation for the transformations presented was to overcome the disparity in the abilities between the various solvers supported by Pyomo. By making these transformations, Pyomo is able to function with the least-capable solver available. However, these transformations become unnecessary when attempting to solve a problem with a solver capable of returning both the dual optimal solutions and directions of dual unboundedness. Pyomo can take advantage of this, performing transformations only when the solver is incapable of performing them itself.

Additional enhancements can be made to improve the run-time performance of this technique. Note that only constraints are added to the master problem (2.4); this of course corresponds to adding variables to the dual, and so a dual-optimal solution to (2.4) is guaranteed to be a dual-feasible solution after adding constraints. This allows the solver to be ‘warm started’ using a previous solution, both eliminating the need for a phase-I simplex solve and providing a near-optimal starting basis to the solver, potentially reducing the number of simplex iterations required to solve the problem.

Recall that in section 2.1 we considered a problem with two stages of decisions, leading to our formulation of problem 2.2. We can consider, however, the case where the various scenarios of 2.2 in turn had a similar problem structure; this structure would lend itself to *nested Benders decomposition*, wherein there are k ‘stages’ to consider, and each subproblem is solved via Benders decomposition. Because Pyomo allows entire models to be abstracted behind Block objects, it is possible to create a Benders decomposition method where the each top-level scenario is solved via Benders decomposition, thus allowing an arbitrary number of levels to be solved via Benders decomposition simply by having Block objects contain other Block objects with a structure similar to (2.2).

REFERENCES

- [1] *Coopr Home Page*, 2010. <https://software.sandia.gov/trac/coopr>.
- [2] R. E. BELLMANN, *On a Routing Problem*, Quarterly of Applied Mathematics, 16 (1958).
- [3] J. BENDERS, *Partitioning Procedures for Solving Mixed-Variables Programming Problems*, Numerische Mathematik, 4 (1962), pp. 238–252.
- [4] D. BERTSIMAS AND J. TSITSIKLIS, *Introduction to Linear Optimization*, (1997).
- [5] S. BOYD AND L. VANDENBERGHE, *Convex Optimization*, Cambridge University Press, 2004.
- [6] E. DIJKSTRA, *A Note on Two Problems in Connexion with Graphs*, Numerische Mathematik, 1 (1959), pp. 269–271.
- [7] L. FISHBONE AND H. ABILOCK, *MARKAL, a Linear-Programming Model for Energy Systems Analysis: Technical Description of the BNL Version*, International journal of Energy Research, 5 (1981), pp. 353–375.
- [8] W. HART AND J. WATSON, *Coopr: a Common Optimization Python Repository*.
- [9] W. HART, J. WATSON, AND D. WOODRUFF, *Coopr User Manual: Getting Started with the Pyomo Modeling Language*, (2009).
- [10] N. KARMAKAR, *A New Polynomial-Time Algorithm for Linear Programming*, in Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing, ACM, 1984, p. 311.
- [11] V. KLEE, G. MINTY, AND S. O., *How Good is the Simplex Algorithm?*, 1972.
- [12] M. YOUNG, *A Minimax Portfolio Selection Rule with Linear Programming solution*, Management Science, 44 (1998), pp. 673–683.

Appendix

A. Proof of Proposition 2.1. *Proof.* Suppose that (D) is unbounded, and that \mathbf{d}^* produces the optimal value of (AUX). Then \mathbf{d}^{*T} is feasible to (AUX), and so \mathbf{d}^{*T} satisfies

$$\mathbf{d}^{*T} \mathbf{A} \leq \mathbf{0}$$

$$\mathbf{d}^{*T} \mathbf{b} \geq 1.$$

Recall that for \mathbf{d}^* to be an unbounded direction in (D), it must satisfy

$$(\mathbf{p}^T + \theta \mathbf{d}^{*T}) \mathbf{A} \leq \mathbf{c}^T \tag{C1}$$

$$\mathbf{d}^{*T} \mathbf{b} > 0 \tag{C2}$$

for all $\theta \geq 0$ some feasible \mathbf{p} .

We first note that since \mathbf{d}^* is feasible to (AUX),

$$\mathbf{d}^{*T} \mathbf{b} \geq 1$$

which trivially implies that \mathbf{d}^* satisfies (C2).

We now show that \mathbf{d}^* satisfies (C1). Since \mathbf{p} is feasible to (D), it must be that

$$\mathbf{p}^T \mathbf{A} \leq \mathbf{c}^T \iff \mathbf{0} \leq \mathbf{c}^T - \mathbf{p}^T \mathbf{A}.$$

Then, since $\mathbf{d}^{*T} \mathbf{A} \leq \mathbf{0}$,

$$\theta \mathbf{d}^{*T} \mathbf{A} \leq \mathbf{c}^T - \mathbf{p}^T \mathbf{A}, \quad \forall \theta \geq 0$$

$$\mathbf{p}^T \mathbf{A} + \theta \mathbf{d}^{*T} \mathbf{A} \leq \mathbf{c}^T$$

$$(\mathbf{p}^T + \theta \mathbf{d}^{*T}) \mathbf{A} \leq \mathbf{c}^T,$$

as required.

We will now show that if (D) is unbounded, (AUX) has at least one feasible solution. Allow \mathbf{d}^* to be given such that \mathbf{d}^* is an unbounded direction of improvement in (D). Since

\mathbf{d}^* is an unbounded direction of improvement, it must satisfy constraints (C1) and (C2) from above. Allow $\theta = 0$; then we have that

$$\begin{aligned} (\mathbf{p}^T + \theta \mathbf{0}^T) \mathbf{A} &\leq \mathbf{c}^T \\ \mathbf{p}^T \mathbf{A} &\leq \mathbf{c}^T \\ 0 &\leq \mathbf{c}^T - \mathbf{p}^T \mathbf{A}, \end{aligned}$$

and so all components of $\mathbf{c}^T - \mathbf{p}^T \mathbf{A}$ are positive. Since

$$(\mathbf{p}^T + \theta \mathbf{d}^{*T}) \mathbf{A} \leq \mathbf{c}^T \iff \theta \mathbf{d}^{*T} \mathbf{A} \leq \mathbf{c}^T - \mathbf{p}^T \mathbf{A},$$

we can immediately see that if some component of $\mathbf{d}^{*T} \mathbf{A}$, $(\mathbf{d}^{*T} \mathbf{A})_i$, were positive, by choosing

$$\theta = \frac{(\mathbf{c}^T - \mathbf{p}^T \mathbf{A})_i + 1}{(\mathbf{d}^{*T} \mathbf{A})_i}$$

the previous constraint is violated. Thus, all components of $\mathbf{d}^{*T} \mathbf{A}$ are nonpositive, and so \mathbf{d}^{*T} satisfies the first constraint of (AUX), namely

$$\mathbf{d}^{*T} \mathbf{A} \leq \mathbf{0}.$$

If $\mathbf{d}^{*T} \mathbf{b} \geq 1$, all constraints have been met in (AUX), and so \mathbf{d}^* is feasible to (AUX). We now show that if $\mathbf{d}^{*T} \mathbf{b} = \nu < 1$, a new unbounded direction of improvement $\bar{\mathbf{d}}$ can be constructed such that $\bar{\mathbf{d}}^T \mathbf{b} \geq 1$. Recall that (C1) is satisfied by \mathbf{d}^* for all nonnegative θ ; then $\frac{1}{\nu} \mathbf{d}^*$ also satisfies (C1), since ν is constrained to be positive. Additionally,

$$\begin{aligned} \frac{1}{\nu} \mathbf{d}^{*T} \mathbf{b} &= \frac{1}{\nu} \nu \\ &= 1 \\ &\geq 1, \end{aligned}$$

and so $\frac{1}{\nu} \mathbf{d}^*$ is a feasible solution to (AUX) and an unbounded direction of improvement in (D).

Thus, if (D) is an unbounded problem, (AUX) has at least one feasible solution which is an unbounded direction of improvement in (D). \square

B. Pyomo Formulation. Section B.1 shows the Pyomo code representing problem 2.2, excluding the data necessary to instantiate the problem. Note how each scenario in 2.2 is represented as a discrete Block object, keeping the Pyomo representation of the problem similar to the original formulation. Section B.2 shows the PySP code necessary to solve the Pyomo model via Benders decomposition.⁴

B.1. Model.

```
from coopr.pyomo import *
```

```
#
```

```
# Set up the common data
```

⁴Please note that as CoopR continues developing, the exact syntax of the Pyomo formulation and PySP solver code may change. For up-to-date examples of Pyomo and PySP code, please refer to [1].

```

#

master = Model()

master.PRODUCTS = Set()
master.MATERIALS = Set()
master.SCENARIOS = Set()

#
# Stage 1 problem
#

master.stagel = Block()
master.stagel.costs = Param(master.MATERIALS)
master.stagel.buy = Var(master.MATERIALS)
master.stagel.produce = Var(master.PRODUCTS)
master.stagel.store = Var()

master.stagel.buy_con = Constraint(master.PRODUCTS)
master.stagel.store_con = Constraint(master.PRODUCTS)
master.stagel.prod_con = Constraint(master.PRODUCTS)

#
# Stage 2 problems
#

master.scenarios = Block(master.SCENARIOS)

master.scenarios.costs = Param(master.MATERIALS,
                                master.SCENARIOS)

master.scenarios.buy = Var(master.MATERIALS)
master.scenarios.produce = Var(master.PRODUCTS)

master.scenarios.buy_con = Constraint(master.PRODUCTS)
master.stagel.prod_con = Constraint(master.PRODUCTS)

```

B.2. PySP Solver Code.

```

from coopr.opt.base import SolverFactory
from pyutilib.misc import import_file
from coopr.pysp import Benders

solver = SolverFactory("glpk")

problem = import_file("benders.py").model
instance = problem.create("some_data_file.dat")

# Specify the Block holding the complicating variables;
# in this case, instance.stagel

```

```
solution = Benders(instance , instance.stage1).solve()  
print solution
```

STOCHASTIC OPTIMIZATION APPLIED TO ENERGY ECONOMY OPTIMIZATION MODELS

KEVIN HUNTER[¶], JOSEPH DECAROLIS*, SARAT SREEPATHI*, AND JEAN-PAUL WATSON^{||}

Abstract. Addressing future uncertainties is a key challenge with multi-decadal optimization of energy and emissions projections. A common approach creates a small set of projections based on different exogenous scenario assumptions to quantify the range of possible outcomes. While such scenario analysis can help expand thinking about how the future might unfold, such projections are of limited value to policy makers, who must make long-lived decisions before uncertainty is resolved. While stochastic optimization has been applied to energy system models, the computation requirements have limited analysis to relatively simple probability trees. However, multi-stage stochastic optimization enables a more sophisticated approach by embedding the probability of different outcomes within model formulation, yielding a hedging strategy that accounts for future uncertainties.

1. Introduction. Over the next few decades, concerns over climate change and energy security will drive fundamental changes in the global supply, transport, and use of energy. Energy system and integrated assessment models play a vital role in the planning process by providing insight related to the future impacts of policies and technology deployment. Energy economy optimization (EEO) models attempt to optimize (planned) energy system development over a time horizon, often with an objective function that minimizes the total cost or maximizes social utility of energy supply. Since the early 1990s, EEO models have emerged as key tools for analysis of energy and climate policy at the regional, national, and international scale. Given advances in computing and the ever expanding core of mineable energy and climate data, the scope and complexity of EEO models has expanded tremendously.

The application scope widening is a natural consequence of the power of optimization modeling in general, but it remains to be seen whether increased model complexity buys a modeler more objective insight. The thought goes that there is a positive correlation between more complex models and more accurate future energy projections. However, accurately modeling the future is akin to predicting the future, and little analysis is currently performed on the likelihood of different outcomes. EEO modeling should provide robust results in the face of future uncertainty; they have little value to policy makers otherwise.

2. An Open Source Energy Economy Optimization Model. Personal observation and a survey of the literature have led us to identify two critical shortcomings regarding existing EEO models. With few exceptions, the models are not open source.¹ Because EEO models necessarily have long timeframes, expansive system boundaries, and encompass both physical and social phenomena, the level of descriptive detail that can be provided in model documentation and in peer-reviewed journals is insufficient to reproduce a specific set of published results. While there have been rigorous efforts to compare model results², the lack of access to source code prevents a deeper level of external verification. Peer-reviewed journal articles, model documentation, and distribution of executable models provide guidance, but without the ability for researchers to investigate internal model equations and assumptions, insight into specific model results is cursory, at best. Dubois noted in 2002 that replication and verification of scientific models can only be accomplished by freely available source code.[2] Making EEO models open source, freely available, and electronically attainable solves both the external replication and verification problems.

[¶]North Carolina State University, [kmhunte2, jdecarolis, sarat.s]@ncsu.edu

^{||}Sandia National Laboratories, jwatson@sandia.gov

¹The (recent) exceptions include Yale's DICE model <http://nordhaus.econ.yale.edu/>, and the OSE-MOSYS model <http://osmosys.yolasite.com/>

²e.g. the Stanford Energy Modeling Forum, <http://emf.stanford.edu/>

A second critical shortcoming of current EEO models is their treatment of uncertainty. Either intentionally, or through project “feature creep,” a major trend in current EEO models is toward large, complex models that endogenize all possible phenomena. The drawback is that large and complex models are computationally expensive to iterate, which makes uncertainty analysis difficult at best to perform. Further, given the multi-decadal time horizons of EEO models, validation through comparative observation to real world phenomena is not practical. This suggests that modelers have little ability to assess the strength of model features toward model performance. Consequently, complex models are mainly used to create highly detailed scenarios, keeping their applicability limited in scope, and often excluding consideration of insights EEOs may offer for energy system analysis.[6].

In response to these issues, we introduce a new modeling framework called Tools for Energy Model Optimization and Analysis. TEMOA is a technology explicit, partial equilibrium EEO model that minimizes the present system-wide cost of energy supply by optimizing energy capacity installations and commodity flows to end-use demands. Technologies are divided into three basic categories: resource supply, intermediate transformations, and demand technologies.

Energy prices and flows are determined endogenously such that end use demands are matched exactly by energy supply. The intersection of the inverse demand curve and supply curve represents equilibrium, and this equilibrium is calculated for every commodity in the energy system. Exogenously specified fixed demands will be used initially in TEMOA and represent a vertical demand curve. In this case, the equilibria are not responsive to changes in demand based on commodity prices. Elasticities can be specified in order to make demand responsive to price, but in either case, the model still represents a partial equilibrium on energy markets because it does not take into account broader economic effects and the associated feedbacks outside of the energy system. For a detailed explanation of the economic rationale behind partial equilibrium models, see Loulou et al. (2004).[5]

2.1. Formulation as a Linear Program. The initial development phase of TEMOA has focused on creating a simplified version that represents the energy system as a linear programming (LP) model. LPs are restricted to have *only* linear objective functions and constraint equations and represent a well-studied class of optimization problems. For a feasible set of assumptions (input parameters), LP algorithms guarantee finding the optimal solution.

The TEMOA objective function is provided below and represents the present cost of supplying energy over the model time horizon (C_{tot}):

$$\begin{aligned} \text{Icost}(p, t, v) &= \left(C_{i(t,v)} \cdot \left[\frac{r_{i,t}}{1 - (1 + r_{i,t})^{-r_{i,t}}} \right] \cdot \text{imat}_{t,v,p} + C_{f(t,v,p)} \right) \cdot \text{Cap}_{t,v} \\ \text{Ucost}(p, t, v) &= C_{m_{t,v,p}} \cdot \text{vmat}_{t,v,p} \cdot \text{Util}_{t,v,p} \\ C_{tot} &= \sum_p \sum_t \sum_v \sum_{y=0}^{t_p+1-t_p} (\text{Icost}(p, t, v) + \text{Ucost}(p, t, v)) \cdot \left[\frac{1}{(1 + r_g)^{t_p+y-t_0}} \right] \end{aligned}$$

where the parameters are defined in Fig. 2.1.

There are two driving sets to this equation: model time **p**eriods and **t**echnologies. There is also the **v**intage set, but the model internally creates the **v**intage set (investment period) as an upper triangle of **p**eriods \times **p**eriods for each technology; the model uses the first period axis to represent the period in which a technology was built – the “vintage”. The decision variables *Capacity* and *Utility* represent the model’s decision to build capacity of a specific technology in a certain year, and the model’s decision to utilize that capacity. The bracketed terms are the standard economics multipliers to annualize the technology investment costs C_i and to convert to present-day value. Other parameters include the marginal cost of technology

$C_{i(t,v)}$	Investment cost for a technology and vintage
$C_{f(t,v,p)}$	Fixed cost for a technology vintage, in a period
$C_{m,t,v,p}$	Marginal cost of operation of a technology vintage in a period
$imat_{t,v,p}$	Binary matrix; tracks technology loan investment lifetime by vintage
$vmat_{t,v,p}$	Binary matrix; tracks technology lifetime by vintage
$Cap_{t,v}$	Variable; installed technology capacity by vintage
$Util_{t,v,p}$	Variable; technology vintage utilized by period
$r_{i,t}$	Technology investment rate
r_g	Global discount rate

FIG. 2.1. Parameters for the TEMOA objective function.

operation C_m , the technology-specific interest rate of the investment r_i and loan life r_l , and finally the global discount rate r_g . The precalculated $imat$ and $vmat$ multipliers are sparse, three dimensional, binary matrices to track the investment period and technology lifetime by vintage.

Beyond the basic nonnegativity constraints on the variables, there are three basic constraint equations:

- A technology-level constraint set ensures that commodity inputs are at least equal to commodity outputs for each period.
- A technology-level constraint set ensures that production cannot exceed the installed capacity for each period.
- A set of constraints enforces that all end use demands are met by production from appropriate demand technologies for each period.

With the variables, objective function, and constraints defined, instantiating an LP problem from this model merely requires specification of each parameter.

2.2. Motivation for Stochastic Program Formulation. “Merely” is deceptive, however, as parameter specification is the crux of energy modeling. In the context of EEO models, LPs implement a “perfect foresight” expectation of a problem instance; to solve an LP, every input parameter must be fully realized. Unfortunately, many real world problems have parameters that cannot be specified until they are realized. For example, what will be the price of fuel, or the efficiency rating of coal power plants in 2030? (Will we even have coal power plants?) Without perfect foresight, it is impossible to know the route to an optimal solution.

Stochastic programming (SP) is similar to linear programming, but more formally addresses inherent model uncertainty. SPs incorporate data into the objective function and constraints that may be uncertain. Though real-world problems have unknown coefficients (e.g. price of natural gas 10 years in the future), these can often be described via probability distributions. For instance, though we do not yet know the price of natural gas one year in the future, we exactly know the current price. We also have a conditional probability range for the price next year, based on a range of mineable data inputs (e.g. historical usage, planned business ventures, policy incentives). In this vein, we can assign a probability that the price will increase (for example) by $\pm 5\%$. Though we still do not know how the price will change, we can at least take action now to minimize a worst-case scenario next year.

One way to formulate a stochastic program is to break it up into two parts – what we can decide now and what we can do after the consequences of the decision are known. This method is called a “recourse” model. More generally, recourse models try to minimize the cost of the *non-anticipative* stages and expected cost of the “recourse” decision[1]. Sub problem 2.2 is the recourse:

$$\begin{aligned} \min \quad & \mathbf{c}^T \mathbf{x} + E_{\xi} Q(\mathbf{x}, \xi) \\ \text{s.t.} \quad & \mathbf{A} \mathbf{x} \leq \mathbf{b} \end{aligned} \quad (2.1)$$

$$\begin{aligned} Q(\mathbf{x}, \xi) = \min \quad & \mathbf{q}^T \mathbf{y} \\ \text{s.t.} \quad & \mathbf{W} \mathbf{y} = \mathbf{h} - \mathbf{T} \mathbf{x} \\ & \mathbf{y} \geq 0 \end{aligned} \quad (2.2)$$

Here, the \mathbf{x} vector denotes the *non-anticipative* variables for which all input parameters \mathbf{c} and all constraint coefficients \mathbf{A} and \mathbf{b} are known. The subproblem contains the recourse decision variable \mathbf{y} , the not-yet realized (*anticipative*) input parameters \mathbf{q} , constraint coefficients \mathbf{h} and \mathbf{T} , and the outcome of the \mathbf{x} decision as the new decision coefficient matrix \mathbf{W} . E_{ξ} denotes mathematical expectation with respect to ξ .

In less formal math language, this basically means that an SP will optimize the variables it can (the non-anticipative stages), and try to find a best preparatory step for minimizing the future cost over the range of possible scenarios (*expected* future cost). Though the SP is “aware” of the future possibilities, it must make a decision without knowledge of which future scenario will actually occur: this is the *non-anticipatory property*. All future stages can be resolved only when more information about them is realized, as by waiting a year.

Suppose a coal power plant is tasked with keeping a city’s lights on. The plant furnace can acquire coal from either its own storage facilities or by purchasing coal from a coal mine. Coal from the mine costs a certain amount normally, but during higher demand seasons, it might be more. It’s a normal season this year, but given past weather data, it might also be warmer or colder next year. Due to overhead, coal costs \$0.04/lb/year to store. Finally, the city’s general demand for electric lighting will increase during warmer or colder years. (This information is summarized in Table 2.1.) Given that next year’s average temperature is unknown, how should the coal plant plan its purchases?

Scenario	Description	Probability	Coal Cost (\$)	Demand (units)
1	Normal	$\frac{1}{5}$	0.50	100
2	Warmer	$\frac{2}{5}$	0.55	110
3	Colder	$\frac{2}{5}$	0.70	140

TABLE 2.1
Example problem data summary

One method of solution replaces the expectation $Q(\mathbf{x}, \xi)$ in the initial problem with a weighted sum of the second-stage decision variables y_i for each possible scenario. This method is called the *extensive form* because it exhaustively states all possible scenarios in the original problem:

$$\begin{aligned}
 \min \quad & 0.50x + 0.04 \cdot (s_1 + s_2) + \frac{1}{5} \cdot 0.50y_1 + \frac{2}{5} \cdot 0.55y_2 + \frac{2}{5} \cdot 0.70y_3 \\
 \text{s.t.} \quad & x - s_1 = 100 \quad (a) \\
 & y_1 + s_1 - s_2 = 100 \quad (b) \\
 & y_2 + s_1 - s_2 = 110 \quad (c) \\
 & y_3 + s_1 - s_2 = 140 \quad (d) \\
 & x, s_1, s_2, y_1, y_2, y_3 \geq 0
 \end{aligned} \tag{2.3}$$

Where x is number of coal units to purchase this year, y_n are the number of coal units to purchase for the respective scenarios in the following year, and s_n are the amount of coal units to put into storage in year 1 and 2. Constraint (a) ensures that demand is met in the first year, and that any excess is put in to storage. Constraints (b), (c), and (d) ensure similar for year two, but also take into account the left over storage from the first year. (Note that s_2 may “intuitively” be 0 in the optimal solution, but this problem may more easily be extended to multiple years.) The extensive form possible outcomes are summarized in Table 2.2.

Stage (Scenario)	Bought	Stored	Cost (\$)
1	200	100	104.00
2 (Normal)	0	0	0.00
2 (Warmer)	10	0	5.50
2 (Colder)	40	0	28.00

TABLE 2.2
Example solution summary

The optimal *expected* cost is $104.00 + \frac{1}{5} \cdot 0 + \frac{2}{5} \cdot (5.50 + 28.00) = \117.40 over the two years.

Stochastic programs can also be thought of in terms of scenario trees. Each leaf node of the tree represents a possible scenario. For a simple 3 stage problem with 1 random parameter with 3 discrete outcomes, the tree is small at 9 (3^2) scenarios (visually depicted in Figure 2.2). But if there are x random variables, each with y possibilities, to be solved over z stages, that is $y^{x^{z-1}}$ leaf nodes or scenarios.

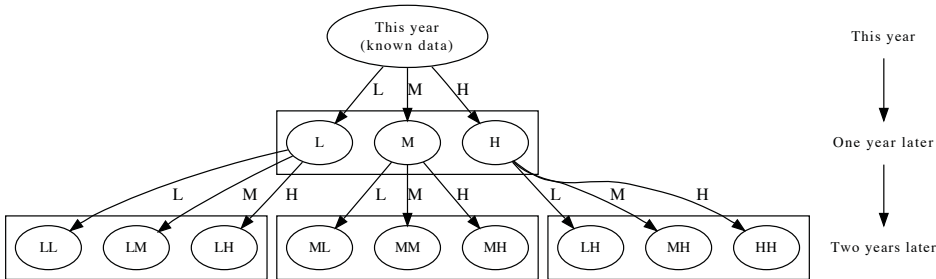


FIG. 2.2. Example single random variable 3-stage scenario tree

For EEO models that necessarily observe long time horizons and have between tens and thousands of variable parameters, this represents a very real obstacle to calculating useful results in a timely fashion.

2.3. Formulation as a Stochastic Program. While each constraint of the LP version of TEMOA ensures that the model meets demand or production requirements at all periods, the objective function minimizes the cost over the entire model time horizon. As time is a natural stochastic partition point due to future uncertainty, we chose to remove perfect foresight from TEMOA by converting the singular objective function into the sum of a set of “sub-objectives”, indexed by the model period. Each sub-objective is aware of its model period, the model decisions of the previous periods, and the expectation of future outcomes. We implement this simplistically through the use of an additional period-indexed variable \mathbf{pc} :

$$\mathbf{pc}_{k \in \mathbf{p}}$$

$$\begin{aligned} \text{Icost}(k, t, v) &= \left(C_{i(t,v)} \cdot \left[\frac{r_{i,t}}{1 - (1 + r_{i,t})^{-r_{i,t}}} \right] \cdot \text{imat}_{t,v,k} + C_{f(t,v,k)} \right) \cdot \text{Cap}_{t,v} \\ \text{Ucost}(k, t, v) &= C_{m_{t,v,k}} \cdot \text{vmat}_{t,v,k} \cdot \text{Util}_{t,v,k} \\ \mathbf{pc}_k &= \sum_t \sum_v \sum_{y=0}^{t_{k+1}-t_k} (\text{Icost}(k, t, v) + \text{Ucost}(k, t, v)) \cdot \left[\frac{1}{(1 + r_g)^{t_k+y-t_0}} \right], \\ &\quad \forall k \in \mathbf{p} \end{aligned}$$

Note that this reformulation removes the outer sum of the LP version for each \mathbf{pc} index, but replaces the entire objective function with this new sum:

$$C_{tot} = \sum_{k \in \mathbf{p}} \mathbf{pc}_k$$

Finally, to make it stochastic, we must specifically describe which stages are anticipatory, and which parameters are to be treated stochastically (to be decided by the modeler on a per-run basis). In reference to equations 2.1 and 2.2, the anticipatory stages relate to \mathbf{x} and the non-anticipatory stages relate to \mathbf{y} . Strictly speaking, this reformulation is not necessary, but it splits up the actions of the model into logical parts (for cognitive purposes), and eases the next step to describe the stochastic problem to the computer modeling system. We give a brief code snippet of how to do this in the next section.

2.4. Modeling Environment. *Algebraic modeling languages* (AML) are computer languages used to describe optimization problems. There are a plethora of AMLs available, but EEOs have historically favored AMPL³, AIMMS⁴, or GAMS⁵. The modeling language choice is a critical decision because it heavily shapes interaction with developers and users alike. We have chosen to use Python Optimization Modeling Objects (Pyomo), a package that provides capabilities often associated with more well-known languages such as AMPL, AIMMS, and GAMS. As the Pyomo application programmer interface (API) is written in the high-level programming language Python, any model written with it gains access to a number of Python advantages. Among other boons, Python offers cross-platform portability and, given the popularity of Python in the scientific community, there exist a large number of libraries for nearly any task.[3]

³A Mathematical Programming Language, <http://www.ampl.com/>

⁴<http://www.aimms.com/>

⁵General Algebraic Modeling System, <http://www.gams.com/>

While Pyomo affords a powerful programming environment for mathematical optimization problems, it lacks the compact nature of pure AMLs, which were specifically designed with a concise syntax that closely mimics mathematical notation. Despite its verbosity, Pyomo is intuitive and uses the same model structure (i.e., sets, parameters, variables, equations) as AMLs. The example below is the Pyomo version of the TEMOA objective function from the SP formulation above:

```
def AnnualCost ( per, model ):
    M = model
    cost = 0.0

    for t in M.tech_new:
        for i in M.invest_period:
            if (t, i, per) in M.investment:
                # If we built it in a recent time period, need to finish
                # paying off that loan;
                cost += ( M.period_spread[ per ] *
                        M.xc[t, i]
                        * ( M.investment_costs[t, i, per]
                          * M.loan_cost[ t ]
                          + M.fixed_costs[t, i, per] )
                )
            else:
                # otherwise, if it's still operational, we just need to pay
                # the operating costs (fixed O&M).
                cost += (
                        M.period_spread[ per ]
                        * M.xc[t, i]
                        * M.fixed_costs[t, i, per]
                )
        # Finally, how much capacity did we use? Have to pay for that too.
    cost += sum( [
        M.xu[t, i, per]
        * M.marg_costs[t, i, per]
        * M.period_spread[ per ]

        for i in M.invest_period
        for t in M.tech_all
    ] )

    return cost
```

A secondary benefit of coding TEMOA against the Pyomo API is the tie-in with a sister project called PySP. PySP extends Pyomo to support multistage stochastic programs with enumerated scenarios.[4] The PySP package enables a fairly simple declaration of the structure of an SP, and automatically handles many of the low-level solving details. Though each node in a scenario tree requires unique input data (for each stochastic parameter's probability distribution), much of the tedium can be outsourced to a generating script.

To understand how to create a stochastic program with PySP, here is a simple 2 stage scenario tree. For brevity, this snippet only illustrates how to set up the stage variables and associated model variables.

```
param StageCostVariable :=
    p2000 PeriodCost[2000]
    p2010 PeriodCost[2010]
;

set StageVariables[p2000] := xc[:,2000] xu[:,2000,2000] ;
set StageVariables[p2010] := xc[:,2010] xu[:,2000,2010] xu[:,2010,2010] ;
```

3. Future Work. The general design philosophy of TEMOA is to make the model just complex enough to answer specific questions, but no more. The goal is to keep the model

lightweight in order to facilitate rigorous uncertainty analysis. We have only very recently begun working with stochastic programming and PySP, but, with limited computing resources available to us, we have already identified a challenge in regards to the sheer size of problems. We have work ahead of us in learning how to “prune” our scenario trees. Relatedly, scenario tree generation may prove to be an area of modeler workflow optimization in that we currently manually modify our stochastic parameters. There is work underway to automate and script scenario tree generation.

Making more efficient use of resources is fast becoming an issue. Currently we are only developing in singleton runs and batch sets of output. However, as we migrate to a stochastic model, the computational power increase predicted by Moore’s law will soon not be enough to solve our model in a reasonable amount of time. The problem of a mushrooming number of scenarios and choices will quickly necessitate a less naive approach to a model solution.

In a larger sense, uncertainty analysis is only one component of the project. We also are working on a general EEO database, that we hope can be a basis and central repository of interaction for a number of different EEO models. Short of time travel, true validation of EEO model results in a timely fashion will never be possible. However, interacting with multiple EEO models through a central hub of information may provide a form of relative validation.

REFERENCES

- [1] J.R. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer Verlag, 1997.
- [2] P. Dubois. Key Techniques for Open-Source Science. In *Coalition for Academic Scientific Computation (CASC) Seminar*, March 2002.
- [3] W.E. Hart. Python Optimization Modeling Objects (Pyomo). *Operations Research and Cyber-Infrastructure*, pages 3–19, 2009.
- [4] D Woodruff J Watson. PySP Version 1.1, User Documentation. <https://software.sandia.gov/trac/coopr/wiki/PySP>, 2010.
- [5] R. Loulou, G. Goldstein, K. Noble, et al. Documentation for the MARKAL Family of Models. *IEA Energy Technology Systems Analysis Programme. Paris*, 2004.
- [6] M.G. Morgan, M. Henrion, and M. Small. *Uncertainty: a guide to dealing with uncertainty in quantitative risk and policy analysis*. Cambridge University Press, 1990.

AN ACRO IMPLEMENTATION OF THE HYBRID OPTIMIZATION ALGORITHM EAGLS

JACOB W. ORSINI^{||} AND GENETHA A. GRAY ^{**}

Abstract. EAGLS is a hybrid optimization which combines the global functionality of genetic algorithms with the speed of a local direct search. This novel approach to hybridization has shown promise when used on various benchmarking and real world applications. Due to this, there is a need to implement the algorithm in a widely available package, such as Sandia's DAKOTA, in order to allow public use. The task of implementing EAGLS into DAKOTA is most easily done by using an existing front end available in DAKOTA. This front end, ACRO, has built in libraries and interfaces which allow for developers to use pre-built optimizers in order to test their own problems and create custom hybrid schemes.

1. Introduction. Comprehensive studies of complex systems in science and engineering benefit from the inclusion of physical experimentation. However, in many cases, such experiments are prohibitively expensive or even impossible to perform. To overcome such limitations, scientists and engineers take advantage of the fact that the behavior of many of these systems can be imitated by computer models and use computer simulations as an alternative or augmentation to experimental data. Such simulations created a need for tools to help assess and improve the predictive capabilities of numerical models. This can be achieved by pairing the simulations with optimization models.

Many of these simulation-based optimization problems contain both continuous and integer or categorical variables. In fact, within Sandia's nuclear weapons program, there is an abundance of mixed variable optimization problems. For example, a discrete variable may represent a material type or the number widgets in a design. Mixed variable models are also prevalent in the area of critical infrastructure such as the electrical grid, highway system, and water supply system. Therefore, the research and development of optimization algorithms that can handle MINLPs has become a focus.

It should also be noted that within the simulation-based framework, the objective function to be minimized is complicated in the sense that derivatives are unavailable and approximate derivatives are often unreliable. Moreover, the objective function landscapes can be disconnected, non-convex, non-smooth, or contain undesirable, multiple local minima due to the complicated physical phenomena being modeled. Therefore, a variety of derivative-free optimization methods have emerged and matured over the years to address simulation-based problems in general. They are theoretically supported and have established convergence criteria.

The evolutionary algorithm guiding local search, EAGLS, has been developed to address derivative-free MINLPs of the form

$$\begin{aligned} \text{minimize } & x \in \mathbb{R}^{nr}, z \in \mathbb{Z}^{nb} & f(z, x) \\ \text{subject to } & & c(z, x) \leq 0 \\ & & z_\ell \leq z \leq z_u \\ & & x_\ell \leq x \leq x_u. \end{aligned} \tag{1.1}$$

where $c(x) : \mathbb{R}^n \rightarrow \mathbb{R}^m$. This method has shown to be successful in common hydrology problems and shows promise among other standard tests. Due to this, it would be beneficial to make the EAGLS available for a larger scale of use by implementing it within Sandia's Design Analysis Kit for Optimization and Terascale Applications (DAKOTA). DAKOTA was chosen since it provides a flexible, extensible interface between analysis codes and iterative

^{||}Clarkson University Department of Mathematics and Computer Science, orsiniwj@clarkson.edu

^{**}Sandia National Laboratories, gagray@sandia.gov

systems analysis methods [1]. This task is most easily accomplished by leveraging a common repository for optimizers (ACRO), which is a user friendly front end for solving optimization problems in science and engineering [5]. Built into this software are robust versions of common optimization algorithms, which can be executed singly or in parallel by a user. This functionality allows for the implementing of custom hybrid schemes, such as EAGLS.

The goal of this project is to implement the EAGLS algorithm using optimizers currently built into the ACRO package. This paper describes this process as follows. First, in Section 2, we give an overview of the EAGLS algorithm, focusing on the characteristics which are most important in its implementation. Section 3 summarizes the ACRO package, and Section 4 describes how EAGLS was implemented as an ACRO scolib package. Finally, Section 5 gives some general comments and describes future work.

2. EAGLS. Derivative-free optimization via evolutionary algorithms guiding local search, EAGLS, provides a novel way creating an asynchronous parallel hybrid optimization scheme as seen in [7]. In the past, derivative free optimization methods have relied on relaxation techniques in order to approximate discrete variables, such as integers, using a similar continuous function. Due to the fact that an approximation is being used it is possible that the answer obtained may not be the best possible answer for the discrete case. EAGLS combines an existing heuristic algorithm and direct search algorithm in order to efficiently search for local optima. The heuristic algorithm used by EAGLS is the non-dominated sorting genetic algorithm (NSGA-II) as described in [2]. By using this genetic algorithm, EAGLS is able to search the entire domain for potential optimal area without having to rely on derivatives. However, using a GA on its own would require large amounts of computational power in order to fully run and get a set of solution points. To alleviate this, the EAGLS algorithm uses the asynchronous parallel pattern search (APPS), as described in [3], in order to refine the points found by NSGA. These two algorithms were chosen for this application as both have been widely tested and are easily interfaced with other methods. This allows NSGA to focus on handling integer variables and global searches while APPS runs local searches with real variables.

2.1. Description of Algorithm. The algorithm shown in Alg 8 is a synchronous version of EAGLS which demonstrates the algorithm's core functionality. In practice, EAGLS is implemented asynchronously in order to distribute computational loads evenly. Step 3 through step 7 is the classical implementation of a genetic algorithm. Step 8 and Step 9 are the local search portion of EAGLS. Points of synchronization occur at lines 1 and 5, where new points of the GA are evaluated, as well as Step 9, when instances of APPS are called. To alleviate the potential load imbalance that may occur at these steps a shared evaluation queue is used so that the local search can run while waiting for the GA to complete its evaluations. A more complete example how EAGLS works can be seen in Figure 2.1.

2.2. Benefits of EAGLS. The real power behind the EAGLS hybrid is that the algorithm allows NSGA-II and APPS to work together while focusing in on what each algorithm does best. In doing this, the GA is able to pick promising areas for global optima and stop before its own function evaluations begin to grow. After points are picked by the GA, the local search then refines them quickly and sends the points back into the population in order to be ran through the algorithm again. In [7], EAGLS is tested on a compression spring model, a simulation-based hydrology application, and a standard mixed integer test problem. These problems and results can be found in [7] and its references. Though testing and additions need to be made for EAGLS, its numerical results are promising. Along with this, the solutions found using EAGLS are comparable to those found using methods in each problems' literature.

Algorithm 8 Genetic Algorithm Guiding Local Search

Require: Population size: n_p
Require: Maximum number of generations: n_g
Require: Budget for local search: n_b
Require: Number of parallel local searches desired: n_s

- 1: Generate (evaluate in parallel) and rank initial population: $P_1 = p_1, \dots, p_{n_p}$
- 2: **for** $k = 1, \dots, n_g$ **do**
- 3: $P_{k+1} = \text{select}(P_k)$
- 4: $P_{k+1} = \text{mutate}(P_{k+1})$
- 5: Evaluate in parallel new points in P_{k+1}
- 6: $P_{k+1} = \text{merge}(P_k, P_{k+1})$
- 7: $P'_{k+1} = \text{rank}(P_{k+1})$
- 8: Choose first n_s of P'_{k+1} for local search
- 9: Create n_s instances of APPS for $1 \leq i \leq n_s$ subproblems:

$$\begin{array}{ll} \text{minimize} & f(x, \text{int}(p'_i)) \\ & x \in \mathcal{R}^{n_r} \\ \text{subject to} & x_\ell \leq x \leq x_u \end{array}$$
- 10: **while** number of evaluations $< n_b$ **do**
- 11: Run APPS instance in parallel with parallel evaluations
- 12: **end while**
- 13: **end for**
- 14:

3. ACRO. A common repository for optimizers (ACRO) is an open source optimization package developed and maintained by Sandia National Laboratories. ACRO is included in the DAKOTA optimization package under the name Coliny for distribution to the public. ACRO is the combination of previous optimization libraries developed by Sandia. These include the common optimization library interface (COLIN) and the Sandia COLIN optimizer library (SCOLIB), both originally developed by William Hart [6] and maintained by John Sirola. These two libraries contain both first and third party optimizers to be used with the ACRO interface. In addition to optimizers, the COLIN library contains package used for managing execution of the objective function, handling of input and output, reading input from XML files or AMPL methods as well as other background tasks which are required for ACRO to run. Users are able to communicate with the ACRO package using either XML or AMPL via the Coliny interface. Either method simply passes an objective function, what optimizer is to be used, and whatever constraints and initial values are needed. With this input, Coliny will handle running of ACRO automatically. Coliny also allows for the user to specify multiple optimizers in order to create a hybrid scheme, provided the two optimizers have compatible domains. One of the most notable libraries built into ACRO is UTILIB, a library of C and C++ utility methods similar to the Boost libraries [4]. UTILIB contains generic data types which allow for optimizers and objective functions that may not have perfectly compatible data types to talk to each other, as the data goes through a generic typecasting. With a deep understanding of the aforementioned libraries, it is possible for a user to customize or develop their own solvers. In this case, we will be implementing EAGLS into the ACRO package.

4. Implementing EAGLS into ACRO. In order to implement EAGLS into ACRO it was necessary to have a full understanding of some specific libraries built into ACRO. The PointSet package is used by ACRO in order to receive and set points used by each individual optimizer. Methods within PointSet are able to get the final point, which would most likely be the best answer, or set the initial point to be used in an optimizer call. In either of these

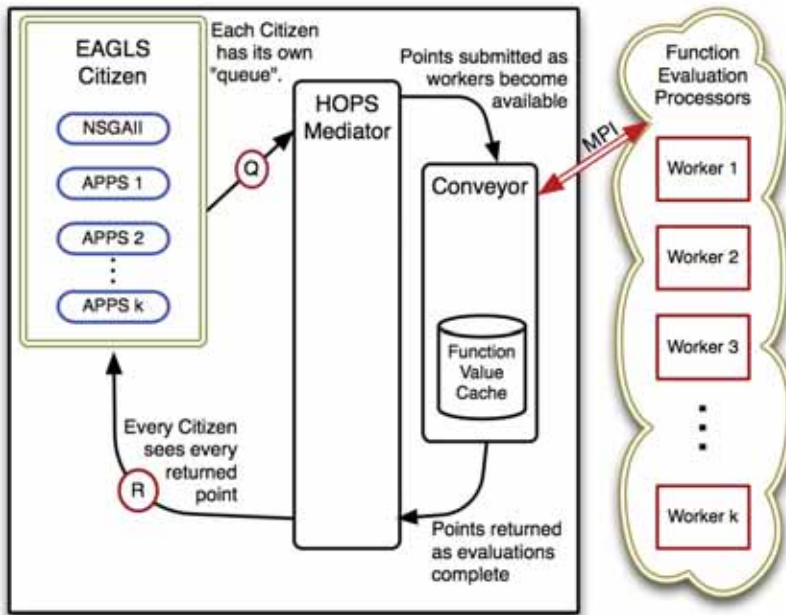


FIG. 2.1. EAGLS execution

cases, the points are stored in an appropriate container using ACRO's generic typecasting from the UTILIB library. The solver manager library, SolverMngr, allows for optimizers to be easily called, whether it be by the Coliny interface directly or within another block of code. In the case of EAGLS, the pattern search (PS) and generic evolutionary algorithm (EA) in the SCOLIB library will be called from within a parallel framework using methods from the aforementioned packages, without the worry of how each optimizer handles points themselves. These two optimizers were chosen due to their resemblance to APPS and NSGA-II, which EAGLS was designed to use. Parallelism is done using the message passing interface (MPI). MPI allows for the EA to be set as a master node which can call a number of slave nodes based on how many available processors are defined by the user.

For actual implementation, only a small portion of the code shown in Alg 8 needs to be considered. As mentioned previously, the optimizers used will be taken directly from ACRO, so the extent of their implementation will be method calls and data passing. Input and output is also handled directly by ACRO and require that the EAGLS package call the correct data members which hold the data passed by the input file. In order to make the program asynchronous, a queue must be present in order to handle data going to the direct search. For the time being, a queuing system built into ACRO will be used until a more specialized one can be developed. The cache used to hold points found by the pattern search until they are required to be used by the EA will be implemented similarly to the queue. To implement a parallel, most of the necessary code and algorithms for MPI are widely available and just need to be looped over in order to produce the same results as in EAGLS. Over the summer a basic framework was created, but needs to have further development before any tests can be ran on the new implementation of EAGLS. The full implementation could not be completed due to the fact that several functionalities within ACRO which are necessary for EAGLS to run are untested or have not been updated in recent releases. There are also several built features

within ACRO, such as a data queue, which could serve the needs of EAGLS but are not as powerful as the original EAGLS implementation. A fully functional implementation will be due out in late fall.

5. Future Work. Currently, EAGLS is under further development to include more features such as more appropriate stopping conditions and advanced constraint handling. Having output with uncertainty quantification, so a user can be aware if there is any potential error in their answer, is also planned. When these additions are made to the main EAGLS code, it will be necessary to implement them in the EAGLS package within ACRO as well. It will also be necessary to fully test the ACRO implementation of EAGLS against problems similar to those that the EAGLS algorithm was tested on. Once this is done, a functional framework will exist which will allow for more hybrid schemes to be easily created within ACRO. The way the code is currently written will allow this simply by swapping one optimizer function call with another.

REFERENCES

- [1] B. ADAMS, K. DALBEY, M. ELDRED, D. GAY, L. SWILER, W. BOHNHOFF, J. EDDY, K. HASKELL, AND P. HOUGH, *DAKOTA, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification and Sensitivity Analysis Version 5.0+ User Manual*, 2010.
- [2] K. DEB, A. PRATAP, S. AGARWAL, AND T. MEYARIVAN, *A fast and elitist multiobjective genetic algorithm: Nsga-ii*, *IEEE Transactions on Evolutionary Computing*, 6 (2002), pp. 182–197.
- [3] G. A. GRAY AND T. G. KOLDA, *Algorithm 856: Appspack 4.0: Asynchronous parallel pattern search for derivative-free optimization*, *ACM Transactions on Mathematical Software* 32(3):485-507, (September 2006).
- [4] W. HART, *UTILIB 3.0: A Utility Library for Portable C/C++ Software*, <https://software.sandia.gov/acro/releases/votd/acro/packages/utllib/doc/uguide/html/>.
- [5] W. HART AND J. SIROLA, *A Common Repository for Optimizers (ACRO)*, <https://software.sandia.gov/trac/acro>.
- [6] W. E. HART, *An Introduction to the COLIN Interface*.
- [7] J.D.GRIFFIN, K.R.FOWLER, G.A.GRAY, T.HEMKER, AND M.D.PARNO, *Derivative-free optimization via evolutionary algorithms guiding local search (eagls) for minlp*. available as Sandia Technical Report SAND2010-3023J, 2010.

Computational Applications

Articles in this section discuss the use of computational techniques in physical simulations. Presented, here are several simulations of varying physics at both the atomic and continuum level.

Theofanis et al. describe the electron force field wavepacket molecular dynamics method. They show that this force field is able to reproduce shock pressure for the material polyethylene. *Ortega and Scovazzi* consider a new remapping approach for finite element arbitrary Lagrangian-Eulerian methods based on flux corrected transport. They present results that show that the new scheme preserves monotonicity and high-order convergence where the solution is smooth. *Maskey et al.* perform a molecular dynamics study of a particular conjugated nanoparticle polymer. Using these numerical results, the authors show that the particle is in a stable collapsed state in a poor solvent but unravels in a good solvent. *Anderson et al.* use density functional techniques to investigate defects in amorphous silicon dioxide. *Takato et al.* use molecular dynamics to study the collision of two nanoparticles. Their study finds distinct regimes where the particles undergo elastic and plastic deformations. *Costolanski and Salinger* outline a method to model the performance of a resonant tunneling diode. Details about the Trilinos implementation are presented and discussed.

E.C. Cyr
S.S. Collis

December 17, 2010

AN ELECTRON FORCE FIELD STUDY OF SHOCKED POLYETHYLENE

PATRICK L. THEOFANIS*, THOMAS R. MATTSSON†, AND AIDAN P. THOMPSON‡

Abstract. Electron force field (eFF) wavepacket molecular dynamics simulations of the principal shock Hugoniot are presented for crystalline polyethylene (PE) models along with a description of the eFF method. The eFF results are in reasonably good agreement with previous DFT theories and experimental data which is available up to 80 GPa. We predict shock Hugoniots for PE up to 450 GPa. In addition we provide an analysis of the phase transformations which occur due to heating. Our analysis includes ionization fraction and particle temperatures during isotropic compression. We find that above compression to 2.4 g/cm^3 the PE structure begins to break down and electrons are ionized. Despite using a simple spherical Gaussian wavepacket basis, eFF is able to reproduce shock pressures for this relatively complex material and this demonstrates the potential for eFF as a method for studying matter in extreme conditions

1. Introduction: Studying Materials Under Extreme Conditions With an Electron Force Field. The next generation of materials sourced for energy production, high-velocity transportation, military and medical devices, and computer hardware will need to reliably operate under extreme conditions. We define extreme conditions as involving temperatures in excess of 1000 Kelvin, static or dynamic pressures greater than tens of mega-Pascals, strains and strain rates greater than 1 km/s, radiative fluxes greater than 100 dpa, high intensity electromagnetic fields, and corrosive or erosive conditions whether these conditions be encountered in combination or separately.

Properties of hydrocarbons under pressure are of significant interest for basic science as well as for practical applications. At Sandia National Laboratories, hydrocarbon foams are used in dynamical materials research, within the Inertial Confinement Fusion (ICF) program, and as a source of high temperature x-rays in radiation research. There is also significant interest in the properties of hydrocarbons under pressure stemming from the need to understand the deep earth. For example, are there abiotic pathways for formation of methane and longer hydrocarbons under pressure, possibly in combination with catalytic transition metals present in the earth's mantle? There are also astrophysical applications: the atmospheres of the ice giants Uranus and Neptune contain significant amounts of methane, water, and ammonia. Understanding properties of hydrocarbons under pressure are crucial in modeling the structure of giant planets as well as exoplanets.

The interest in material responses to extreme conditions has prompted shock studies on a wide variety of elements [1], but experimental and theoretical progress has been slower for more complicated materials like polymers, semiconductors, superconductors, etc. Mattsson and collaborators provided first-principles and molecular dynamics (MD) simulations of shocked polyethylene (PE) to fill this research void [17]. PE is an interesting material because of the wealth of experimental information about its thermodynamic, elastic, and shocked properties [9][26][18]. Typical experimental data provides the equations of state for materials through the Rankine-Hugoniot curves. This data is invaluable, but other important measurements such as the temperature response to shock, ionization cross sections and rates, and chemical compositions behind the shock wave are difficult or impossible to measure. Without such information it is impossible to determine the types and rates of chemical reactions that occur behind the shock wave. Quantum and classical molecular dynamics simulations can provide temperatures corresponding to the Rankine-Hugoniot equations of state but they cannot provide details regarding photochemical and photophysical processes, and

*Materials and Process Simulation Center, California Institute of Technology, ptheofan@caltech.edu

†Sandia National Laboratories, trmatts@sandia.gov

‡Sandia National Laboratories, athomps@sandia.gov

most classical potentials cannot provide information about transient chemical species. The electron force field (eFF) method, a semi-classical molecular dynamics technique is capable of providing positions, forces, particle velocities and energies for both electrons and nuclei in addition to the standard thermodynamics quantities [23]. We can simulate the experimental observables mentioned above through eFF trajectories. The details of the eFF method are provided in the next section of this manuscript.

In this paper, we will describe the eFF method and describe its application to a hydrostatic shock study of PE. From this study we hope to determine the PE Rankine-Hugoniot and compare it to experiments and other theoretical methods. Mattsson and collaborators demonstrated that DFT with the AM05 functional is able to model polyethylene shock physics despite lacking explicit Van der Waals terms. Like most flavors of DFT, eFF has no explicit energy expression for Van der Waals or London dispersion interactions. In fact, all chemical phenomena like bond lengths and strengths, steric effects, charge distributions, conformational preferences, and electron shell filling are emergent properties in eFF. This study will be an important test of eFF's ability to describe non-covalent, non-metallic systems. We have calculated material properties of the hydrostatically shocked PE and we have characterized the chemical composition and physical phenomena resulting from the shock. Unlike most other theoretical methods, eFF provides ionization yields and a reactive dynamics picture of hydrostatic compression.

2. The Electron Force Field. eFF was developed with modeling matter under extreme conditions in mind. Density functional theory and other *ab initio* quantum mechanical techniques provide excellent descriptions of matter at low temperature and pressure and classical plasma theories provide good descriptions of matter at high temperatures. There exists a “computational no-man’s land” between these thermodynamic regimes that remains a challenge to theorists. eFF was developed to bridge this gap and provide good descriptions of matter near its ground state *and* in highly excited states.

2.1. The eFF Energy Expression. eFF overcomes the difficulties of modeling potentially non-adiabatic systems by evaluating the energy of the system as a function of the nuclear coordinates and electron coordinates with a small set of universal electron parameters. This ensures that energy may be partitioned separately into nuclear and electronic degrees of freedom, thus electrons may hop between states without concomitant nuclear motion. We choose to describe nuclei as classical particles and we describe the electrons with a wavefunction of floating spherical Gaussian orbitals similar to the method of Frost [8]. We define our wavefunction as the Hartree product of floating spherical Gaussian wavepackets as such:

$$\Psi \propto \prod_j \exp\left[-\left(\frac{1}{s^2} - \frac{2p_s}{s}i\right)(\mathbf{r} - \mathbf{x})^2\right] \cdot \exp[i\mathbf{p}_x \cdot \mathbf{x}] \quad (2.1)$$

With positions \mathbf{x} , translational momenta \mathbf{p}_x , radial size s and radial momental p_s . Rather than use a fully antisymmetrized wavefunction (for which an energy evaluation would require $O(N^4)$ operations), we choose instead a Hartree product wavefunction which will only requires $O(N^2)$ operations to compute the energy. Using a Hartree product wavefunction violates the antisymmetry principle for fermions which requires that interchanging any two fermions should cause the sign of the wavefunction to change. In order to satisfy the Pauli principle, we must account for the difference in energy between a full antisymmetrized wavefunction, and a product wavefunction like our Hartree product. To do this we include an explicit Pauli energy term which will be described shortly.

The full potential energy expression is:

$$E = E_{ke} + E_{nuc-nuc} + E_{nuc-elec} + E_{elec-elec} + E_{Pauli} \quad (2.2)$$

We define the component energies as follows:

$$E_{ke} = \frac{\hbar^2}{m_e} \sum_i \frac{3}{2} \frac{1}{s_i^2} \quad (2.3)$$

$$E_{nuc-nuc} = \frac{1}{4\pi\epsilon_0} \sum_{i<j} \frac{Z_i Z_j}{R_{ij}} \quad (2.4)$$

$$E_{nuc-elec} = -\frac{1}{4\pi\epsilon_0} \sum_{i,j} \frac{Z_j}{R_{ij}} \text{Erf}\left(\frac{\sqrt{2}R_{ij}}{s_i}\right) \quad (2.5)$$

$$E_{elec-elec} = \frac{1}{4\pi\epsilon_0} \sum_{i<j} \frac{1}{x_{ij}} \text{Erf}\left(\frac{\sqrt{2}x_{ij}}{\sqrt{s_i^2 + s_j^2}}\right) \quad (2.6)$$

$$E_{Pauli} = \sum_{\sigma_i=\sigma_j} E(\uparrow\uparrow)_{ij} + \sum_{\sigma_i \neq \sigma_j} E(\uparrow\downarrow)_{ij} \quad (2.7)$$

where (2.7) comprises the Pauli potential for same spin and opposite spin electrons, respectively. (2.3) describes the “quantum” electronic kinetic energy, which should not be confused with the classical translational kinetic energy of the electron. The error functions in (2.5) and (2.6) arise from the fact that the electron charges are “smeared” over the volume of the Gaussian sphere. Recall that an error function is defined as the integral over a Gaussian and its argument is the upper limit of the integral. This formulation of the Coulomb interactions ensures that the finite sized spherical Gaussians act like point charges at large distances from the other interacting particle. The same spin Pauli energy function is defined as:

$$E(\uparrow\uparrow)_{ij} = \left(\frac{S_{ij}^2}{1 - S_{ij}^2} + (1 - \rho) \frac{S_{ij}^2}{1 + S_{ij}^2} \right) \Delta T_{ij} \quad (2.8)$$

and the opposite spin Pauli energy is

$$E(\uparrow\downarrow)_{ij} = \left(\frac{(1 - \rho)S_{ij}^2}{1 + S_{ij}^2} \right) \Delta T_{ij} \quad (2.9)$$

where ΔT is the kinetic energy change upon antisymmetrization and S is the overlap of the wavepackets. We can further define these two terms:

$$\Delta T_{ij} = \frac{\hbar^2}{m_e} \frac{3}{2} \left(\frac{1}{\bar{s}_i^2} + \frac{1}{\bar{s}_j^2} \right) - \frac{2(3(\bar{s}_i^2 + \bar{s}_j^2) - 2\bar{x}_{ij}^2)}{(\bar{s}_i^2 + \bar{s}_j^2)^2} \quad (2.10)$$

$$S_{ij} = \left(\frac{2}{\bar{s}_i^2/\bar{s}_j^2 + \bar{s}_i^2/\bar{s}_i^2} \right)^{3/2} \exp(-\bar{x}_{ij}^2/(\bar{s}_i^2 + \bar{s}_j^2)) \quad (2.11)$$

The last two equations contain the only empirical parameterizations in eFF. We define $\rho = -0.2$, $\bar{x}_{ij} = x_{ij} \cdot 1.125$ and $\bar{s}_i = s_i \cdot 0.9$. These parameters were fit using a small set of

hydrocarbons and light metal hydrides. ρ can be thought of as an orthogonalization parameter while \bar{x} and \bar{s} are distance and size scaling parameters, respectively. The Pauli energy functions in (2.10) and (2.11) are derived by taking the kinetic energy differences of orthogonalized and non-orthogonalized wavefunctions. The Pauli functions are derived from $E(\uparrow\uparrow) = E_u - (1 - \rho)E_g$ and $E(\uparrow\downarrow) = -\rho E_g$. The full derivation can be found in [23]. eFF uses the difference between Slater and Hartree wavefunctions for the “ungerade” energy expression and the difference between a general valence bond and a Hartree wavefunction to calculate the “gerade” energy expression. The physical interpretation of this effect is more easily understood in terms of orthogonal orbitals. When two same-spin electrons approach one another, their wavefunctions increase in slope to decrease their overlap (they compress in width). This increase in slope increases their gradient and kinetic energy is increased. Wilson and Goddard interpret this change in energy as the Pauli repulsion energy [28]. (2.7) recovers this energy and ensures that eFF electrons satisfy the Pauli exclusion principle.

The beauty of eFF is in its simplicity. With only three empirical parameters it can reproduce a variety of physical quantities like bond lengths, angles, ionization potentials, and bulk properties. The simple nature of the energy and gradient expressions makes eFF computationally far cheaper than conventional quantum mechanics calculations [23][24][25][12].

2.2. The eFF Equations of Motion and Dynamics. In 1975 Heller demonstrated wavepacket molecular dynamics (WPMD) as a method for simulating systems in a semi-classical manner [10]. Rather than making a WKB approximation, wherein one assumes that \hbar is very small [27][15], he approximated that the wavepacket exists in a local harmonic potential. By substituting a wavefunction of the type in (2.1) into the time-dependent Schrödinger equation with a harmonic potential Heller derived the Hamilton equations of motion:

$$\mathbf{p}_{\mathbf{x}} = m\dot{\mathbf{x}} \quad \dot{\mathbf{p}}_{\mathbf{x}} = -\nabla V \quad (2.12)$$

These equations are consistent with Ehrenfest’s theorem which states that the average position of a wavepacket follows a classical trajectory [7]. Following the same procedure for the first exponential in (2.1), and making the assumption that no external potential exists, we can derive the equations of motion for the radial degree of freedom:

$$p_s = \frac{3}{4}m_{elec}s \quad \dot{p}_s = -\frac{\partial E}{\partial r} \quad (2.13)$$

In eFF, m_e is defined in three places: (2.3), (2.10), and (2.13). In the former two equations, which correspond to the electron’s potential energy, m_e is defined as the true electron mass. In the latter equation, it is a user-defineable parameter. Changing m_e in the potential energy terms would affect the sizes and bond lengths of electrons in GS atoms, so it is fixed. Allowing the “dynamical” electron mass in the equations of motion to be adjusted by the user serves a practical purpose: it allows the user to increase the electron’s mass so that it is commensurate with the nuclear masses, and this allows the user to increase the time step of dynamics simulations. It also makes for better coupling in deterministic thermostats. This is not unprecedented; the Landau theory of Fermi liquids uses heavy quasiparticles that obey Fermion statistics and the electron mass is adjustable in semi-classical theories of electron transport in semiconductors. It is prudent to attempt simulations using both the real electron mass, and a larger, user-defined electron mass. The factor of 3/4 in (2.13) arises directly from the substitution of a Gaussian wave packet into the time dependent Schrodinger equation. These equations are exact for harmonic potentials, and it was shown that they performed well for simple anharmonic potentials like the double well potential [23]. We can use the energies and forces from the electron force field in conjunction with this WPMD scheme as a fully functioning molecular dynamics method.

3. Construction of Simulation Cells and Computational Methods.

3.1. Polyethylene Models. A crystalline PE model was created by taking experimental PE crystal lattice parameters and making the chains finite. Cell parameters were taken from [3]. The final cell contained $12 \times C_{12}H_{26}$ molecules. In real samples of crystalline PE the chains are finite in length and the PE is only crystalline in small domains with lamella ranging from 70 to 300 Å in thickness and extending several microns laterally [6]. The volumes of the boxes were adjusted so that the densities of both systems was 0.98 g/cm³.

3.2. Computational Methods. A version of eFF incorporated into the popular and powerful molecular dynamics simulator, LAMMPS, was used for all simulations [12][20].

To prepare the cells for shock simulations a conjugate gradient scheme was used to minimize the potential energy of each cell. The search terminated when either the linesearch α or mean force became zero with a tolerance of 1.0×10^{-6} or if the energy difference between each step was smaller than 1.0×10^{-6} . Preliminary microcanonical ensemble simulations revealed that a timestep of 0.001 fs was necessary to prevent energy drift over simulation periods of 2 ps. Two canonical ensembles were prepared for the PE model: the first was prepared with NVT using a Nose-Hoover thermostat, and the second was prepared using a Langevin thermostat. System temperature was obtained from the kinetic energy of the atomic nuclei. The samples were allowed to equilibrate for an additional 2 ps. From this “ground state” sample, samples of densities between 1.3-3.0 g/cm³ were generated by isothermally and isotropically compressing the ground state simulation cell over a period of 100 fs using the LAMMPS “fix deform” function in conjunction with either the Nose-Hoover or Langevin thermostat. A total of 17 density points for each ensemble were generated for the $12 \times C_{12}H_{26}$ cell in increments of 0.1 g/cm³. These density points were also allowed to equilibrate for 2 ps at 300K.

3.3. The Principal Hugoniot. A Hugoniot curve is the locus of thermodynamic states that can be reached by shock compression of an initial state. These states satisfy the Rankine-Hugoniot energy condition [21][11]:

$$U - U_0 = \frac{1}{2}(P + P_0)(V_0 - V) \quad (3.1)$$

where U is the internal energy, P is the pressure of the system, and V is the cell volume. The kinetic energy of the electrons is defined by (4.1). It is assumed that each point on this curve Hugoniot corresponds to a state of thermodynamic equilibrium wherein the stress state is hydrostatic. For solids, this latter condition is only valid when the yield stress is much lower than the mean stress [5]. When the initial state variables P_0 , V_0 , and U_0 are those of the uncompressed sample at room temperature, the Hugoniot curve is called the principal Hugoniot.

We generated states on the principal Hugoniot using the following iterative procedure. First the volume of the system is specified, representing a particular degree of compression. Then the temperature of the system is quickly increased by changing the set-point of the thermostat. 20 fs of dynamics are run after the thermostat jump, during which averages of the energy, temperature and pressure of the new state are obtained. These values are used to evaluate the residual energy E_{res} , given by

$$E_{res} = (U - U_0) - \frac{1}{2}(P + P_0)(V_0 - V). \quad (3.2)$$

When $E_{res,i}/E_{ke,i} < 0.05$ the Hugoniot condition is considered satisfied. If this inequality is not satisfied another iteration is performed. The new thermostat setpoint is calculated from:

$$T_{i+1} = T_i \left(1 + \frac{1}{2} \frac{E_{res,i}}{E_{ke,i}} \right) \quad (3.3)$$

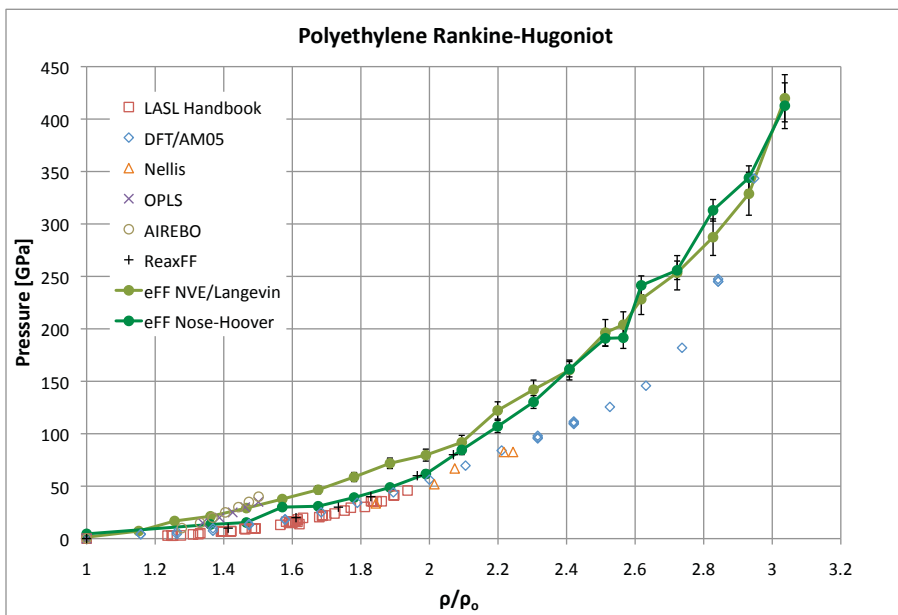


FIG. 4.1. Hugoniot data for the $12 \times C_{12}H_{26}$ system. Two eFF data series are plotted. In dark green closed circles are the steady-state canonical ensembles computed using a Nose-Hoover thermostat. The light green closed circles are the steady-state canonical ensembles computed using a Langevin thermostat. Experimental data from the LASL handbook [1] and Nellis [18] is provided in open red squares and open orange triangles, respectively. Data taken from [17] for the classical MD potentials, Reax, OPLS, and AIREBO is included for comparison.

Once this iterative calculation has converged, the final state is sampled for a further 5 ps. This calculation ensures that the Hugoniot condition is actually met, and the standard deviation in pressure is used to provide error bars for the principal Hugoniot.

4. Results and Discussion. eFF is a reasonably fast and highly scalable MD technique. eFF is slower than Lennard-Jones potentials by a factor of 1500. For reference, ReaxFF is slower than LJ by a factor of 340. Speeds of 5×10^{-6} seconds/timestep/particle were achieved for the 1632 particle $12 \times C_{12}H_{26}$ system during an NVE/Langevin equilibration run on Los Alamos National Laboratory’s “Lobo” high performance computer cluster using 32 processors. More eFF performance information can be found in [12].

4.1. The Polyethylene Principal Hugoniot. Figure 4.1 is the principal Hugoniot projected onto the pressure-density plane. For intermediate compressions the Nose-Hoover thermostat matched the experimental and DFT Hugoniot points quite closely. The Langevin ensemble did not do as well, but still performed reasonable well, within 50% of the experimental value. At greater compressions, both eFF methods predicted the shock pressure high relative to DFT. The results show that eFF is systematically “too stiff” relative to the experimental and DFT/AM05 [2] data. However, eFF outperforms several classical MD potentials such as AIREBO [22], OPLS [13], and exp-6 [4]; the data for these can be found in [17]. These potentials met their pressure asymptote below 1.7 g/cm^3 . These results demonstrate the difficulty in modeling the behavior of complex materials under shock compression.

At high compression interesting features appear in the principal Hugoniot. In the AM05 data series a shoulder feature appears at 2.3 g/cm^3 . In the eFF Nose-Hoover data a series of shoulders appear between 2.4 and 2.8 g/cm^3 . The first of these appears as a clear plateau just past 2.4 g/cm^3 . The next two features are more subtle. Two shoulders appear in the eFF

Langevin data series. The first appears at 2.0 g/cm^3 and the second can be found at 2.45 g/cm^3 . These data features correspond to tangible transitions in the the molecular structure. Mattsson found that the AM05 shoulder corresponded to PE backbone bond breaking [17]. The possible causes for the eFF data features will be discussed shortly.

4.2. Structural Decomposition. The heat caused by hydrostatic shock induces molecular dissociation and ionization. This causes the polymers to transition from liquid to atomic liquid and finally, at extremely high shock pressures, to dense plasma. Along the way the thermophysical character of the ordinarily insulating polymer changes drastically. These changes are manifested in the specific heat capacity, conductivity, and emissivity of the material. The material's response to high shocks may have important implications for its use in certain environments.

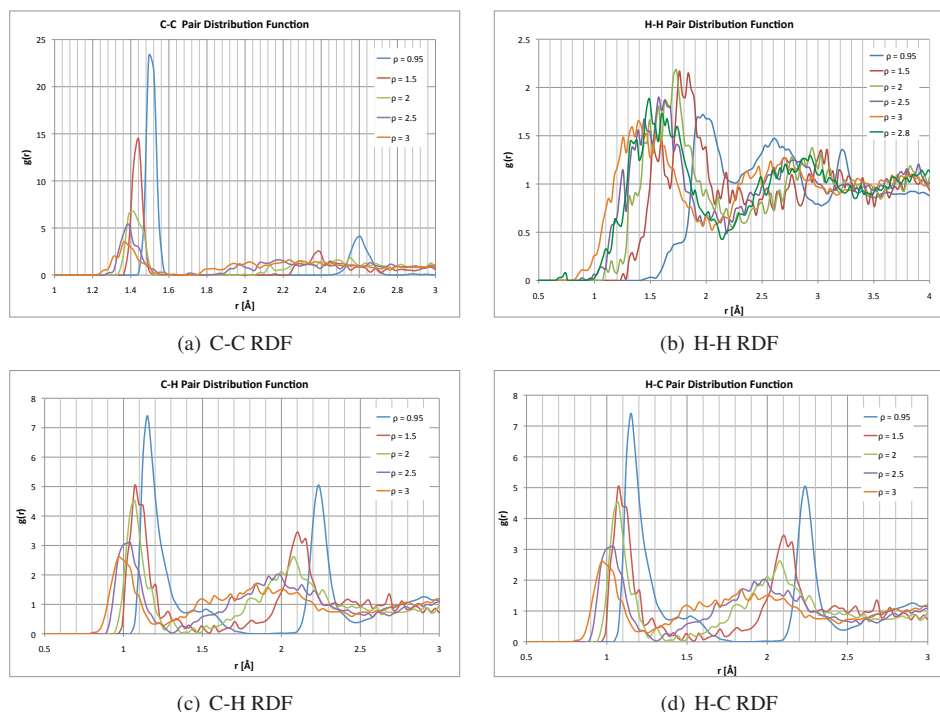


FIG. 4.2. Radial distribution functions for various pairs of nuclei. Each plot contains RDF data for densities of 0.95, 1.5, 2.0, 2.5, and 3.0 g/cm^3 . The H-H RDF (b) also contains an RDF for 2.8 g/cm^3 because compression to this density produces molecular hydrogen.

An analysis of the radial pair distribution functions for different degrees of compression demonstrates that significant structural decomposition occurs upon shock. Figure 4.2(a) shows that carbon bonds are compressed as the sample is compressed. The carbon remain discretely bound to their neighbors because a minima with a value of zero is found just beyond the standard C-C bond length of 1.5 Å for each level of compression. Contrast this to the C-H pair distribution function in Fig. 4.2(c) where the data indicate a phase change to an atomic fluid of hydrogens. The 3.0 g/cm^3 series resembles a Lennard-Jones fluid. At this level of shock compression the hydrogen are partially dissociated from the PE chains. This behavior is also seen in the H-C pair radial distribution function in Fig. 4.2(d). From this data we can conclude that between compression from 2.5 to 3.0 g/cm^3 the structure is shocked strongly enough to cause a phase transition to a state where the carbon backbones remain

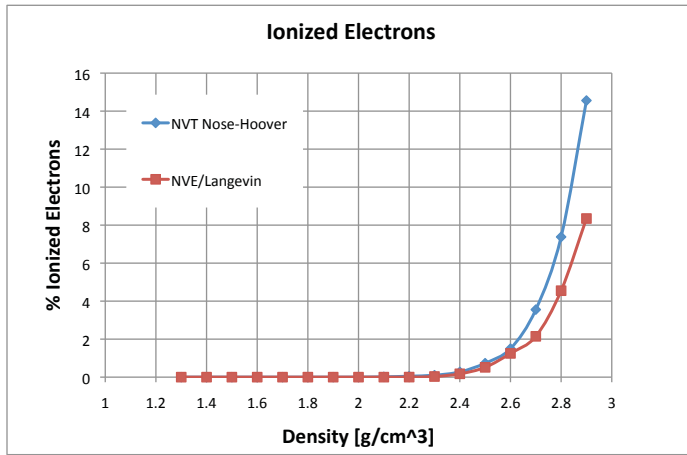


FIG. 4.3. The percentage of ionized electrons at different stages of compression for the two eFF ensembles.

mostly intact but are solvated by loosely associated hydrogen atoms. For compression to 2.8 g/cm³, the small peak in the H-H data in Fig. 4.2(b) near 0.7 Å shows that molecular hydrogen is formed. At this density the system has a shock temperature of 2992 K. Mattsson and collaborators also found hydrogen formation when their shocked PE reached 2800-3100 K [16]. In their simulations this temperature range corresponded to densities of 2.2-2.3 g/cm³. For temperatures higher than 3100 K the molecular hydrogen becomes too energetic to stay bound, and at lower temperatures the hydrogen does not have enough energy to dissociate from their polyethylene backbone. The eFF results are consistent with MD and DFT results for equivalent temperatures, but the structural changes occurring as a result of heating occur at higher compressions in eFF.

One of eFF's greatest assets is its ability to separate electron degrees of freedom, energies, positions, momentum, and forces from those of the nuclei. This gives us unrivaled ability in the world of molecular dynamics to measure electronic physical quantities. In our investigation of PE we have used this to measure the ion fraction at each stage of shock. To do this we measure the total electron – potential and kinetic – energy of each electron at each timepoint in our simulations. In eFF an electron's kinetic energy is given by:

$$E_{ke} = \frac{1}{2}m_e\dot{\mathbf{x}}^2 + \frac{1}{2}\frac{3}{4}m_e\dot{s}^2 \quad (4.1)$$

The electron's potential energy is measured as half the sum of pairwise interactions plus the electronic kinetic energy in (2.3). To measure the correct potential energy of an electron we must multiply the potential energy of the electron in question by two and subtract the quantum electronic kinetic energy (which was doubled by multiplying the potential energy by two). To compute the total energy of the electron we add the electron classical kinetic energy according to (4.1). We define an electron as being ionized if its total energy is greater than zero. The data in Fig. 4.3 was computed in this manner. The number of ionized electrons was averaged over each steady state calculation.

The results of the ionization calculations show that at compressions above a density of 2.4 g/cm³ the electrons in PE begin ionizing. This implies that PE is conductive above this density. Above this threshold electron ionization draws energy from the system and this affects the pressure and temperature of the Hugoniot. The temperature suppression caused by ionization can be seen in Fig. 4.1. The onset of ionization is likely a cause of the plateau and

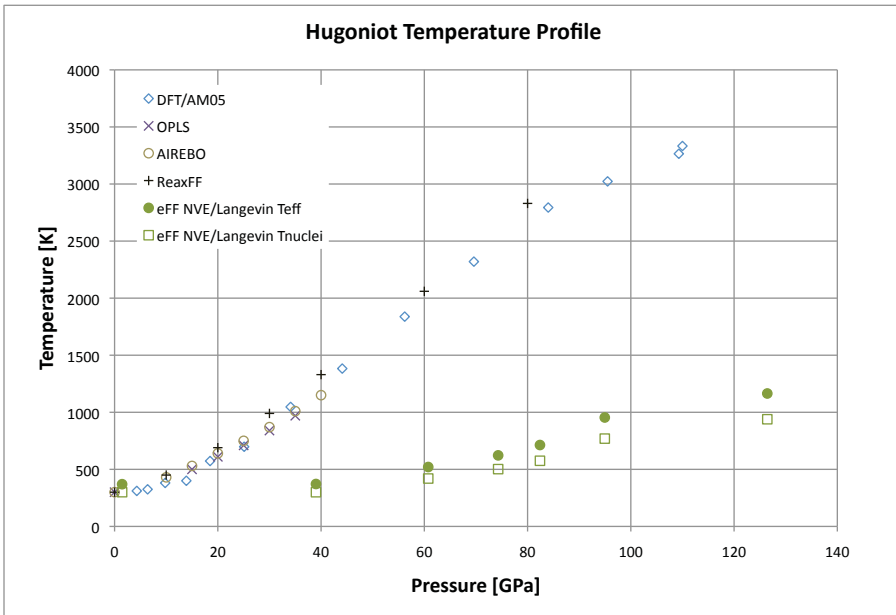


FIG. 4.4. Pressure temperature Hugoniot relationships. Two temperatures are plotted for the eFF Langevin ensemble: The closed light green circles correspond to the eFF temperature computed according to (4.2), and the open light green squares correspond to the temperatures of the nuclei only. Data for DFT/AM05, OPLS, AIREBO, and ReaxFF is also provided from [17]

shoulder features in the principal Hugoniot.

4.3. Shock Temperatures. Before a discussion of the results is presented, the definition of temperature in eFF ought to be discussed. (4.2) is the default temperature definition in eFF.

$$T = \frac{2}{3k_B} \frac{1}{N_{nuc}} \langle K \rangle \quad (4.2)$$

In this equation $\langle K \rangle$ is the average kinetic energy of *all* the particles in the system, and N_{nuc} is the number of nuclear degrees of freedom. This definition sets the kinetic contribution to the specific heat capacity to $\frac{3}{2}k_B T$ where only the nuclear degrees of freedom contribute. This approximation should be valid below the Fermi temperature where electrons are not excited, and thus have negligible kinetic energy. However, in practical eFF simulations electrons gain kinetic energy, and thus a temperature, as soon as the temperature of the system rises above 0 K. As a result, the temperature computed by (4.2) will be too high.

Temperature is a vital measure for any system because it dictates the rates and types of chemical reactions that are occurring within that system. Mattsson and collaborators evaluated the pressure-temperature slice of the PE Hugoniot for several potentials. The classical MD potentials were in near perfect agreement with DFT. The temperatures presented in Fig. 4.4 for the NVE/Langevin data series are calculated from the mean kinetic energies of the nuclei and from the eFF temperature definition. The data shows that relative to DFT, eFF drastically underestimates the temperature along the Hugoniot. To understand the cause of this we measured temperature in a more fundamental way. We conducted an analysis of the temperatures of each degree of freedom calculated according to equipartition, for example:

$$\frac{1}{2}k_B T_i = \left\langle \frac{1}{2} m v_i^2 \right\rangle \quad (4.3)$$

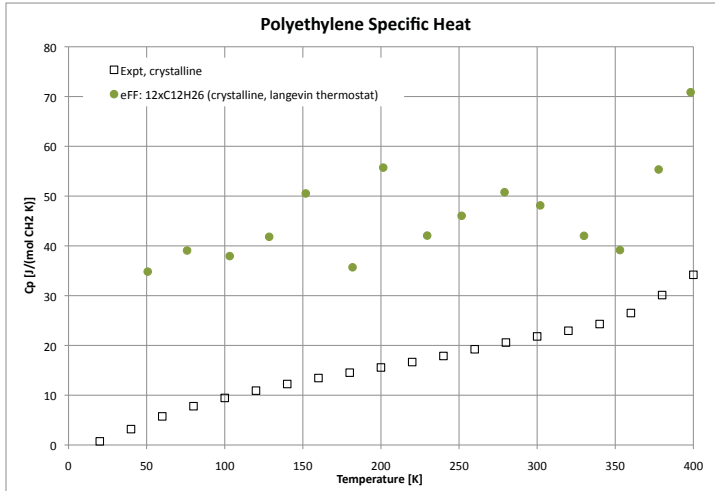


FIG. 4.5. The specific heat capacity of crystalline polyethylene. The green circles show data computed from constant volume heat capacity simulations computed using NVE with a Langevin thermostat. The data was converted to constant pressure heat capacity using the volume and compressibility parameters from [14]. The open black squares show experimental heat capacity data for polyethylene crystal taken from [9]

for nuclear and electron translational degrees of freedom, and

$$\frac{1}{2}k_B T_s = \left\langle \frac{1}{2} \frac{3}{4} m \dot{s}^2 \right\rangle \quad (4.4)$$

for the electron radial degree of freedom. This test revealed that, when using the Langevin thermostat, the temperatures satisfied equipartition and matched the user-specified temperature of the thermostat. Though not shown, we found that Nose-Hoover did a poor job of partitioning energy, even after 5 ps of simulation. It typically provided nuclei with more energy than electrons, and this is likely due to poor coupling between the electrons and heat bath. This can be avoided by using Nose-Hoover chains, just as you would if your system included heavy metals and hydrogen.

Figure 4.5, a plot of the specific heat capacity between 20 K and 400 K, shows that in general eFF overestimates the specific heat capacity of PE. This plot demonstrates that the PE model is resistant to temperature change when energy is added to the system. For systems like polyethylene which contain far more electrons than ions, eFF will exhibit this behavior. As discussed before, this is because electrons are essentially treated as classical particles, and they obtain kinetic energy as soon as the system has a temperature.

There is one other possibility for the temperature discrepancy in Fig. 4.4. In classical molecular dynamics simulations energy can be distributed into translational, rotational, vibrational or bonding degrees of freedom for nuclei only. In eFF the electron degrees of freedom offer additional channels for energy partitioning. Exciting electrons (i.e. ionizing them) uses energy that might otherwise go to the kinetic energy of the nuclei. This possibility might account for the fact that the temperature of the system is too low. Figure 4.4 shows that the energy discrepancy exists even at 300 K, where the system ought to be in its ground state. As was mentioned earlier, this is because the electrons have finite kinetic energy at 300 K. There is experimental evidence for the fact that electron excitations suppress temperature increases in shocked xenon [19], so the argument presented here is not unreasonable. Experimentally measured temperatures do not exist for shocked PE so it has yet to be seen whether eFF incorrectly predicts temperature or not.

5. Conclusions. In this manuscript we have simulated the material response of PE to hydrostatic shock compression using the electron force field wavepacket molecular dynamics method. We conclude that eFF performs well for high energy shock simulations with simulated Hugoniot reproducing experimental and theoretical findings. eFF predicts that above 2.4 g/cm^3 the polymer backbone will begin breaking apart and electrons begin ionizing. For 300 GPa shocks significant structural deterioration and ionization will occur. Temperature remains an issue for eFF, and determining the best measure of temperature for eFF will be the focus of continuing work. The fidelity of the eFF Hugoniot indicates that Van der Waals interactions are not important under shock conditions; eFF should be able to model other complex materials with $Z < 6$. We hope that the results presented in this paper will stimulate further work on the applicability of eFF for problems in high energy-density physics.

REFERENCES

- [1] *LASL Shock Handbook*, edited by S.P. March, University of California Press, Berkeley, CA, 1980.
- [2] R. ARMIENTO AND A. MATTSSON, *Functional designed to include surface effects in self-consistent density functional theory*, Phys. Rev. B, 72 (2005), p. 085108.
- [3] G. AVITABILE, R. NAPOLITANO, B. FIROZZI, K. ROUSE, H. THOMAS, AND B. WILLIS, J. Polym. Sci. Lett. Ed., 13 (1975), p. 351.
- [4] O. BORODIN, G. SMITH, AND D. BEDROV, *Development of many-body polarizeable force fields for li-battery components: 1. ether, alkane, and carbonate-based solvents*, J. Phys. Chem. B., 110 (2006), p. 6279.
- [5] M. BOSLOUGH AND J. ASAY, *High-Pressure Shock Compression of Solids*, ed. by JR Asay and M Shahinpoor., Springer-Verlag, New York, NY, 1993.
- [6] M. DOYLE, Polym. Eng. Sci., 40(2) (2000), pp. 330–335.
- [7] B. DUI, C. COHEN-TANNOUDJI, AND F. LALOE, *Quantum Mechanics Volumes 1 and 2*, Wiley-Interscience, New York, NY, 1978.
- [8] A. FROST, *Floating spherical gaussian orbital model of molecular structure. i. computational procedure. lih as an example.*, J. Chem. Phys., 47 (1967), pp. 3707–3713.
- [9] U. GAUR AND B. WUNDERLICH, *Heat capacity and other thermodynamic properties of linear macromolecules ii. polyethylene*, J. Phys. Chem. Ref. Data, 10(1) (1981), pp. 119–152.
- [10] E. HELLER, *Time-dependent approach to semiclassical dynamics*, J. Chem. Phys., 62(4) (1975), pp. 1544–1555.
- [11] H. HUGONIOT, J. de l'Ecole Polytechnique, 57 (1887), p. 3.
- [12] A. JARAMILLO-BOTERO, J. SU, A. QI, AND W. GODDARD, J. Comput. Chem., Sept 1 (2010). e-Publication available ahead of publication.
- [13] W. JORGENSEN, D. MAXWELL, AND J. TIRADO-RIVES, *Development and testing of the opls all-atom force field on conformational energetics and properties of organic liquids*, J. Am. Chem. Soc., 118 (1996), p. 11225.
- [14] N. KARASAWA, S. DASGUPTA, AND W. GODDARD, *Mechanical properties and force field parameters for polyethylene crystal*, J. Phys. Chem., 95 (1991), pp. 2260–2272.
- [15] H. KRAMERS, *Wellenmechanik und halbzahlige quantisierung.*, Zeitschrift für Physik, 39 (1926), pp. 828–840.
- [16] T. MATTSSON AND K. COCHRANE, *Pair correlations for the hydrostatic shock of polyethylene*. Private communication, 2010.
- [17] T. MATTSSON, M. LANE, K. COCHRANE, M. DESJARLAIS, F. THOMPSON, AND G. GREST., *First-principles and classical molecular dynamics simulation of shocked polymers*, Phys. Rev. B., 81 (2010), p. 054103.
- [18] W. NELLIS, F. REE, R. TRAINOR, A. MITCHELL, AND M. BOSLOUGH, *Equation of state and optical luminosity of benzene, polybutene, and polyethylene shocked to 210 gpa*, J. Chem. Phys., 80 (1984), p. 2789.
- [19] W. NELLIS, M. VAN THIEL, AND A. MITCHELL, *Shock compression of liquid xenon to 130 gpa (1.3 mbar)*, Phys. Rev. Lett., 48(12) (1982), pp. 816–818.
- [20] S. PLIMPTON, *Fast parallel algorithms for short-range molecular dynamics*, J. Comp. Phys., 117 (1995), pp. 1–19.
- [21] W. RANKINE, Phil. Trans. Roy. Soc., 160 (1870), p. 277.
- [22] S. STUART, A. TUTEIN, AND J. HARRISON, *A reactive potential for hydrocarbons with intermolecular interactions*, J. Chem. Phys., 112 (2000), p. 6472.
- [23] J. SU, *An Electron Force Field for Simulating Large Scale Excited Electron Dynamics*, PhD thesis, California Institute of Technology., Pasadena, CA, 2007.
- [24] J. SU AND W. GODDARD, *Excited electron dynamics modeling of warm dense matter*, Phys. Rev. Lett., 99 (2007), p. 185003.
- [25] J. SU AND W. GODDARD, *Mechanisms of auger-induced chemistry derived from wave packet dynamics*, Proc. Nat. Acad. Sci., 106(4) (2009), pp. 1001–1005.

- [26] K. TASHIRO, *Prog. Polymer Sci.*, 18(3) (1993), pp. 377–435.
- [27] G. WENTZEL, *Eine verallgemeinerung der quantenbedingungen fur die zwecke der wellenmechanik.*, *Zeitschrift fur Physik*, 38 (1926), pp. 518–529.
- [28] C. WILSON AND W. GODDARD, *Exchange kinetic energy, contragradience, and chemical binding*, *Chem. Phys. Lett.*, 5(1) (1970), pp. 45–49.

FLUX-CORRECTED TRANSPORT ALGORITHM FOR THE REMAPPING STEP OF A FEM-ALE METHOD

A. LOPEZ ORTEGA* AND G. SCOVAZZI†

Abstract. Arbitrary Lagrangian-Eulerian (ALE) methods are widely used in continuum mechanics. The idea behind them is to use the complex boundary definitions allowed by a mesh that follows particles, while having the ability to track discontinuities, like in a fixed Eulerian grid. The most common implementation involves various Lagrangian steps, where the mesh moves with the material particles and can be twisted and deformed, followed by a remapping step when necessary, where the mesh is modified (without changes in connectivity) to obtain a more regular one for the next Lagrangian steps. The remapping procedure must ideally assure conservation of mass, momentum and energy for the whole system. We present an approach for the remapping algorithm based on Flux-Correction Transport (FCT) and applied to a Finite Element (FE) scheme that, in addition to global conservation, preserves positivity. Our test results show monotonicity preservation (although it cannot be completely proven) and high-order behavior where the solution is sufficiently smooth.

1. Introduction. Flux-Correction Transport (FCT) algorithms are commonly used in continuum mechanics [2]. In FCT, two numerical schemes are employed. The first one is high-order but can produce overshoots/undershoots, oscillatory or unphysical solutions, most commonly near discontinuities. The second one is a low-order scheme and more diffusive in character. This second method is chosen such that it does not create new extrema and gives meaningful results (i.e. the density cannot become negative). FCT attempts to correct the low-order method by adding parts of the high-order one, where it is possible, while preserving monotonicity. This method was first developed by D. Book and J. Boris (see chapter 1 of [2]) and was mainly used in Finite Volume and Finite Difference schemes in compressible and incompressible fluid dynamics.

In this article, we present a FCT-based algorithm for the remapping step of a FE-ALE method. The application of FCT to Finite Elements is not trivial and was first explored by L. Löhner, D. Kuzmin et al. (see e.g. [6], [4], [8], [7] and Chapters 6, 7 and 8 of [2]). The first one developed a consistent method for obtaining the low-order scheme from a Taylor-Galerkin formulation of the problem and an algorithm to compute the flux correction based on the high-order solution. The second introduced a strategy based on obtaining a low order scheme from the pure Galerkin formulation and estimating the flux correction from the low order solution, normally by iteration. This second approach is closer to the one followed in this article. They applied this technique to the Euler equations in a fixed FE grid with good results. R. Liska et al. in [5] followed the FCT approach for the remapping step but applied it to a Finite Difference ALE method.

This article is structured as follows. First, we describe the Galerkin formulation and present the system of equations to be solved for remapping the density, pointing out the properties that we want to enforce in the subsequent steps. The second step is obtaining the low-order scheme by adding diffusion to the Galerkin formulation. After that, we describe the algorithm needed to compute the correction fluxes. This idea is extended to a system of equations, in which we remap the density, momentum and total energy of a system while maintaining positive density and internal energy. Finally, we present some relevant test results that highlight the accuracy and benefits of this approach.

2. Finite Elements for remapping step. A continuum can be described in both Lagrangian and Eulerian coordinates. The Lagrangian coordinates \mathbf{X} move with the material

*California Institute of Technology, alopezor@caltech.edu

†Sandia National Laboratories, gscovaz@sandia.gov

particles. Eulerian coordinates \mathbf{x} are fixed in space and can only track trajectories by integration of the velocity field. The one-to-one Lagrangian-to-Eulerian map $\mathbf{x} = \mathbf{x}(\mathbf{X}, t)$ with $\mathbf{x}(\mathbf{X}, t_0) = \mathbf{X}$ answers the question of where the particle that was originally at \mathbf{X} is at time t in spatial coordinates. The gradient of this map is the well-known deformation tensor, defined as $F_{ij} = \frac{\partial x_i}{\partial X_j}$. The determinant of this tensor is the Jacobian J of the transformation.

We use the Finite Element Method (FEM) for the Lagrangian step, which advances in time, and the remapping step, which is held at constant time. The aim of the remapping step is to accurately calculate the value of the conserved variables at the nodes of the new mesh, knowing the values of these variables on the old mesh. In this process, we must conserve the density, momentum and energy of the considered system.

We start with the simplest case, a remapping for a scalar variable (in this case, the density ρ). The weak integral form of the mass conservation is the following in Lagrangian coordinates (see [10]):

$$\int_{\Omega_X} \phi(X) \frac{\partial}{\partial t} (J(X)\rho(X)) \Big|_X dX - \int_{\Omega_X} \nabla_X \phi(X) \cdot (\rho(X)\mathbf{w}) dX = 0, \quad (2.1)$$

where $\phi(X)$ is a test function that satisfies the boundary conditions of the problem, ρ is the density and $\mathbf{w} = \hat{\mathbf{c}}\mathbf{J}\mathbf{F}^{-T}$ with $\hat{\mathbf{c}}$ being the relative velocity between the Eulerian and Lagrangian grids.

The remeshing is more easily described in terms of Eulerian variables, since it involves a change in the spatial position of the nodes. The last expression can be transformed to an Eulerian frame of reference using the map $x = x(X, t)$ and in particular $dx = JdX$. For the first integral we get:

$$\int_{\Omega_X} \phi(X) \frac{\partial}{\partial t} (J(X)\rho(X)) \Big|_X dX = \frac{d}{dt} \int_{\Omega_X} \phi(X)\rho(X)J(X) dX = \frac{d}{dt} \int_{\Omega_x} \phi(x)\rho(x) dx, \quad (2.2)$$

and for the second:

$$\int_{\Omega_X} \nabla_X \phi(X) \cdot (\rho(X)\mathbf{w}) dX = \int_{\Omega_x} \nabla_x \phi(x) \cdot (\rho(x)\hat{\mathbf{c}}) dx, \quad (2.3)$$

where we have used $\frac{\partial}{\partial x_i} = \frac{\partial}{\partial X_j} \frac{\partial X_j}{\partial x_i} = (F^{-T})_{ij} \frac{\partial}{\partial X_j}$.

The discretization of ρ is given by a generalized group FE Galerkin formulation [1], where we express $\rho(x) = \rho_i \phi_i(x)$ and $\rho(x)\hat{\mathbf{c}}(x) = \rho_i \hat{\mathbf{c}}_i \phi_i(x)$, where i represents the nodal indices of our mesh and $\phi_i(x)$ are now shape functions of compact support (i.e. $\phi(x) = \sum_i \phi_i(x)$ with $\phi_i(x_i) = 1$ and $\phi_i(x_j) = 0$ for all $j \neq i$).

The time discretization allows us to obtain a fully discrete scheme. We choose to use a finite difference scheme for the temporal derivative of equation (2.2) and evaluate equation (2.3) at the half time step ($\rho_i^{n+1/2} = (\rho_i^{n+1} + \rho_i^n)/2$). This procedure yields:

$$v_{ij}^{n+1} \rho_j^{n+1} - v_{ij}^n \rho_i^n = k_{ij}^{n+1/2} \rho_j^{n+1/2}, \quad (2.4)$$

with

$$v_{ij}^l = \int_{\Omega_{x(l)}} \phi_i^l \phi_j^l dx, \quad (2.5)$$

and

$$k_{ij}^{n+1/2} = -\mathbf{u}_j \cdot \int_{\Omega_{x(n+1/2)}} \phi_j^{n+1/2} \nabla_x \phi_i^{n+1/2} dx, \quad (2.6)$$

where $\mathbf{u}_j = -\hat{\mathbf{c}}_j \Delta t$ is the known displacement of the nodes from the old to the new mesh. The step n defines the current mesh while the $n+1$ defines the new one. This way, we eliminate the time dependency of the problem in favor of the remapping displacements. The negative sign is included because the movement of the boundaries of the cell is opposed to the movement of the flow. Throughout this article, we assume that \mathbf{u}_i is always known and given by a particular remesh strategy.

2.1. Relevant properties of the matrices \mathbf{K} and \mathbf{V} . Both the volume \mathbf{V} and flux \mathbf{K} matrices have important properties that are given by the compact support shape functions used in our Finite Element scheme.

We define a lumped volume matrix \mathbf{V}^L , which is diagonal and whose terms are the sum of the terms of the consistent matrix \mathbf{V} over a row (i.e. $v_i^L = \sum_j v_{ij}$). This lumped matrix conserves the mass of the system. It can be computed directly by:

$$v_i^{L,k} = \sum_j \int_{\Omega_{x(k)}} \phi_i^k \phi_j^k dx = \int_{\Omega_{x(k)}} \phi_i^k dx. \quad (2.7)$$

The stiffness matrix \mathbf{K} can be written as $k_{ij} = -\mathbf{u}_j \cdot \mathbf{c}_{ij}$, where $\mathbf{c}_{ij} = \int_{\Omega_x} \phi_j \nabla \phi_i dx$. The matrix \mathbf{C} is skew-symmetric and $\sum_j \mathbf{c}_{ij} = 0$. The sum over rows of the stiffness matrix ($\sum_j k_{ij} = \sum_j -\mathbf{u}_j \cdot \mathbf{c}_{ij}$) is the discrete Galerkin counterpart of $\int_{\Omega_x} \phi(x) \nabla_x \cdot \mathbf{u}(x) dx$ (the divergence) when only displacements that are tangential to the boundary are allowed, and accounts for changes in volume. The diagonal entries of \mathbf{K} are zero.

These facts are used in the following sections to prove the properties of the low-order scheme and are necessary in applying the flux correction algorithm.

3. Assembly of the low-order system. Our final aim is to solve equation (2.4). This is a linear system and, so, it can be solved directly by:

$$\rho^{n+1} = \left(\mathbf{V}^{n+1} - \frac{1}{2} \mathbf{K}^{n+1/2} \right)^{-1} \left(\mathbf{V}^n + \frac{1}{2} \mathbf{K}^{n+1/2} \right) \rho^n. \quad (3.1)$$

This process however is not computationally efficient for large systems and can give rise to unphysical oscillations. These oscillations arise since this system is typically second order in space and time and, so, cannot handle jumps or shocks without creating overshoots and undershoots. This behavior can lead to unphysical results, like negative densities.

The solution to this issue is based on the FCT algorithm. For this approach, first we need to assemble a low-order scheme that has the correct properties. In Finite Volume and Finite Difference fields a wide variety of different methods are available, depending on the stencils used. That is not the case however for Finite Elements, where the stencil is defined by the shape functions, which are taken typically as simple as possible. Requirements for the low-order scheme are to be conservative, enforce positivity (the scheme cannot give unphysical values), and smooth the extrema of the function while not creating new ones. In addition, it is a FCT algorithm requirement that the low-order scheme follow the same time discretization used for the high-order scheme.

A proposed solution for assembly of the low-order scheme is to substitute the volume matrices \mathbf{V} by their lumped counterparts \mathbf{V}^L and substitute the flux matrix \mathbf{K} by another matrix \mathbf{L} (see chapter 6 of [2]):

$$v_i^{L,n+1} \widehat{\rho}_i^{n+1} - v_i^{L,n} \rho_i^n = l_{ij}^{n+1/2} \widehat{\rho}_j^{n+1/2}. \quad (3.2)$$

The substitution of the consistent volume matrix by the lumped one adds some diffusion to the system. The low-order flux matrix \mathbf{L} is obtained from \mathbf{K} by adding a diffusion matrix \mathbf{D} (i.e., $\mathbf{L} = \mathbf{K} + \mathbf{D}$). The diffusion matrix is symmetric and $\sum_j d_{ij} = 0$ to enforce conservation.

Next, we explore the requirements that the diffusion matrix must satisfy to preserve the positivity and the Local Extremum Diminishing (LED) constraints.

3.1. The positivity constraint. We can express the low-order scheme as:

$$\mathbf{A}\widehat{\rho}^{n+1} = \mathbf{B}\rho^n, \quad (3.3)$$

with $\mathbf{A} = \mathbf{V}^{L,n+1} - \frac{1}{2}\mathbf{L}^{n+1/2}$ and $\mathbf{B} = \mathbf{V}^{L,n} + \frac{1}{2}\mathbf{L}^{n+1/2}$. We suppose that ρ^n is a vector of positive components and we require the same property for $\widehat{\rho}^{n+1}$. Sufficient conditions for this are $b_{ij} \geq 0$, $a_{ij} \leq 0$ for $j \neq i$, $a_{ii} \geq 0$ and $a_{ii} \geq \sum_{j \neq i} a_{ij}$ (see chapter 6 of [2]).

The only non-diagonal contributions to the \mathbf{B} matrix comes from \mathbf{L} . Then, all the non-diagonal components of this matrix must be zero or positive. This can be obtained if \mathbf{D} has positive off-diagonal components. Negative diagonal components are then needed to preserve the null sum over rows. The conditions $a_{ii} \geq 0$ and $a_{ii} \geq \sum_{j \neq i} a_{ij}$ are automatically satisfied because the lumped volume matrix is a diagonal matrix with positive components and the diagonal components of \mathbf{L} are negative (the diagonal entries of \mathbf{K} are zero and \mathbf{D} has non-positive diagonal terms).

The last constraint is that the diagonal terms of \mathbf{B} must be positive. This is not automatically satisfied and depends on the displacement field chosen for the problem. This is a CFL-like condition and appears in every method that includes terms at the previous step n in the right-hand side of Equation (3.2).

In conclusion, we need to generate a low-order stiffness matrix whose non-diagonal components are positive. A simple algorithm for achieving this is the following: we initialize $l_{ij} = k_{ij}$ and, in a loop for all non-diagonal components above (or below) the principal diagonal, we take $d_{ij} = \max(-k_{ij}, -k_{ji}, 0)$ and we update:

$$\begin{aligned} l_{ii} &\leftarrow l_{ii} - d_{ij}, & l_{ij} &\leftarrow l_{ij} + d_{ij}, \\ l_{ji} &\leftarrow l_{ji} + d_{ij}, & l_{jj} &\leftarrow l_{jj} - d_{ij}. \end{aligned} \quad (3.4)$$

Note that the diagonal components are updated each time one non-diagonal component of the same row or column is modified. This algorithm is similar to the one used by Kuzmin et al. in chapter 6 of [2].

3.2. The Local Extreme Diminishing (LED) constraint. The low-order scheme cannot create new local extrema. When the volume matrices at the step $n + 1$ and n are the same (i.e., pure advection), this is achieved just with the conditions we introduced for the low-order stiffness matrix \mathbf{L} . To prove this, we start with a semi-discrete formulation:

$$d\rho_i = \sum_{j \neq i} \frac{l_{ij}}{v_i^{L,n}} (\rho_j - \rho_i), \quad (3.5)$$

where we have used the fact that for pure advection, the sum over a row of \mathbf{K} is zero. It is easy to see that when ρ_i is a minimum $d\rho_i \geq 0$, and when it is a maximum $d\rho_i \leq 0$, because all the non diagonal components of the low-order stiffness matrix are positive. We use compact support shape functions and, in consequence the non-diagonal terms only relate neighboring nodes, so the smoothing happens for all local extrema.

The remapping scheme however deviates from this structure. In this case, we have two different volume matrices at pseudo-time $n + 1$ and n , and a non-zero sum over rows of the stiffness matrix. At a fully discrete level, we can write for our low-order scheme:

$$\rho_i^{n+1} - \rho_i^n = \sum_{j \neq i} \frac{l_{ij}}{v_i^{L,n+1}} (\rho_j^{n+1/2} - \rho_i^{n+1/2}) + \left(\frac{v_i^{L,n}}{v_i^{L,n+1}} - 1 \right) \rho_i^n + \sum_j \frac{k_{ij}}{v_i^{L,n+1}} \rho_i^{n+1/2}. \quad (3.6)$$

Apparently, the two extra terms at the end of the expression could ruin the condition for the LED constraint described in Equation (3.5). An analysis of them reveals that they both account for changes in volume. The first one is just the difference between the initial and final volumes divided by the final volume. The sum over the rows of $k_{ij}/v_i^{L,n+1}$ is a discrete expression of the divergence of the remesh displacement field, which also defines the change in volume.

We now introduce a new desirable property. We would like to obtain a constant remapped density when the input is a constant density, even using the high-order scheme. If we substitute this requirement in Equation (2.4) and sum over rows, the following expression is obtained:

$$0 = \rho_0 \left(\int_{\Omega_{x(t_{n+1})}} \phi_i(x, t_{n+1}) dx - \int_{\Omega_{x(t_n)}} \phi_i(x, t_n) dx - \sum_j \int_{\Omega_{x(t_{n+1/2})}} \phi_j \nabla_x \phi_i(x, t_{n+1/2}) \cdot \mathbf{u}_j dx \right). \quad (3.7)$$

We can achieve this constraint by computing the lumped volume not by quadrature of the shape functions but from the previous expression, i.e. $v_i^{L,n+1} = v_i^{L,n} + \sum_j k_{ij}$. It can be proven that, if the shape functions and the displacement field are linear in space and time respectively, we need an integration formula in time with polynomial exactness equal to the number of dimensions $n_d - 1$ to satisfy equation (3.8) (see Chapter 8 of [10]). Therefore, for a 1D scheme with piecewise linear elements ($c_{i,i+1} = -1/2$ and $c_{i,i-1} = 1/2$), we can demonstrate that this condition is accomplished just by constructing the matrices from shape function quadrature no matter what point in time we take to construct \mathbf{K} (the fact is that the flux matrix will be the same no matter what pseudo-time point between n and $n + 1$ we take for evaluation):

$$v_i^{L,n+1} - v_i^{L,n} = \frac{1}{2} (x_{i+1}^{n+1} - x_{i+1}^n - x_{i-1}^{n+1} + x_{i-1}^n) = \frac{1}{2} (u_{i+1} - u_{i-1}) = k_{i,i+1} + k_{i,i-1}. \quad (3.8)$$

For a 2D scheme, we guarantee that the matrices constructed by quadrature of the shape functions satisfy the geometric conservation law if we evaluate \mathbf{K} at time $n + 1/2$. In the case of the 3D scheme, we need two quadrature points in time. The code that was developed takes into consideration this conservation law and uses the quadrature points that are required to compute the remesh volume matrix. Then, the $\mathbf{K}^{n+1/2} \rho^{n+1/2}$ right-hand side term can be rearranged as a function of these two quadrature points.

Due to the enforcement of a constant solution by the geometric conservation law, the LED constraint is automatically satisfied. We show this by writing:

$$\rho_i^{n+1} - \rho_i^n = \sum_{j \neq i} \frac{l_{ij}}{v_i^{L,n+1}} (\rho_j^{n+1/2} - \rho_i^{n+1/2}) + \left(\frac{v_i^{L,n}}{v_i^{L,n+1}} - 1 \right) \frac{1}{2} (\rho_i^n - \rho_i^{n+1}), \quad (3.9)$$

and taking the last term of the right-hand side to the left-hand side:

$$\rho_i^{n+1} - \rho_i^n = \sum_{j \neq i} \frac{2l_{ij}}{v_i^{L,n+1} (1 + \kappa_i)} (\rho_j^{n+1/2} - \rho_i^{n+1/2}). \quad (3.10)$$

Note that this expression is equivalent to (3.5) but with a modified volume term $(v_i^{L,n+1} + v_i^{L,n})/2$.

We must point out that there can be a small monotonicity preservation violation when computing the low-order solution using a predictor-multi-corrector algorithm. We propose a

possible solution strategy:

$$\begin{aligned} & v_i^{L,n+1} \left(\rho_i^{n+1,(m+1)} - v_i^{L,n} \rho_i^n \right) \\ &= \sum_{j \neq i} l_{ij}^{n+1/2} \left(\rho_j^{n+1/2,(m)} - \rho_i^{n+1/2,(m)} \right) - \left(\sum_j k_{ij} \right) \frac{1}{2} \left(\rho_i^n - \rho_i^{n+1,(m)} \right), \end{aligned} \quad (3.11)$$

where m defines the number of iterates of the predictor-multi-corrector strategy. One can argue that the last term of the right hand side can be moved to the left hand side with $m+1$ instead of m . This will violate the conservation properties however since we would be using two different sets of temporal values to evaluate the matrix \mathbf{L} . Consequently, the scheme shown above is the only iterative one that preserves conservation at every step. Rearranging terms and using the equivalency between the volume matrices and the sum over rows of the flux matrix, we obtain:

$$\begin{aligned} & \frac{v_i^{L,n+1} + v_i^{L,n}}{2} \left(\left(\rho_i^{n+1,(m+1)} - \rho_i^n \right) + \left(\rho_i^{n+1,(m+1)} - \rho_i^{n+1,(m)} \right) \right) \\ &= \sum_{j \neq i} l_{ij}^{n+1/2} \left(\rho_j^{n+1/2,(m)} - \rho_i^{n+1/2,(m)} \right). \end{aligned} \quad (3.12)$$

The second term of the left hand side introduces a possible perturbation in monotonicity depending of the difference between iterations. In practice, this term tends to zero as the iteration converges and possibly can be damped sufficiently by the excessive diffusion of the low order. In our simulations, we have never noticed overshoots/undershoots due to this term after running the predictor-multi-corrector for three times.

4. Antidiffusive flux computation. The low-order solution is in general overly diffusive, while the high-order solution can create new extrema or be physically incorrect. The secret of FCT relies in adding a weighted amount of the high-order solution to the low-order one. To achieve this, we first must compute an estimate of the difference between the two schemes, called the raw antidiffusive fluxes. Suppose for the moment that the low-order scheme could have been computed using a different time discretization from that of the high-order solution, to prove that they have to be the same in order to use the FCT algorithm. Then,

$$\begin{aligned} P_i &= \sum_{j \neq i} \left(\left(k_{ij}^{n+1/2} \left(\bar{\rho}_j^{n+1/2} - \bar{\rho}_i^{n+1/2} \right) - l_{ij}^{n+\theta} \left(\widehat{\rho}_j^{n+\theta} - \widehat{\rho}_i^{n+\theta} \right) \right) + v_{ij}^n \left(\rho_j^n - \rho_i^n \right) \right) + \dots \\ &\dots + \sum_j k_{ij}^{n+1/2} \bar{\rho}_i^{n+1/2} - \sum_j k_{ij}^{n+\theta} \widehat{\rho}_i^{n+\theta} + v_i^{L,n+1} \bar{\rho}_i^{n+1} - v_{ij}^{n+1} \bar{\rho}_j^{n+1}, \end{aligned} \quad (4.1)$$

where $\bar{\rho}_i$ is the density solution of the high-order scheme and $\widehat{\rho}_i$ is the solution of the low-order one. If we add this flux \mathbf{P} to the low-order scheme, we recover the high-order one. The solution we would like is an intermediate between the two: it must preserve positivity and LED but it must be as minimally diffusive as possible. To achieve this, the antidiffusive fluxes are applied after weighting them from 0 to 1 (the first value corresponding to the low-order solution and the last to the high-order). Zalesak's algorithm is widely used for this purpose and will be described in the next section.

Zalesak's algorithm however imposes some restrictions to the form of P_i . First, we need to write $P_i = \sum_{j \neq i} f_{ij}$, with the property $f_{ij} = -f_{ji}$. This can only be achieved if we use the same density (either the low-order or the high-order one) to estimate the flux. The structure of the consistent and lumped volume matrices calls for this. In addition, we need $\theta = 1/2$:

this value is chosen to guarantee the skew-symmetry property of the fluxes that cannot be achieved otherwise (we know that $\mathbf{K} - \mathbf{L} = -\mathbf{D}$ and that $d_{ij}(\rho_j - \rho_i) = -d_{ji}(\rho_i - \rho_j)$). With these assumptions, each individual flux reads:

$$\begin{aligned} f_{ij}(\rho^{n+1}, \rho^n) \\ = -d_{ij}^{n+1/2} \left(\frac{\rho_j^{n+1} + \rho_j^n}{2} - \frac{\rho_i^{n+1} + \rho_i^n}{2} \right) - v_{ij}^{n+1} (\rho_j^{n+1} - \rho_i^{n+1}) + v_{ij}^n (\rho_j^n - \rho_i^n). \end{aligned} \quad (4.2)$$

5. An iterative algorithm for the remeshing step. An iterative algorithm, similar to that proposed by Kuzmin and Möller in Chapter 6 of [2], can be implemented. This algorithm computes the antidiffusive flux in a few steps. At each iteration, we obtain a new positivity preserving and LED solution that becomes a “new” low-order solution for which we can compute new antidiffusive fluxes. As the iteration progresses, the amount of additional flux needed decreases.

First, consider the partial problem:

$$v_i^{L,n+1/2} \tilde{\rho}_i = v_i^{L,n} \rho_i^n + \frac{1}{2} l_{ij}^{n+1/2} \rho_j^n, \quad (5.1)$$

with $v_i^{L,n+1/2} = (v_i^{L,n+1} - v_i^{L,n})/2$. That is, we obtain an explicit low-order solution for the density at the half-step ($\tilde{\rho} = \tilde{\rho}^{n+1/2}$). This solution can be updated by adding antidiffusive fluxes. This is going to be performed in various steps. At each of them, we solve:

$$v_i^{L,n+1/2} \tilde{\rho}_i^{(m+1)} = b_i^{(m)}, \quad (5.2)$$

where $\mathbf{b}^{(m)}$ is defined as:

$$b_i^{(m)} = b_i^{(m-1)} + \sum_j \alpha_{ij}^{(m)} \Delta f_{ij}^{(m)}, \quad (5.3)$$

with:

$$b_i^{(0)} = v_i^{L,n} \rho_i^n + \frac{1}{2} l_{ij}^{n+1/2} \rho_j^n, \quad (5.4)$$

and

$$\Delta f_{ij}^{(m)} = f_{ij}^{(m)} - g_{ij}^{(m)}, \quad (5.5)$$

$$g_{ij}^{(m)} = g_{ij}^{(m-1)} + \alpha_{ij}^{(m)} \Delta f_{ij}^{(m)}. \quad (5.6)$$

This way, at each step of the iteration, we add the flux corresponding to the difference between what is needed and what was previously added. The antidiffusive fluxes are computed using Equation (4.3) with the last iterate solution as the estimate for the densities. The solution of (5.2) remains positivity-preserving as long as the data at time step n is monotone and the additional fluxes are calculated using Zalesak’s limiter. The Δf_{ij} values shrink when the number of performed iterations increase. Normally, a very low number of iterations is needed to get acceptable results. Note that this algorithm remeshes exactly a constant solution only if the volume matrix at the half step is used because $v_i^{L,n+1/2} = (v_i^{L,n+1} + v_i^{L,n})/2 = v_i^{L,n} + \frac{1}{2} \sum_j k_{ij}^{n+1/2}$ from Equation (3.8).

After this, we update the solution to the whole step. This is achieved by solving:

$$\left(\mathbf{V}^{L,n+1} - \frac{1}{2} \mathbf{L}^{n+1/2} \right) \rho^{n+1} = \mathbf{b}^{(last)}. \quad (5.7)$$

This can be done iteratively by:

$$v_i^{L,n+1} \rho_i^{n+1,(m+1)} = \frac{1}{2} \mathbf{L}^{n+1/2} \rho_i^{n+1,(m)} + b_i^{(last)}, \quad (5.8)$$

where the first guess is given by the last density value obtained in the previous loop. We note that monotonicity could be compromised if we use a low number of iterations (see Equation (3.12)). A typical number of iterations for this step is three.

Finally, we can add a bit more antidiffusion to the solution by using the same multistep algorithm that was used at the half step:

$$v_i^{L,n+1} \rho_i^{n+1,(m+1)} = v_i^{L,n+1} \rho_i^{n+1,(m)} + \alpha_{ij}^{(m)} \Delta f_{ij}^{(m)}, \quad (5.9)$$

where $\Delta f_{ij} = f_{ij} - g_{ij}$. The best results are obtained when g_{ij} is not reset for this iteration and is kept with the value it had after the half-point antidiffusion addition.

The number of iterations that need to be run to get accurate results varies with the problem. A pure advection problem looks better as we add more iterations at the half-step or at the end. This is not usually the case for the remesh (typically, the one-dimensional remesh proposed in [5]). Normally, a strategy consisting of one half-step and one final antidiffusion addition yields good results for straight shocks. For other tests, like a triangle 1D function or a bump, it can produce terracing (the final function looks like a ladder). If the half-step antidiffusion addition is eliminated, the results are more robust (i.e. no appreciable terracing is observed), but somewhat more dissipative. We have observed that using more than one iteration for any of the antidiffusive steps normally gives poor results for continuous functions. The algorithm tries to convert, for example, a bump function, into a square or terraced function. Finally, if the final antidiffusive flux addition is not used, the solution looks less sharp for straight discontinuities, producing a “shark fin” shape.

In conclusion, the two best strategies are the one half-step and one final antidiffusion addition, which is less diffusive but can produce terracing in some cases and the final addition of antidiffusion, which gives a very robust algorithm and is probably more suitable for multivariable problems.

5.1. Zalesak’s algorithm. Zalesak’s algorithm [12] is used to obtain the weights α_{ij} in a consistent way. Before formulating the algorithm, we can consider setting to zero all the antidiffusive fluxes that satisfy $\Delta f_{ij}(\rho_j - \rho_i) < 0$. This is because in this case, the differential fluxes are adding diffusion instead of eliminating it. The aim of the algorithm is to preserve the positivity and LED constraints by adding pieces of the high-order solution where possible (i.e. smooth regions). To do this, we first split the total flux P_i in positive and negative fluxes:

$$P_i^\pm = \sum_{j \neq i} \max_{\min} \{0, \Delta f_{ij}\}. \quad (5.10)$$

This way, we estimate the possibility of overshooting a maximum/minimum of the function at the node i only by considering the positive/negative contributions, respectively. We define a new quantity Q which estimates how far we are from the possible extrema:

$$Q_i^+ = \rho_i^{\max} - \rho_i, \quad (5.11)$$

$$Q_i^- = \rho_i^{\min} - \rho_i. \quad (5.12)$$

To estimate the maximum and minimum values, we just have to look at the neighboring nodes because the shape functions have compact support. For the iterative algorithm, we impose monotonicity by estimating these values using the maximum and minimum of the previous

iterate and the solution at step n . Then, in order to prevent the formation of spurious undershoots or overshoots, the antidiffusive flux f_{ij} must be multiplied by:

$$R_i^\pm = \begin{cases} \min\{1, v_i^{L,n+1} Q_i^\pm / P_i^\pm\} & \text{if } P_i^\pm \neq 0, \\ 1 & \text{if } P_i^\pm = 0. \end{cases} \quad (5.13)$$

Finally, we must satisfy the balance of fluxes for global conservation (i.e., $\alpha_{ij} = \alpha_{ji}$). This is achieved by:

$$\alpha_{ij} = \begin{cases} \min\{R_i^+, R_j^-\} & \text{if } \Delta f_{ij} \geq 0, \\ \min\{R_i^-, R_j^+\} & \text{if } \Delta f_{ij} < 0. \end{cases} \quad (5.14)$$

Taking the minimum value of the pair, we take the most conservative option to eliminate any possible overshoot/undershoot. More information about Zalesak's algorithm can be found in Chapter 2 of [2].

6. Extension to a system of equations. The FCT iterative algorithm can be extended to a multivariable system. The remapping step is equivalent to the advection of the conserved quantities. The most common procedure is to remap ρ , $\rho \mathbf{v}$ and ρE for the Euler equations, where \mathbf{v} is the velocity field and E the specific total energy ($E = e + \frac{\mathbf{v} \cdot \mathbf{v}}{2}$, with e the specific internal energy).

We can construct the high-order scheme the same way we did for the density. The spatial discretization of the conserved variables is performed using the same shape functions for the state vector and for the convective fluxes (generalized Galerkin). This way, for the state vector, we have:

$$\rho(x) = \sum_i \rho_i \phi_i(x), \quad (6.1)$$

$$\rho \mathbf{v}(x) = \sum_i \rho_i \mathbf{v}_i \phi_i(x), \quad (6.2)$$

$$\rho E(x) = \sum_i \rho_i E_i \phi_i(x), \quad (6.3)$$

and for the fluxes:

$$\rho(x) \mathbf{u}(x) = \sum_i \rho_i \mathbf{u}_i \phi_i(x), \quad (6.4)$$

$$\rho \mathbf{v}(x) \otimes \mathbf{u}(x) = \sum_i \rho_i \mathbf{v}_i \otimes \mathbf{u}_i \phi_i(x), \quad (6.5)$$

$$\rho E(x) \mathbf{u}(x) = \sum_i \rho_i E_i \mathbf{u}_i \phi_i(x). \quad (6.6)$$

The construction of the volume and stiffness matrices under these assumptions is trivial. They have the same structure as in the case of the density:

$$\mathbf{V}^{n+1} \mathbf{y}^{n+1} - \mathbf{V}^n \mathbf{y}^n = \mathbf{K}^{n+1/2} \mathbf{y}^{n+1/2}, \quad (6.7)$$

where \mathbf{y} is the vector of nodal quantities of any or the components of the state vector and \mathbf{V} and \mathbf{K} are defined in Equations (2.5) and (2.6).

Therefore, to construct a low-order method from this one we follow the same procedure that was used for the density case, constructing a low-order stiffness matrix \mathbf{L} from the high-order counterpart by adding a symmetric matrix \mathbf{D} in the way described in Equation (3.4) and substituting the consistent volume matrix by the lumped one.

The system can be solved independently for each conserved variable. Another possible strategy for directly obtaining the primitive variables is to solve first for the density, then construct a new lumped mass matrix $m_i^L = v_i^L \rho_i$ and stiffness matrix $l_{ij}^L = l_{ij} \rho_j$, and then solve directly for the velocity. The same strategy applies to the energy.

The low-order scheme is chosen to preserve monotonicity and LED constraints for the conserved variables. However, this is not always true for the primitive variables; for example, in a jump, if the density is smoothed more than the linear momentum, we most likely obtain a new maximum/minimum of the velocity. It appears that there is no way to circumvent this issue and we need to trust that the low-order scheme is dissipative enough to prevent this potential problem.

6.1. Antidiffusive correction for a system. The construction of the low-order method was just an extension of the algebraic case. The choice of a strategy to add antidiffusive fluxes to the low-order solution is not unique however and still subject of research. Here, we propose a strategy inspired by [3].

The fluxes can be computed for each one of the conserved variables in the same way we did for the density. Our interest is not to prevent the conserved variables from going beyond certain extrema, but to do this with the primitive variables (for example, the density, the internal energy, or the pressure). So, we must estimate a primitive variable antidiffusive flux and weight it to prevent overshoots and undershoots.

One possible way of doing this is supposing that the complete solution of any variable is just the low-order solution with an addition of a small flux such that the product of two fluxes conveniently non-dimensionalized is much less than unity. This way $y_i = \widehat{y}_i + f_{ij}^y$, where y is any variable and the hat denotes the low-order solution. For the linear momentum flux we get:

$$\widehat{\rho_i \mathbf{v}_i} + f_{ij}^{\rho \mathbf{v}} = (\widehat{\rho_i} + f_{ij}^{\rho}) (\widehat{\mathbf{v}_i} + f_{ij}^{\mathbf{v}}), \quad (6.8)$$

and doing some algebra and using that $\widehat{\rho_i \mathbf{v}_i} = \widehat{\rho_i} \widehat{\mathbf{v}_i}$:

$$f_{ij}^{\mathbf{v}} = \frac{f_{ij}^{\rho \mathbf{v}} - \widehat{\mathbf{v}_i} f_{ij}^{\rho}}{\widehat{\rho_i}}. \quad (6.9)$$

The same principle can be applied to the total energy flux to get the internal energy flux:

$$f_{ij}^e = \frac{f_{ij}^{\rho E} - \widehat{\mathbf{v}_i} \cdot f_{ij}^{\rho \mathbf{v}} + \frac{\widehat{\mathbf{v}_i} \cdot \widehat{\mathbf{v}_i}}{2} f_{ij}^{\rho} - \widehat{e_i} f_{ij}^{\rho}}{\widehat{\rho_i}}. \quad (6.10)$$

For the iterative algorithm described in section 5, the low-order solution is just the previous estimate. Note that these new fluxes do not have the skew-symmetry property $f_{ij} = -f_{ji}$; however, Zalesak's algorithm can be generalized to be applicable to them. The first part of the algorithm remains the unchanged (Equations (5.10) through (5.14)) repeating the algorithm for each of the primitive variables.

Experience tells us that a synchronized algorithm, which means that the same flux correction is applied to all fluxes, is preferable, because it avoids possible spurious oscillations. Two possible choices for picking the final weight are: $\alpha_{ij} = \min(\alpha_{ij}^{\rho}, \alpha_{ij}^{\mathbf{v}}, \alpha_{ij}^e)$ or using a progressive algorithm where the weights are computed for one of the primitive variables and applied to the conserved fluxes. Then the flux for the second primitive variable is computed using the new fluxes and new weights are obtained, repeating this with all the variables that need to be limited.

Note that the computation of weights α for each primitive variable is not necessary. We can compute the weights only for the variables that we want to preserve between bounds (e.g. ρ and e) and compute a global weight using the minimum of these two. This and the linearization performed to obtain the primitive fluxes can give rise to spurious overshoots and undershoots. An easy way of avoiding this is by the addition of a failsafe algorithm [3]. The failsafe algorithm detects when an overshoot/undershoot occurs and decreases the amount of antidiffusive flux added to the particular node where this happens, and their neighbors, in order to preserve conservation. This can be done by subtracting all the antidiffusive contributions at once to obtain the low-order solution or doing this in various steps. At each step, we subtract a proportional part of the flux until we cancel the spurious oscillations.

7. Test Results. In this section, we show some typical test cases that are used to measure the accuracy of the method. For simplicity, we use a one-dimensional version of the code. First, results for one variable are shown and in the second part of this section we analyze the multivariable algorithm. The mesh motion is based on the one-dimensional remesh strategy described in [5]:

$$x(\epsilon, t) = x_{min} + (x_{max} - x_{min})\tilde{\epsilon}(\epsilon, t), \quad (7.1)$$

$$\tilde{\epsilon}(\epsilon, t) = (1 - \alpha(t))\epsilon + \alpha(t)\epsilon^3, \quad (7.2)$$

$$\alpha(t) = \frac{\sin 4\pi t}{2}, \quad (7.3)$$

for $0 \leq \epsilon \leq 1$ and $0 \leq t \leq 1$. The node positions are $x_i^k = x(\epsilon_i, t^k) = x\left(\frac{i-1}{N}, \frac{k}{k_{max}}\right)$, where N is the number of cells and k_{max} the number of pseudo-time steps. Typical tests are run for the pairs $(N, k_{max}) = (64, 320), (128, 640), (256, 1280)$. Note that the ratio between the two is constant, leaving the CFL number unchanged. In Figure 7.1 the mesh motion is shown for $N = 24$ and $k_{max} = 16$.

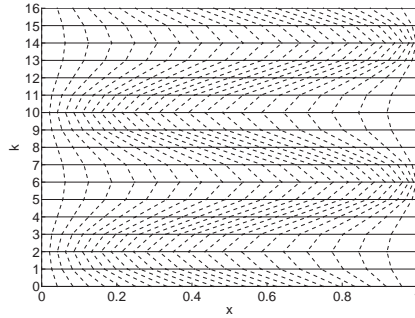


FIG. 7.1. Mesh motion for $N = 24$ and $k_{max} = 16$. Under this motion, each cell completes two sinusoidal cycles of expansion and contraction

7.1. Results for one variable. The typical tests that are run for testing an advection code are the square, triangle and bump shapes. These tests are applied to the remesh in Figure 7.2. As described at the end of section 5, the best results for the remapping are obtained adding antidiffusive fluxes either one at half-step and one at final step (called FCT2 in following figures) or just adding the antidiffusive flux at the end (called FCT1). To show this, the following figures compare the exact solution, the low-order solution and the solution obtained with the FCT algorithm using the two different antidiffusion strategies for $N = 128$ and $k_{max} = 640$. Table 7.1, along with the figures, shows that the FCT2 algorithm is better for

the square shape. However, the error of FCT2 and FCT1 is comparable for the triangle and the bump (FCT1 even performs better for the bump). While the more dissipative algorithm adapts better to the shape of the curve, the less dissipative gives a better peak value.

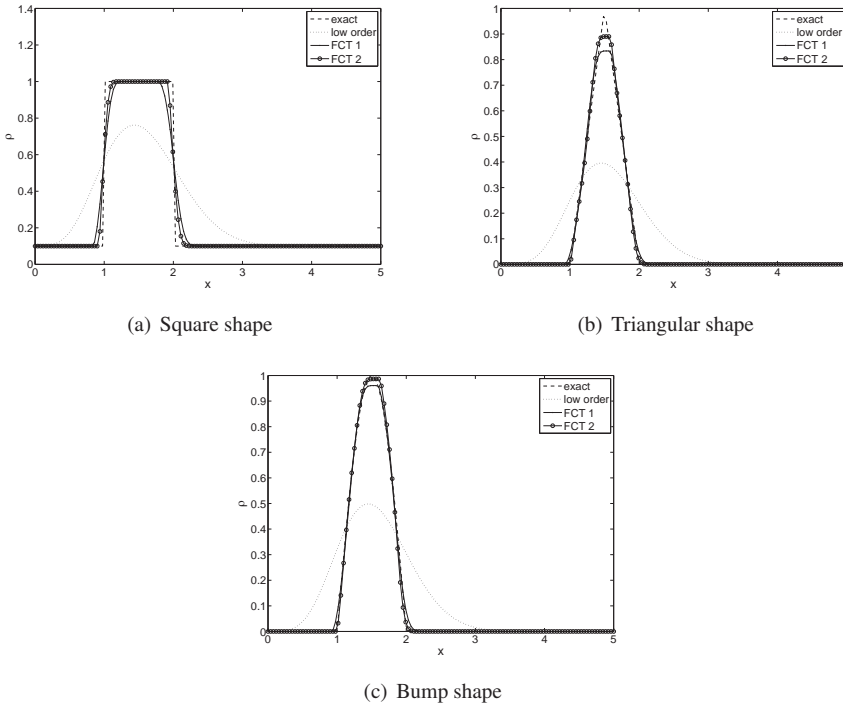


FIG. 7.2. Remeshed solution for square, triangular and bump shapes. The method FCT2 produces a better result for the square than FCT1, which is more dissipative since only one antidiffusion addition is performed. Zalesak's algorithm attempts to transform the triangular and bump shapes into a square shock or a succession of them (terracing), especially when applied multiple times or the resolution increases. This way, FCT1 is normally more robust although FCT2 offers better peak values.

TABLE 7.1
 L_1 error for $N = 128$, $k_{max} = 640$

Shape	Low Order	FCT1	FCT2
Square	0.6434	0.1292	0.0741
Triangle	0.3968	0.0368	0.0306
Bump	0.4894	0.0247	0.0249

The most demanding test found in the literature is the exponential shock ([5], [9],[11]). The density profile for this test is composed of an exponential increase followed by a shock and represents an out-going explosion. This test is run using FCT1 and FCT2 in Figure 7.3 for three pairs of (N, k_{max}) in order to analyze convergence of the method. FCT2 gives a higher density peak but shows terracing. FCT1 is more diffusive but behaves better for the exponential part of the test. Table 7.2 shows that indeed the errors in the final result are paired for both methods.

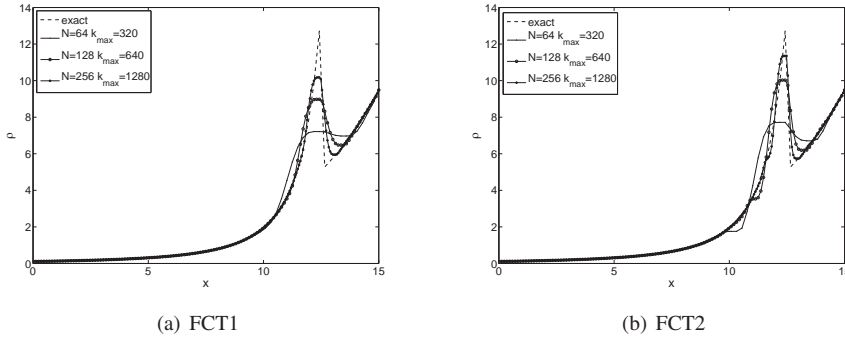


FIG. 7.3. Remeshed solution for the exponential jump. As predicted, FCT2 offers a best peak value but terracing is appreciated before the peak at the exponential part of the profile. FCT1 does not produce terracing.

TABLE 7.2

L_1 error and convergence rate (C.R.) for the exponential shock problem with FCT1 and FCT2

Mesh	FCT1		FCT2	
	Error	C.R.	Error	C.R.
$k_{max} = 320, N = 64$	4.6840		4.6624	
$k_{max} = 640, N = 128$	3.1614	0.5672	2.8510	0.7096
$k_{max} = 1280, N = 256$	1.7572	0.8473	1.3873	1.0392

7.2. Results for a multivariable system. Two one-dimensional multivariable tests can be found in [5] and [11]. The first one is a simple jump for density, velocity and internal energy, while the second one is an extension of the exponential jump described for one variable. The density jump is complemented by a linear function in velocity and internal energy followed by a shock.

The results shown are obtained by limiting the density and internal energy. Velocity can be limited too, but in multidimensions that would involve the computation of a limit for each of the three components of the vector. A limiting for the kinetic energy is a possible substitute for this. The failsafe algorithm is used to avoid possible overshoots and undershoots.

For the simple jump, the limiting of density and internal energy is enough to preserve the monotonicity of all the variables of the system. Figure 7.4 shows results for FCT2. Table 7.3 shows how the FCT2 behaves better than the FCT1 for this type of problem, since straight jumps do not normally suffer from terracing.

TABLE 7.3

L_1 error and Convergence Rate (C.R.) for the simple jump problem with FCT1 and FCT2

Mesh	FCT1			FCT2		
	ρ	v	e	ρ	v	e
$k_{max} = 320, N = 64$	0.0816	0.0356	0.0232	0.0483	0.0222	0.0149
C.R.	0.7038	0.7075	0.6981	0.7763	0.7832	0.7762
$k_{max} = 640, N = 128$	0.0501	0.0218	0.0143	0.0282	0.0129	0.0087
C.R.	0.7019	0.7021	0.7004	0.7558	0.7444	0.7705
$k_{max} = 1280, N = 256$	0.0308	0.0134	0.0088	0.0167	0.0077	0.0051

Figures 7.5 and 7.6 show results for the exponential shock system. FCT1 gives smooth

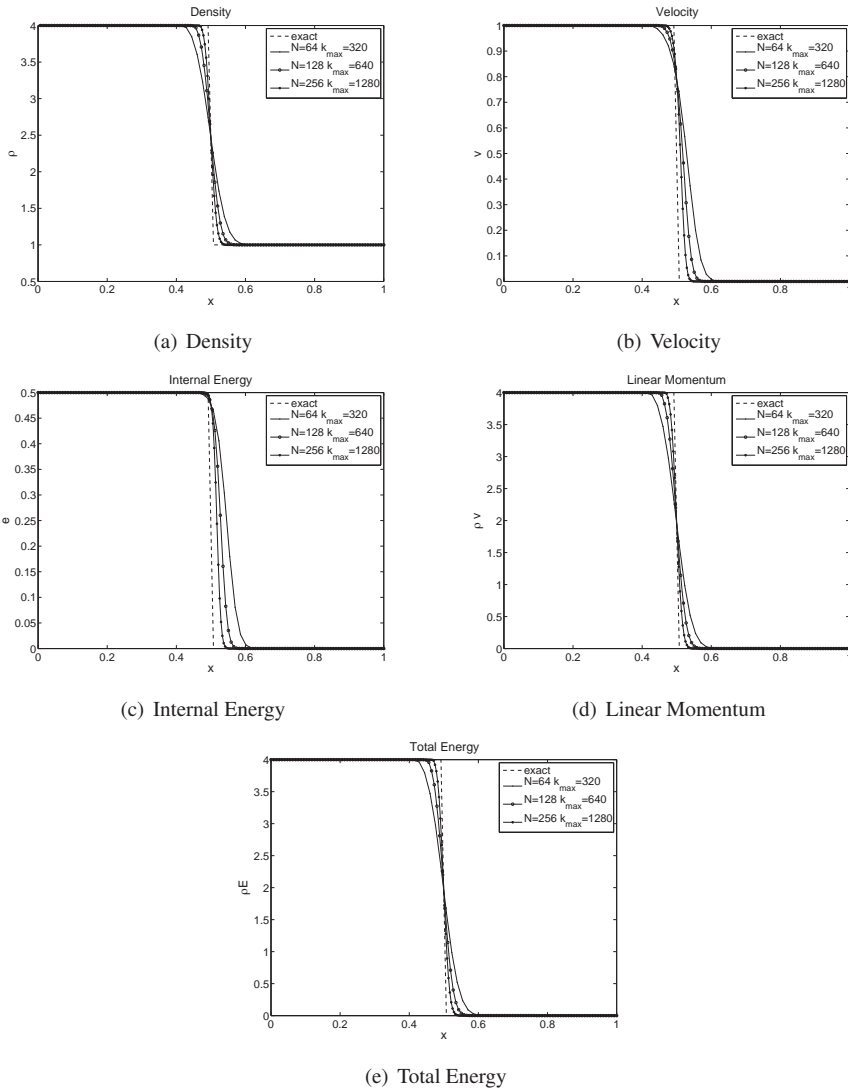


FIG. 7.4. Remesh of a simple jump. Density, linear momentum and total energy are conserved. Internal energy and density are limited to remain monotone. In this case, the methods leads to monotonicity of all variables.

results for the exponential part of the curve, while FCT2 provides a better peak result but some terracing. Table 7.4 reveals that FCT1 is close to the accuracy of FCT2 due to the absence of terracing, while for the simple shock the error obtained with the first was almost twice the error of the second in the L_1 norm.

8. Conclusions. We developed a Flux-Corrected Transport algorithm for the remeshing step of a Finite Element ALE method using an multi-step algorithm for the antidiffusive flux addition and solving the linear system with the aid of a predictor-multi-corrector algorithm. This way, the algorithm is fast and typically only requires one or two steps of antidiffusion addition and two or three steps of the multi-corrector to converge.

The results obtained are comparable with other results from the same field (i.e., [5] and [11], where finite-differences were used). Our tests reveal that the ideal number of

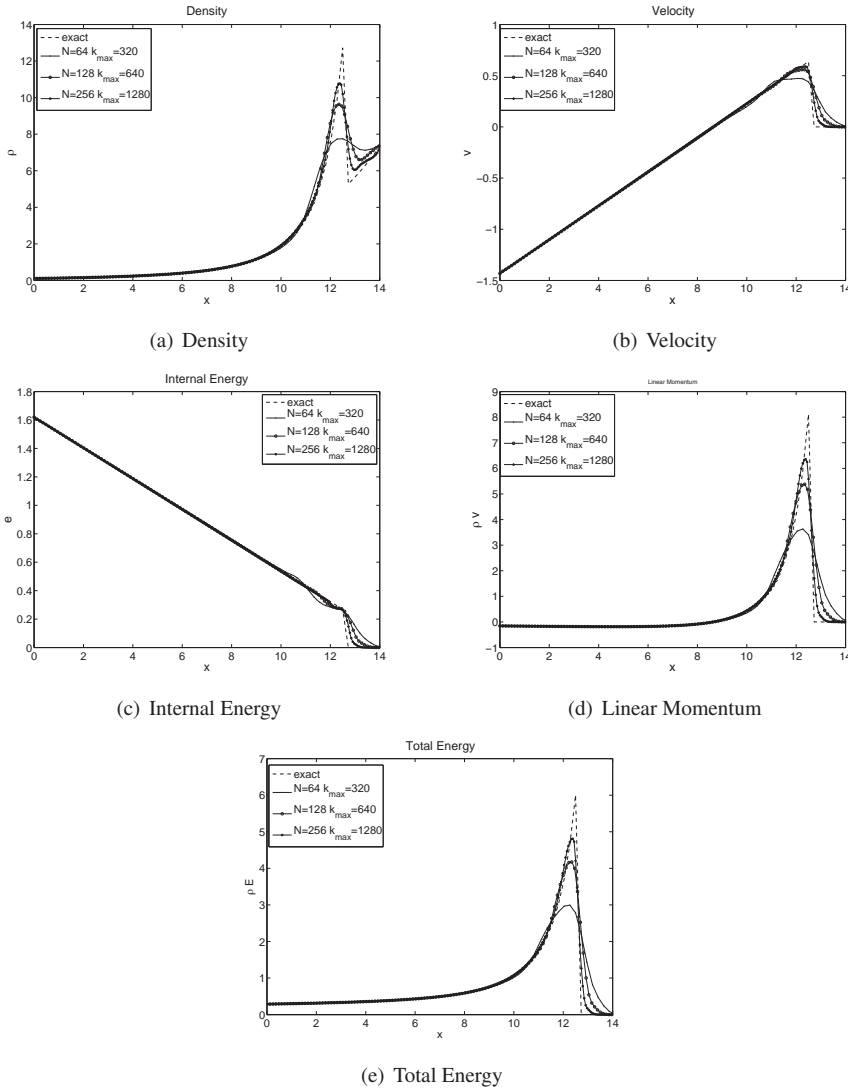


FIG. 7.5. Remesh of the exponential shock extended to velocity and internal energy with FCT1. Results do not show terracing and extra-diffusion. Constraining of multiple variables does not seem to influence the peak value for the density when compared to the previous one-variable results

antidiffusion-addition steps is between one and two. For straight jumps, having two or more steps proves to be better, because Zalesak's algorithm tends to transform any possible shape into straight shocks or a succession of them. For general problems with continuous functions, the results using one and two steps are comparable, the first one being more capable of correctly remeshing the original shape of the curve and the second giving more accurate values close to the possible peaks.

9. Acknowledgments. The authors want to acknowledge Edward Love, James Kamm, William Rider and John Shadid for their valuable comments on this article and help throughout the process of algorithm design. A. López Ortega is supported by the PSAAP program, funded by the Department of Energy National Nuclear Security Administration under Award

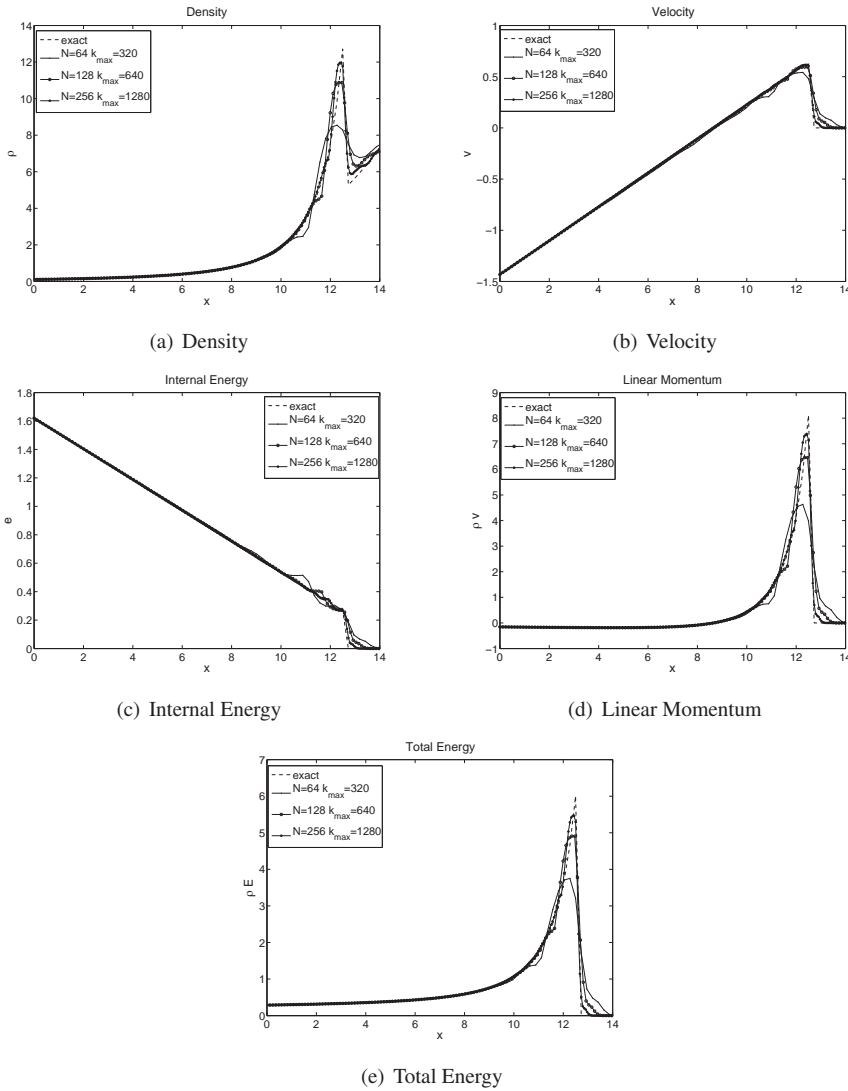


FIG. 7.6. Remesh of the exponential shock extended to velocity and internal energy with FCT2. Some terracing is appreciated but the peak values are closer to the original value than with FCT1.

Number DE-FC52-08NA29613.

REFERENCES

- [1] C. FLETCHER, *The group finite element formulation*, Computer Methods in Applied Mechanics and Engineering, 37 (1983), pp. 225–244.
- [2] D. KUZMIN, R. LÖHNER, AND S. TUREK, eds., *Flux-Corrected Transport: Principles, Algorithms and Applications*, Springer, 2005.
- [3] D. KUZMIN, M. MÖLLER, J. N. SHADID, AND M. SHASHKOV, *Failsafe flux limiting and constrained data projections for systems of conservation laws*. Preprint submitted to Elsevier, 2010.
- [4] D. KUZMIN AND S. TUREK, *Flux correction tools for finite elements*, Journal of Computational Physics, 175 (2002), pp. 525–538.
- [5] R. LISKÁ, M. SHASHKOV, P. VÁCHAL, AND B. WENDROFF, *Optimization-based synchronized flux-corrected con-*

TABLE 7.4
 L_1 relative error for the exponential shock problem with FCT1 and FCT2

Mesh	FCT1			FCT2		
	ρ	v	e	ρ	v	e
$k_{max} = 320, N = 64$	4.5413	0.3769	0.2097	4.5093	0.3033	0.1992
C.R.	0.4900	0.9589	1.0583	0.6546	1.0187	1.1575
$k_{max} = 640, N = 128$	3.2335	0.1939	0.1007	2.8646	0.1497	0.0893
C.R.	0.8329	0.7970	0.8363	0.9758	0.9629	1.0016
$k_{max} = 1280, N = 256$	1.8153	0.1116	0.0564	1.4565	0.0768	0.0446

servative interpolation (remapping) of mass and momentum for arbitrary lagrangian-eulerian methods, Journal of Computational Physics, 225 (2010).

- [6] R. LÖHNER, K. MORGAN, M. VAHDATI, J. BORIS, AND D. BOOK, *Fem-fct: Combining unstructured grids with high resolution*, Communications in Applied Numerical Methods, 4 (1988), pp. 717–729.
- [7] R. LÖHNER, K. MORGAN, J. PERAIRE, AND M. VAHDATI, *Finite element flux-corrected transport (fem-fct) for the euler and navier-stokes equations*, International Journal for Numerical Methods in Fluids, 7 (1987), pp. 1093–1109.
- [8] R. LÖHNER, K. MORGAN, AND O. ZIENKIEWICZ, *The solution of non-linear hyperbolic equation systems by the finite element method*, International Journal for Numerical Methods in Fluids, 4 (1984), pp. 1043–1063.
- [9] L. MARGOLIN AND M. SHASHKOV, *Remapping, recovery and repair on a staggered grid*, Computer Methods in Applied Mechanics and Engineering, 193 (2004), pp. 4139–4155.
- [10] G. SCOVAZZI AND T. J. HUGHES, *Lecture notes on continuum mechanics on arbitrary moving domains*, November 2007. Approved for public release, Sandia National Laboratories.
- [11] P. VÁCHAL AND R. LISKA, *Sequential flux-corrected remapping ale methods*, Numerical Mathematics and Advanced Applications, 16 (2006), pp. 671–679.
- [12] S. T. ZALESAK, *Fully multidimensional flux-corrected transport algorithms for fluids*, Journal of Computational Physics, 31 (1979), pp. 335–362.

MOLECULAR DYNAMICS SIMULATIONS OF SINGLE CONJUGATED POLYMER NANOPARTICLE

SABINA MASKEY*, FLINT PIERCE†, DVORA PERAHIA‡, STEVEN J. PLIMPTON§, AND GARY S. GREST¶

Abstract. Molecular dynamics simulations have been utilized to study the factors that hold nanoparticles formed by conjugated polymers in their collapsed conformations. Here we present simulations in which dialkyl poly *para* phenyleneethynylene (PPE), an optically active polymer, is used to make a single conjugated polymer nanoparticle (CPN). CPNs have potential applications in fluorescence imaging, bio-sensors and optoelectronic devices. The use of these nanoparticles depends on their stability in different solvents. We found that these polymer nanoparticles are stable and remain collapsed in a poor solvent but unravel in a good solvent.

1. Introduction. Conjugated polymers are organic semi-conducting materials which are electro-optically active having applications from sensing to organic electronics [1, 3, 11, 13]. The packing of these highly optically active polymers into nano dimensions enhance their potential for sensing and fluorescent probes. These polymers, whose natural conformation is stretched, can be forced experimentally [12, 16] into collapsed spherical conjugated polymer nanoparticles (CPNs). This raises questions regarding the factors that control their stability since this collapsed conformation is far from equilibrium. Nanoparticles have different properties than those of bulk materials and molecules. Much of the interest in nanoparticles arises from their large surface area and their small size which is comparable to that of many molecules together with their unique electro-optical and magnetic properties. Their electro-optical characteristics differ from that of bulk due to quantum confinement on the nanoscale.

Because of the immense technological potential of polymer dots- or CPNs and their far-from-equilibrium conformation within these nanoparticles, we probe the factors that retain the PPE molecules within their collapsed configuration. Molecular dynamics (MD) simulation is used to investigate the molecular conformation of confined PPEs into nanoparticles and elucidate the interactions that result in the formation of stable nanoparticles. The current study focuses on diethylhexyl *para* phenyleneethynylene (PPE) and compares the results to the same chain without side chains. The repeat unit of the dialkyl PPE is shown in Figure 1.1.

The backbone of PPE consists of alternate single and triple bonds in conjunction with aromatic rings as shown in Figure 1.1. The single bonds along the backbone allow the aromatic ring to freely rotate along the long axis of the molecule. When the aromatic rings are confined in a single plane, the overlap of π orbitals result in delocalization of electrons along extended segments of the backbone and the backbone of the PPE becomes fully conjugated. As a result, PPE are intrinsic semi-conductor and often emit light on excitation. Small angle neutron scattering (SANS) have shown that dialkyl PPE in toluene, which is a good solvent for the backbone, forms a complex phase diagram depending on the concentration and temperature including molecular solutions, micellar structures and fragile gels [7]. The conformation of the PPE molecules in these three phases depends on their degree of constraint.

Conformational studies of single chains of various di-substituted alkyl PPEs have been carried out by using MD simulations in explicit and implicit solvents. These studies have shown that these PPE molecules form rigid rod-like structures and do not form random coils

*Department of Chemistry, Clemson University, smaskey@clemson.edu,

†Sandia National Laboratories, fpierce@sandia.gov,

‡Department of Chemistry, Clemson University, dperahi@ces.clemson.edu,

§Sandia National Laboratories, sjplimp@sandia.gov,

¶Sandia National Laboratories, gsgrest@sandia.gov

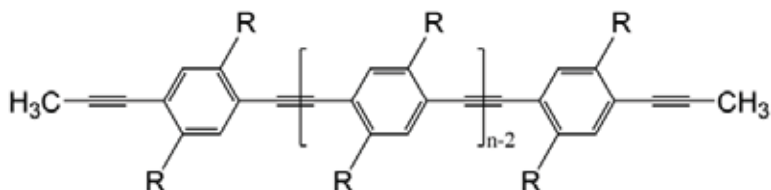


Fig. 1.1. The chemical structure of poly para phenyleneethynylene(PPE). Here R represents ethylhexyl and H.

or collapsed structures for degrees of polymerization up to $n=2000$ for a number of solvents [6].

Recently experimental studies have been performed on single conjugated polymer nanoparticles (CPNs) to study their properties and stability. Their optical properties differ significantly from those of free molecules and in addition are highly fluorescent also. McNeill and co-workers have demonstrated that several hydrophobic conjugated polymers, when dissolved in a good solvent for overall polymer and poured into a poor solvent such as water, which is miscible with organic solvent and sonicated, assist in the formation stable nanoparticles [12, 16]. After that organic solvent is removed, stable CPNs are dispersed in a aqueous medium. Depending on the concentration and conformation of the polymers, the size and properties of these CPNs can be tuned. The confinement of these unimolecular nanoparticles resulted in a clear change in their photophysics. Nanoparticles consisting of conjugated polymers have emerged as bright fluorescent markers. McNeill and coworkers investigated optical properties and energy transfer phenomena of these CNPs [14]. They have shown that these CPNs have potential applications in fluorescence imaging [17]. Other studies on CPNs formed by PPE containing amines have shown their use for imaging of cells in a tissue [9]. These confined PPE CNPs remain in suspension and are optically active for months after their preparations. The main reason that these CPNs are gaining interest is due to the fact they are easy to synthesize as well as can be easily tuned according to desired application by choice of conjugated polymers and above all are biocompatible.

Using MD simulations, the current study is aimed to determine conformation and stability of the conjugated PPE polymer nanoparticles formed in both good and poor solvents. Computationally, it is challenging to imitate the exact experimental procedures used to make these CNPs. Mimicking the confinement of these CNPs, we have enclosed the PPE chain in a large sphere. We then slowly decreased the size of the sphere collapsing the polymer chain as the function of time until the sphere is compressed to the maximum extent possible after which the collapsed polymer nanoparticle was allowed to relax in either a good or a poor solvent.

2. Simulation Methodology and Results. The PPEs used in this work have been modeled using the fully atomistic Optimized Parameter for Liquid Simulator-All Atoms (OPLS-AA) potential of Jorgensen *et al.* [4, 5]. The OPLS-AA is comprised of several potential terms,

$$U_{OPLS-AA} = U_{nb} + U_{bond} + U_{ang} + U_{tor} + U_{imp} \quad (2.1)$$

Nonbonded interactions U_{nb} are a sum of standard 12-6 Lennard-Jones (LJ) and electrostatic potentials [4, 5],

$$U_{nb}(r_{ij}) = U_{LJ} + U_{coul} = 4\epsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r_{ij}} \right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}} \right)^6 \right] + k_{coul} \frac{q_i q_j}{r_{ij}} \quad (2.2)$$

where ϵ_{ij} is the LJ energy and σ_{ij} is the LJ diameter for atoms i and j , q_i and q_j are their partial charges. For atoms of different species, geometric mixing rules are used, $\epsilon_{ij} = (\epsilon_i \epsilon_j)^{\frac{1}{2}}$ and $\sigma_{ij} = (\sigma_i \sigma_j)^{\frac{1}{2}}$. Nonbonded interactions are calculated between all atomic pairs on different molecules in addition to all pairs on the same molecule separated by three or more bonds. The interaction is reduced by a factor of 1/2 for atoms separated by three bonds. All LJ interactions were cut off at 12Å. All electrostatic interactions for atom pairs closer than 12Å are calculated in real space, while those outside this range are calculated in reciprocal (Fourier) space by the use of a standard particle-particle particle-mesh (PPPM) algorithm [2] with a precision of 10^{-4} . Direct covalent bonds between atoms are modeled in the OPLS as harmonic potentials of the form

$$U_{bond}(r_{ij}) = k_r (r_{ij} - r_0)^2 \quad (2.3)$$

where r_{ij} is the distance between atoms i and j and r_0 is their equilibrium separation. Harmonic forms are also used to describe the angle bending potential of directly bonding chains of three atoms (i,j,k)

$$U_{ang}(\theta_{ijk}) = k_\theta (\theta_{ijk} - \theta_0)^2 \quad (2.4)$$

θ_{ijk} is the angle between the vectors \mathbf{r}_{ji} and \mathbf{r}_{jk} , and θ_0 is the equilibrium value. The torsional (dihedral) component of the OPLS potential is given by

$$U_{tor}(\phi) = \sum_{n=1}^4 \frac{k_n}{2} [1 - (-1)^n \cos(n\phi)] \quad (2.5)$$

where ϕ is the dihedral angle and k_n is energy [15]. In the OPLS-AA framework, the out of plane (improper) potential has the same form as that of the torsional potential.

The PPE chains were built using a Polymer Builder of Material Studio from Accelrys Inc[®]. The samples were originally built using the polymer consistent force field (pcff) as OPLS-AA potential is not implemented in Material Studio. An in-house conversion utility was used to convert the Material Studio data files into LAMMPS data files and the force field as changed from pcff to OPLS-AA. All the simulations were performed using the LAMMPS classical MD code [8]. The equation of motion has been integrated using velocity-Verlet algorithm with a time step of $\Delta t = 1fs$. PPE nanoparticles are formed by compressing an isolated PPE chains into a final radii of 1 and 2 nm by enclosing them within a large spherical cavity whose radius is slowly reduced over 1 ns in a poor solvent. The cavity wall interacts with the PPE chains via a harmonic potential using the indent fix in the LAMMPS MD code. The PPE chains were collapsed to form a nanoparticle of a desired radius using NVE ensemble with a Langevin thermostat [10] with 100 fs damping time to regulate the temperature at 300 K. After compressing the PPE molecule to its final size, the enclosed cavity is removed and the compressed PPE nanoparticle is allowed to relax. Then simulations were run in an implicit good and an implicit poor solvents due to a computational limitation.

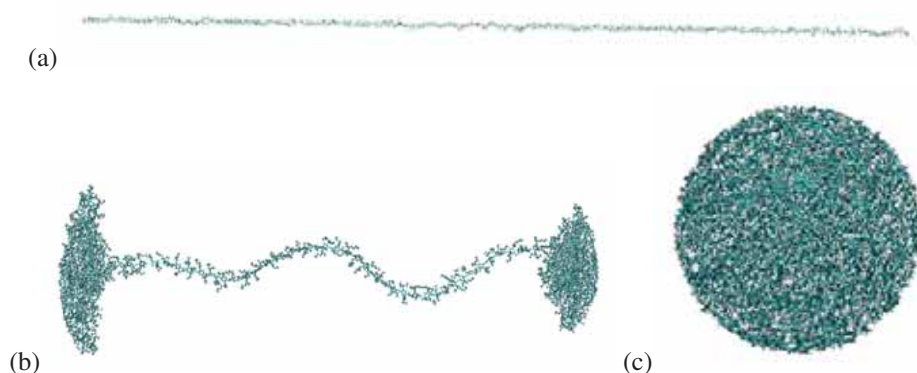


FIG. 2.1. Snapshots of diethylhexyl PPE for (a) $t = 0$ (b) $t = 0.5$ and (c) $t = 1$ ns. The final radius of the CPN is 2 nm. Here carbon is cyan and hydrogen is white.

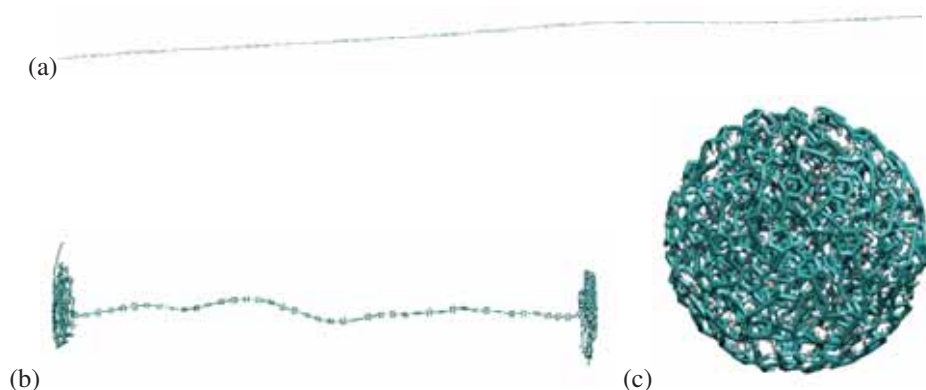


FIG. 2.2. Snapshots of PPE without side chains for (a) $t = 0$ (b) $t = 0.5$ and (c) $t = 1$ ns. The final radius of the CPN is 1 nm.

To model a solvent which is poor for both the backbone and side chains such as water, all the interactions were truncated at 12 Å. To model a good solvent for both backbone and side chains, all the Lennard-Jones interactions in Equation 2 were truncated at the potential minimum ($r_c = 2^{1/6}\sigma_{ij}$), producing a purely repulsive nonbonded forces.

Snapshots of nanoparticles as they are formed are shown in Figure 2.1 for diethylhexyl PPE and in Figure 2.2 for the PPE molecule without side chains, $R=H$. For comparison, Figures 2.1a and 2.2a, show the equilibrated PPE chain of length $n = 240$ repeat unit before enclosing it in the spherical cavity. As the radius of the cavity decreases the ends of the chains start to coil up gradually with time as shown in Figures 2.1b and 2.2b, finally forming a collapsed PPE nanoparticles. The final size of the CPN depends on the side chains. The maximum amount that CPNs could be compressed was radii of 1 nm for PPE without side chains and 2 nm for diethylhexyl PPE.

Once the PPE polymers were compressed, the indenter was removed, and the CPNs were allowed to relax in a solvent. Figures 2.3 and 2.4 show our results for PPE CPNs with and without side chains in a poor and a good solvent. Visually, we can see strong differences. In the poor solvent, the CPNs initially expand somewhat but remain compact over the course of the simulation. However, in the good solvent, these CPNs rapidly uncoil and continue to

unravel on the course of the run.

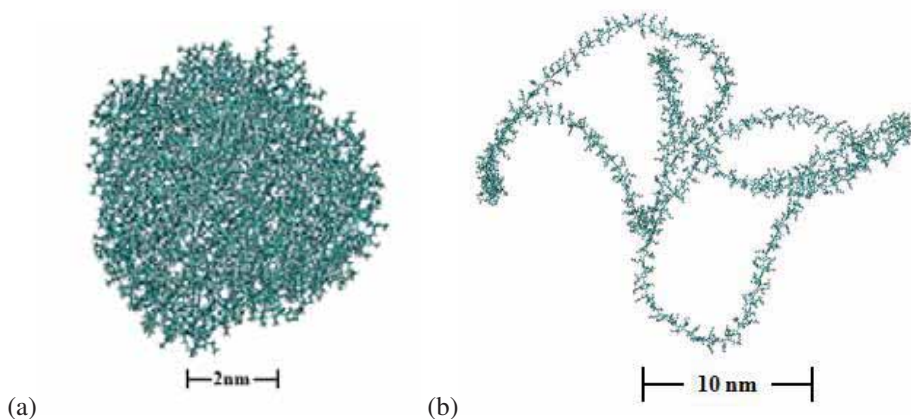


FIG. 2.3. Snapshots of diethyhexyl PPE CNPs in (a) poor solvent at 50 ns and (b) good solvent at 100 ns

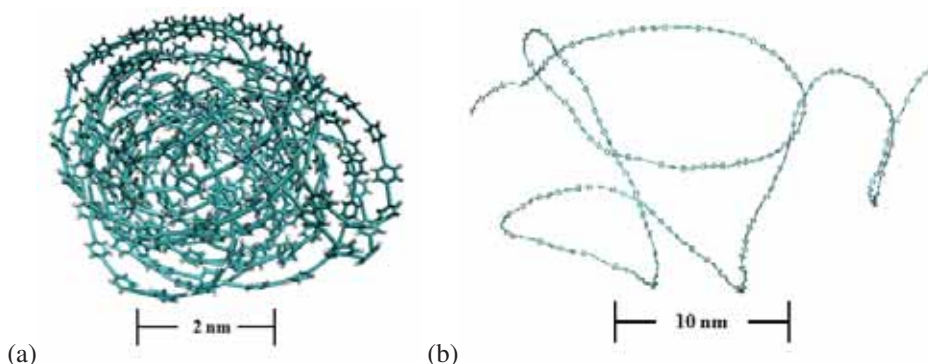


FIG. 2.4. Snapshots of PPE CNPs without side chains in (a) poor solvent at 50 ns and (b) good solvent at 100 ns

To quantify our results, we measured radius of gyrations, R_g of NPs as a function of time in different solvents. Figure 2.5 shows the radius of gyration R_g as a function of time for PPE CNPs with and without side chains in poor and good solvents. The radius of gyration R_g for the two CPNs in the poor solvent increase slightly at first and then remain stable for the period of the run. Hence, these CNPs are stable in poor solvents in agreement with the experiment. In the good solvent, however, R_g continues to increase over the course of the simulation as the chain unravels. This uncoiling takes place faster in case of CNP without side chains than with side chains which is supported by an increase in R_g values as in Figure 2.6. R_g rapidly increases in CNP without side chains than with side chain CNP.

3. Conclusions. Our molecular dynamics (MD) simulations have shown that collapsed PPE nanoparticles depending on solvents and side chains show different stability patterns. In a poor solvent, these polymers retain their collapsed conformations which is shown by the calculation of R_g . This stability pattern is consistent with the experimental measurements of R_g by SANS. However, in the good solvent the CPNs unravel rapidly and continue to uncoiling during the course of simulation. The size of the collapsed nanoparticles depends on the

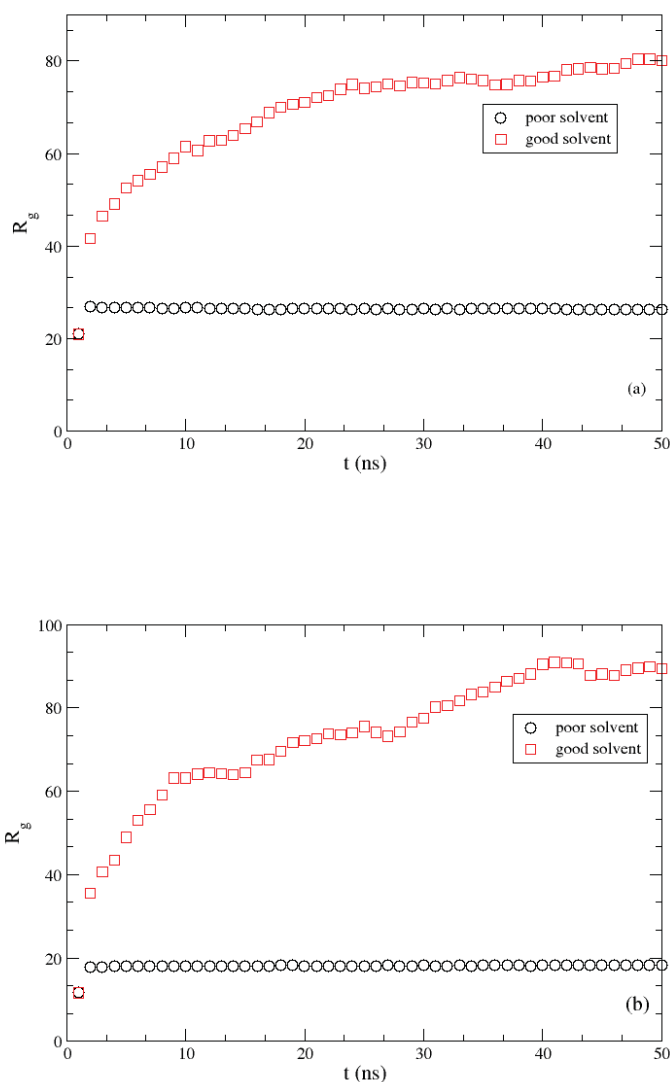


FIG. 2.5. Radius of gyration of PPE CNPs as a function of time for (a) diethylhexyl PPE and (b) for PPE without side chain in poor (circles) and good (squares) solvents.

type of side chains. For the first time, this study has provided an insight into the factors that control the stability of PPE polymers as nanoparticle. It seems that van der Waals interactions are sufficient to hold these CPNs together. To further our work, the behavior of these CPNs in explicit solvent models such as water and toluene will be studied to tune the interaction between the solvent and the polymers. Continuing with the study, we will investigate the conformations and stability of conjugated polymer nanoparticles using different lengths and types of side chains, sizes, and molecular weights in both implicit and explicit solvents as

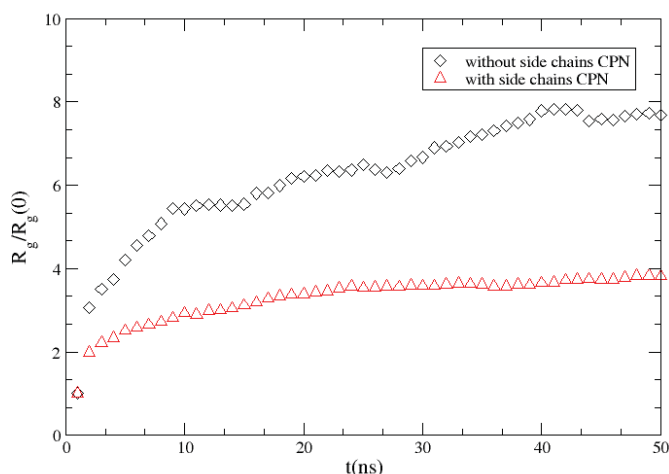


FIG. 2.6. Radius of gyration of PPE CNPs with (triangle) and without (diamond) side chain as a function of time in good solvent.

well as function of temperature.

4. Acknowledgement. This work was made possible by advanced computational resources deployed and maintained by Clemson Computing and Information Technology and we would also like to acknowledge the support of the staff from the cyber infrastructure and Technology Integration group, Clemson University. We thank the DOE for partial support of this work under contract No. ER46456. This work was performed, in part, at the Center for Integrated Nanotechnology, a U.S. Department of Energy, Office of Basic Energy Sciences user facility. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract No. DE-AC04-94AL85000.

REFERENCES

- [1] A. J. HEEGER, *Semiconducting and metallic polymers: The fourth generation of polymeric materials*, *Angew. Chem. Int. Ed.*, 113 (2001), pp. 2591–2611.
- [2] R. W. HOCKNEY AND J. W. EASTWOOD, *Computer Simulation Using Particles*, Adam Hilger-IOP, Bristol, 1988.
- [3] C. HOVEN, R. YANG, A. GARCIA, A. J. HEEGER, T.-Q. NGUYEN, AND G. BAZAN, *Ion motion in conjugated polyelectrolyte electron transporting layers*, *J. Am. Chem.*, 129 (2007), pp. 10976–10977.
- [4] W. L. JORGENSEN, J. D. MADURA, AND C. J. SWENSON, *Optimized intermolecular potential for liquid hydrocarbons*, *J. Am. Chem. Soc.*, 106 (1984), pp. 6638–6646.
- [5] W. L. JORGENSEN, D. S. MAXWELL, AND J. TIRADO-RIVES, *Development and testing of the opls all-atom force field on conformational energetics and properties of organic liquids*, *J. Am. Chem. Soc.*, 118 (1996), pp. 11225–11236.
- [6] S. MASKEY, F. PIERCE, G. S. GREEST, AND D. PERAHIA, *Conformational study of a single molecule of poly para phenyleneethylenes (ppes) in dilute solution*, *manuscript in preparation*.
- [7] D. PERAHIA, R. TRAIPIHOL, AND U. H. F. BUNZ, *From molecules to supramolecular structure: Self assembling of wirelike poly(p-phenyleneethylenes)s*, *Macromolecules*, 34 (2001), pp. 151–155.
- [8] S. PLIMPTON, *Fast parallel algorithms for short-range molecular dynamics*, *J. Comp. Phys.*, 117 (1995), pp. 1–19.

- [9] N. A. A. RAHIM, W. McDANIEL, K. BARDON, S. SRINIVASAN, V. VICKERMAN, P. T. C. SO, AND J. H. MOON, *Conjugated polymer nanoparticles for two-photon imaging of endothelial cells in a tissue model*, Adv. Mater., 21 (2009), pp. 3492–3496.
- [10] T. SCHNEIDER AND E. STOLL, *Molecular-dynamics study of a three-dimensional one-component model for disortive phase transitions*, Phys. Chem. B, 17 (1978), pp. 1302–1322.
- [11] B. J. SCHWARTZ, *Conjugated polymers as molecular materials: how chain conformation and film morphology influence energy transfer and interchain interactions*, Annual Rev. Phys. Chem., 54 (2003), pp. 141–172.
- [12] C. SZYMANSKI, C. F. WU, J. HOOPER, M. A. SALAZAR, A. PERDOMO, A. DUKES, AND J. McNEILL, *Single molecule nanoparticles of the conjugated polymer meh-ppv, preparation and characterization by near-field scanning optical microscopy*, J. Chem. Phys. B, 109 (2005), p. 8543.
- [13] S. W. THOMAS III, G. D. JOLY, AND T. M. SWAGER, *Chemical sensors based on amplifying conjugated polymers*, Chem. Rev., 107 (2007), pp. 1139–1386.
- [14] Z. TIAN, J. YU, C. WU, C. SZYMANSKI, AND J. McNEILL, *Amplified energy transfer in conjugated polymer nanoparticles tags and sensors*, Nanoscale, (2010).
- [15] E. K. WATKINS AND W. L. JORGENSEN, *Perfluoroalkanes: Conformational analysis and liquid-state properties from ab initio and monte carlo calculations*, J. Phys. Chem. A, 105 (2001), pp. 4118–4125.
- [16] C. F. WU, B. BULL, C. SZYMANSKI, K. CHRISTENSEN, AND J. McNEILL, *Multicolor conjugated polymer dots for biological fluorescence imaging*, ACS Nano, 2 (2008), p. 2415.
- [17] C. F. WU, C. SZYMANSKI, Z. CAIN, AND J. McNEILL, *Conjugated polymer dots for multiphoton fluorescence imaging*, J. Am. Chem. Soc, 129 (2007), p. 12904.

CHARGE TRAPS AND LOCAL ATOMIC RELAXATIONS IN AMORPHOUS SILICON DIOXIDE

NATHAN L. ANDERSON*, PETER A. SCHULTZ†, AND ALEJANDRO STRACHAN‡

Abstract. We apply density functional techniques that properly account for the electrostatics in the presence of periodic boundary conditions to report the defect levels of various charge transitions for a number of different defects in amorphous silicon dioxide. In addition, for each transition level, we provide the physical explanation of the atomic bonding rearrangements which occur upon electron or hole capture. We find that paramagnetic E'_γ and E'_β defects exist in the neutral charge state and are capable of trapping both electrons and holes. Statistical support for the oxygen vacancy originated dimerized model of the positively charged E'_δ defect is demonstrated. As the result of hole trapping by an undercoordinated silicon we show a significant number of stable positively charged overcoordinated oxygen defects result and are likely a prevalent source of trapped charge. Finally, evidence for the existence of an ill-established hole trap associated with an overcoordinated silicon floating bond defect is presented.

1. Introduction. Point defects in amorphous silicon dioxide (a -SiO₂) are known to play a principal role in trapping charges. The wide range of technological applications utilizing a -SiO₂ results in these trapped charges being responsible for the failure of an assortment of systems from electronic devices to communication technologies. Most prominently observed are threshold voltage shifts that lead to the inability to perform the switching operations of metal-oxide-semiconductor transistors and attenuation in optical fibers resulting in distance limitations on signal propagation. Understanding the physical nature of these charge traps at an atomic scale can provide insight toward charge trap specific processing techniques targeted at annealing the corresponding defects.

In addition, predicting defect levels through atomistic scale modeling techniques yields parameters that can be used in larger scale device level models, improving the accuracy of predicting long-term device performance. This multiscale approach can be extended to other material systems eventually culminating in the ability to predict new materials and better performing devices. Here, we use quantum mechanical methods to predict the energy levels and provide a physical explanation for defect states that lie within the a -SiO₂ band gap.

2. Method. A total of 27 stoichiometric and 42 oxygen deficient (single oxygen atom removed) a -SiO₂ structures containing respectively 192 and 191 atoms have been obtained from a previous study [3]. Each one of these structures contained a singly isolated defect or defect pair described in the proceeding section. Density functional theory (DFT) [6] within the generalized gradient approximation (GGA) [12] is performed on each structure for charge states spanning ($2\pm$) with the code SeqQuest [13]. Each charge state is computed via the local moment counter charge (LMCC) method [14] which eliminates the divergence created by the Coulomb potential when subject to periodic boundary conditions (PBCs)[15]. An electron reservoir (chemical potential) is set by putting the charged cell in contact with the neutral cell for each defect. The charge is located at each defect site to investigate the differences in the resulting electron densities after solving the Kohn-Sham (KS) self-consistent equations [9], resulting in a total of 675 DFT calculations. In addition the polarization energy which is added to a region outside of the supercell from each charge is accounted for through a model proposed by Jost [7] and refined by Schultz [16]. Polarization energies are calculated in both the static and infinite frequency dielectric constant limits. This method results in the total energies and relaxed geometries of five different charge states at each defect site, leading

*School of Materials Engineering, Purdue University, nlanders@purdue.edu

†Sandia National Laboratories, paschul@sandia.gov

‡School of Materials Engineering, Purdue University, strachan@purdue.edu

to the ability to calculate charge transition levels and correlate each level to a unique atomic relaxation.

3. Neutral Defects in α -SiO₂. Here we review the structural properties of the neutral α -SiO₂ point defects that are subject to this investigation. Defects in amorphous materials must be defined with reference to a defect free system. Defect free α -SiO₂ is defined as a continuous random network of SiO₄ tetrahedra connected via doubly coordinated bridging oxygen atoms. Hence, point defects in α -SiO₂ are prominently associated with a coordination deviation from the ideal four-fold Si and two-fold O. The defects we are studied here involve under-coordinated silicon (III-Si), overcoordinated silicon (V-Si), overcoordinated oxygen (III-O), and two-fold coordinated silicon (II-Si).

The neutral oxygen vacancy (NOV) is a defect that can basically be considered a Si–Si wrong bond within the amorphous network



where the “ \equiv ” denotes three network Si–O bonds. The dissociated three-coordinated silicon (D-III-Si) is a defect pair consisting of two isolated paramagnetic III-Si atoms



with the “ \bullet ” representing an unpaired electron, or a Si dangling bond. III-O/III-Si is a defect pair containing an isolated positively charged diamagnetic III-O and an isolated negatively charged diamagnetic III-Si



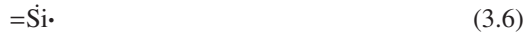
the “ \bullet ” denotes paired electrons and the positive charge corresponding to a hole delocalized over three Si–O bonds made explicit in the notation. Observed in systems with perfect stoichiometry, III-Si/V-Si is a defect pair that includes a positively charged diamagnetic III-Si and a negatively charged V-Si



with “ $=$ ” adopted for two Si–O bonds. Again seen in stoichiometric systems is a pair including an isolated positive diamagnetic III-O and an isolated negative V-Si



given the nomenclature III-O/V-Si and again the location of positive charge distinctly incorporated in the notation. The final defect investigated here is a isolated II-Si



where the “ \bullet ” on each side of the Si denotes unpaired electrons in separate sp^3 orbitals.

4. Charge Transition Levels. Self-consistently solving the KS equations results in the KS eigenvalues. Hence, in systems containing a forbidden electronic occupation region, DFT does not calculate the fundamental band gap but rather calculates a KS gap which is typically an underestimation of what is observed experimentally. This is the so called “band gap problem” associated with DFT. Schultz [16] has demonstrated, however, that by calculating the energy levels for a variety of defects in multiple charge states one can obtain a charge transition spectrum that spans the experimental band gap. The charge transitions are simply

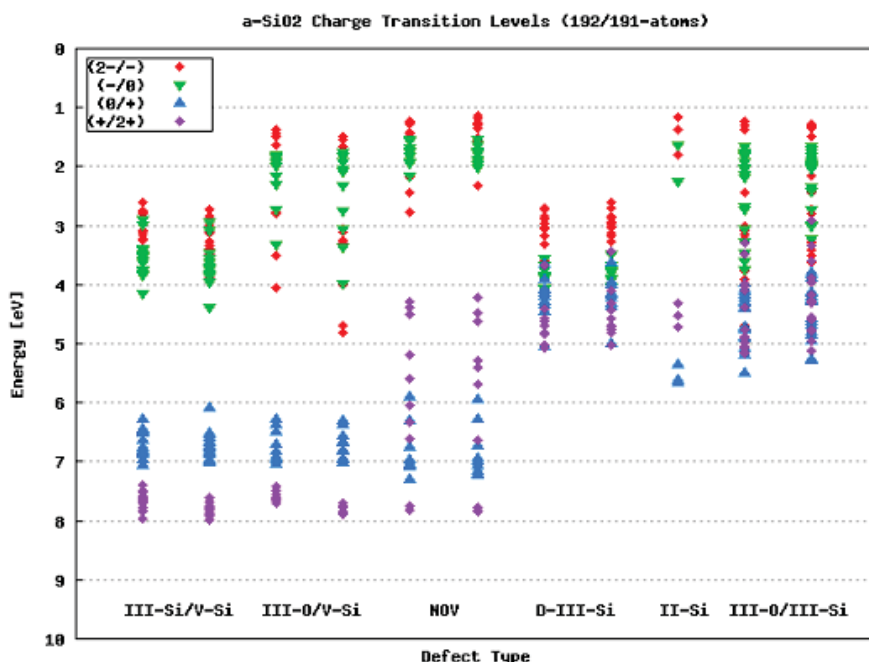


FIG. 4.1. Defect levels of charge transitions in *a*-SiO₂.

ionization potentials and electron affinities, the difference between the polarization corrected total energies of the two charge states. Each transition level is calculated for every defect site and the resulting spectrum in the static dielectric limit is shown in Fig. 4.1. Although this does not span the experimental *a*-SiO₂ band gap of ~ 9 eV, it does provide a rich amount of qualitative information about the physical nature of each defect level.

To determine the energy level of a defect within the silica band gap we must first define the valence band (VB) and conduction band (CB) edges. Our approach to this definition involves a combination of the KS eigenvalue occupations and local geometric relaxations.

For every defect site the KS eigenvalues are plotted for each charge state as a function of electron occupation near the KS gap, we will refer to this as a KS-spectrum. To facilitate comparative analysis each charge state KS-spectrum is aligned at the oxygen 2s level of the neutral calculation, which lies deep in the valence band preventing any edge distortions from biasing the alignment. One example of a KS-spectrum for each defect type is shown in Fig. 4.2(a) and all 135 KS-spectra used in our analysis are available in the Supplementary Material [2].

The local geometric structure of each defect site is then examined as a function of each charge state. An example of a geometry relaxation is shown in Fig. 4.2(b). Within the CB the accommodation of more electrons can be delocalized throughout the structure and virtually no local geometric relaxation will occur. Likewise, at the VB edge the removal of an electron (addition of a hole) can result in the partial occupation of several KS eigenstates due to the near degeneracy of their energies and thus the hole will also be delocalized, again resulting in no geometric relaxation. If the defect state lies within the gap however, the charge will be localized at the defect site which accommodates electrons (or holes), typically resulting in bonding rearrangement. This generally leads to shifts in the energy and also the appearance/disappearance of KS eigenvalues when comparing the different charge states, as

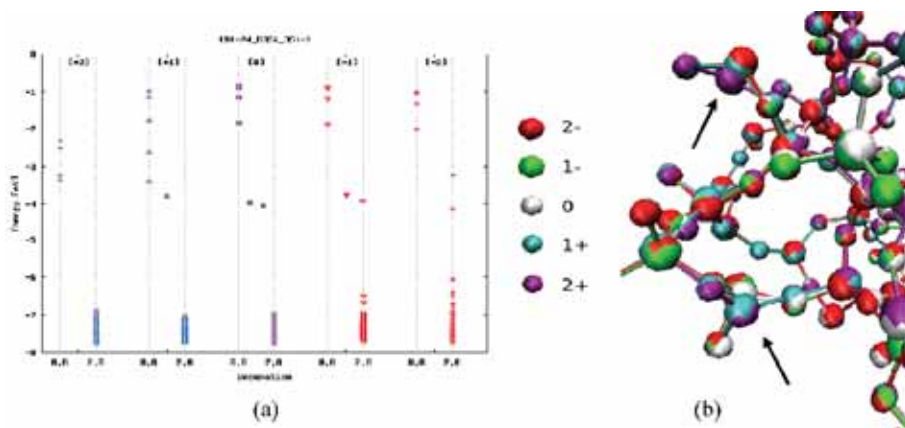


FIG. 4.2. (a) Sample KS-spectrum used for a D-III-Si defect and (b) corresponding geometry relaxation with each charge state labeled by appropriate color (arrows point to the III-Si involved).

each charge state corresponds to a different structure.

A parallel analysis of the KS-spectrum and the local geometric structure of each defect site is performed to facilitate a prediction of whether or not the defect state is contained within the band gap. Each defect charge transition is associated with a characteristic geometry relaxation. From these a geometry relaxation specific charge transition spectrum (analogous to Fig. 4.1) is generated shown in Fig. 4.3. The CB edge is determined by taking minimum energy of either the (1-/0) or (2-/1-) transition levels which corresponded to sites in which the KS eigenvalue that had zero, single, and double occupation for the respective (0), (1-), and (2-) states had the same energy and no geometric relaxation occurred. The VB edge is taken as the maximum energy of either the (0/1+) or (+/2+) transitions which corresponded to sites that led to partial occupations of the KS eigenstates for the (1+) and (2+) charge states, respectively, and that had no local geometric relaxation.

5. Geometric Relaxations. We now provide a physical interpretation of the characteristic geometry relaxations that occurred for the different charge transitions investigated. The percentage that each relaxation occurred for each defect and transition are shown as a pie charts in Figs. 5.1 - 5.6. All 675 of the final geometrically relaxed structures and 135 images of the charge state comparisons are available in the Supplementary Material [2]. The notation we adopt for each relaxation is that of DEFECT(Transition).number, e.g the third relaxation that occurs for the (0/1+) charge transition in a system containing an NOV would be NOV(0/1+).iii. All transitions that are theorized as VB or CB states are denoted *.o, the lower case zero used as the number because no localized geometry changes occurred. For conciseness within each defect sub-section the defect name is omitted.

NOV. The removal of a single electron results in two different geometry relaxations. Almost ubiquitously, an increase of the average Si-Si bond length of ~ 0.15 Å occurs [(0/1+).i]. Suggesting that a hole is captured by the bond and the remaining electron delocalizes over the dimerized Si sp^3 hybrid orbitals,



lending credence to an assumption of a remaining weaker bond. Secondly, (0/1+).ii relaxations occur in systems in which the missing oxygen is essentially part of an edge sharing tetrahedron (EST), a two member ring, and the neutral Si-Si bond distance is ~ 2.2 Å. We

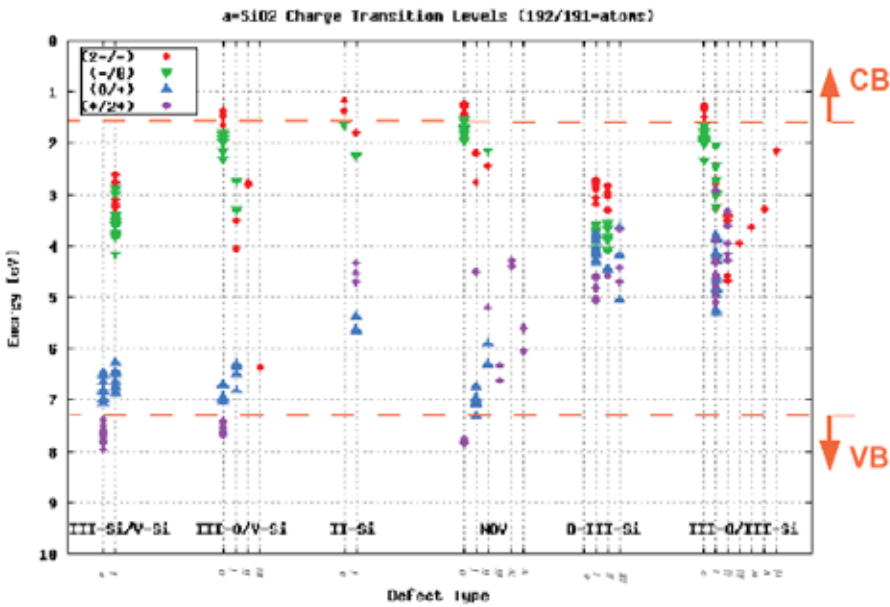
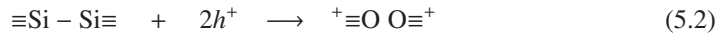


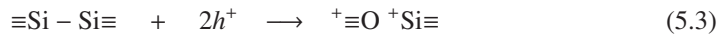
FIG. 4.3. Defect levels for each charge transition sorted by specific corresponding geometric relaxation.

shall refer to the systems containing this defect as EST-NOV and it should be regarded as separate from the NOV as characteristic relaxations undergone are exclusive to this defect. The (0/1+).ii involves the weakening of the Si-Si bond through a distance increase of ~ 0.4 Å and the increase of the Si-O-Si bond angle by $\sim 20^\circ$, thus this transition would also follow 5.1.

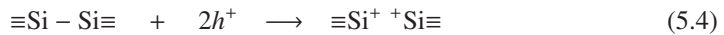
Absence of two electrons led to six unique geometry relaxations. (1+/2+).o involves no localized geometry change and partially occupied KS eigenvalues, again suggesting VB states. Both NOV silicon atoms puckering through the plane of their nearest neighbor oxygen atoms and back-bonding to another network oxygen constitute the (1+/2+).i relaxation.



This results in two positively charged III-O centers. Only one of the silicons becomes puckered and the second silicon becoming planar with its oxygen neighbors



is (1+/2+).ii. The three planar O atoms form O-Si-O bond angles of $\sim 120^\circ$ suggesting the Si atom involved becomes sp^2 hybridized and a hole is trapped on the remaining Si 3*p* orbital which lies perpendicular to the O containing plane. (1+/2+).iii involves both silicon atoms trapping a hole and becoming planar.



A dramatic local network rearrangement in which two nearby oxygen atom become over-coordinated and form a two member ring containing the two Si from the NOV and the two new III-O is the (1+/2+).iv relaxation. This relaxation would take on the same form as 5.2. Finally, in the EST-NOV systems, (1+/2+).v entailed further increases the Si-O-Si bond angle to $\sim 130^\circ$, a value within the experimentally observed range of Si-O-Si bond angles in

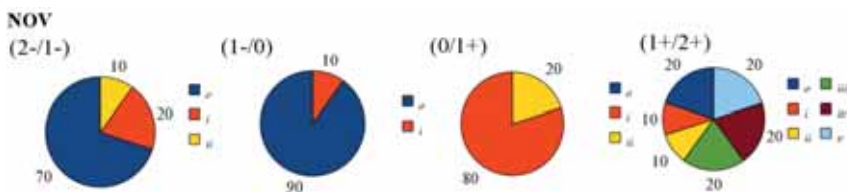


Fig. 5.1. Geometric relaxation occurrence percentages for the NOV.

α -SiO₂, and the silicon atoms moving into the plane of their three neighboring oxygen atoms, fully breaking the Si-Si bond. This leaves two positively charged III-Si who share a bridging oxygen atom and would be chemically equivalent to 5.4.

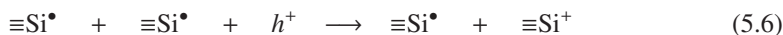
Addition of a single electron only led to two relaxations. Most prominently, no localized geometry change and no shift in the energy of the occupation effected eigenvalue of the aligned KS-spectrum constitutes (1-/0).o, indicating a delocalized CB state. And (1-/0).i, seen only in the EST-NOV systems, the Si-O-Si angle shifts to 85° and the Si-Si bond distance increases to 2.4 Å, indicating a weakening but no change chemical bonding.

Three relaxations were observed upon doubly adding electrons. Frequently, (2-/1-).o, no local geometry or KS eigenvalue energy shift. Very interestingly, (2-/1-).i involves a non-nearest neighbor bridging oxygen atom in the network breaking one of its Si bonds and becoming a singly coordinated non-bridging oxygen (NBO) and leaving behind a triply coordinated silicon.



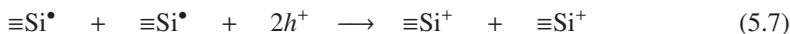
This new III-Si has an average bond angle of ~104°, suggesting that it has paired electrons localized on its non-bonding sp^3 orbital and the NBO takes on the second electron to complete the filling of its singly occupied $2p$ orbital. Lastly, again only seen in EST-NOV systems, (2-/1-).ii is a rupture of the Si-Si bond as they further separate by 0.5 Å and the Si-O-Si angle moves to 113°. Accompanying (2-/1-).ii, the retreat from the initial oxygen vacant site results in one of the silicons forming a new Si-Si bond with its second nearest neighbor Si, forming a new EST-NOV with a Si-Si bond length of 2.1 Å and Si-O-Si angle of 75°.

D-III-Si. In the neutral state the KS-spectrum is characterized by two partially occupied mid-gap states, presumed to be the two silicon dangling bonds. All of the charge transitions associated with the D-III-Si are theorized to lie within the band gap. There are three observed different geometry relaxations observed to occur for the removal of a single electron. Most commonly, (0/1+).i, upon the removal of one electron one of the III-Si moves into the plane of its three oxygen neighbors.



Here, as with NOV(1+/2+).ii and NOV(1+/2+).iii, the O-Si-O bond angles around the newly unpuckered III-Si are all ~120°, again implying that the bonds become sp^2 hybridized leaving a hole localized on a $3p$ orbital perpendicular to the Si≡O₃ plane. (0/1+).ii is the partial movement of each of the two III-Si toward the O containing plane. A network rearrangement to allow an oxygen to move closer to one of the III-Si comprises (0/1+).iii.

Upon the removal of the second electron one of three geometry relaxations occur. Primarily, (1+/2+).i, the III-Si not involved in a (0/1+).i relaxation it replicates the performance of the first III-Si, resulting in two isolated $3p$ orbital trapped holes.



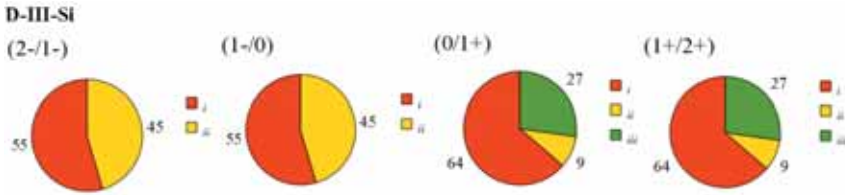
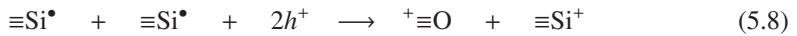


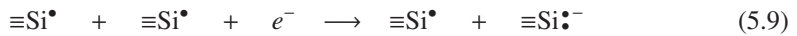
FIG. 5.2. Geometric relaxation occurrence percentages for the D-III-Si.

A second relaxation, (1+/2+).ii, occurs when both partially planar III-Si involved in (0/1+).ii become planar with $\sim 120^\circ$ O-Si-O bond angles. Again leaving two isolated III-Si with $3p$ orbital trapped holes, and is chemically equivalent to 5.7. Another third relaxation, (1+/2+).iii, involves the III-Si nearest the oxygen involved in (0/1+).iii becomes tetrahedrally coordinated forming a positively charged III-O center.



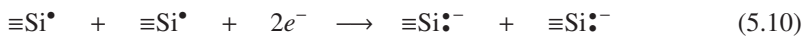
Also, (1+/2+).iii consists of the III-Si that doesn't undergo a coordination change to becoming planar with its three oxygen neighbors in analogous fashion as (0/1+).i, resulting in a $3p$ localized hole.

Adding an electron to a neutral system yielded two relaxations which occur with similar frequencies. The (1-/0).i, in which the average O-Si-O bond angle on both III-Si decreases from $\sim 107^\circ$ to $\sim 105^\circ$, which suggests that the sp^3 hybridization slightly weakens and the extra electron is delocalized over both dangling bonds. Or (1-/0).ii, just one of the III-Si atoms tightens its average O-Si-O bond angle from $\sim 107^\circ$ to $\sim 103^\circ$ implying the sp^3 hybridization weakens,



leading to a doubly occupied orbital and the extra negative charge density results in Coulomb repulsion of the electrons in the $\text{Si}\equiv\text{O}_3$ bonds.

Upon adding a second electron, two relaxations were found to take place, again with similar frequencies. The (2-/1-).i, where both the III-Si involved in (1-/0).i tighten their O-Si-O angle to $\sim 102^\circ$, both become diamagnetic with paired sp^3 electrons localized on its non-bonding orbital.



Or (2-/1-).ii, where the III-Si not involved in (1-/0).ii undergoes consonant relaxation, leaving a final geometry which is indistinguishable from that which follows (2-/1-).i.

III-O/III-Si. The neutral KS-spectrum for this defect is characterized by a doubly occupied mid gap state. The III-Si involved in this defect pair had an average O-Si-O bond angle of $\sim 103^\circ$, which leads us to hypothesize that this III-Si has a doubly occupied sp^3 orbital where the Coulomb repulsion bends the other Si-O bonds away. This extra electron likely came from the III-O center which loses an electron to become positively charged. Thus, the III-Si is able to stabilize the III-O by taking on an additional electron, and the pair of electrons localized on the dangling bond of this III-Si corresponds to the observed neutral mid gap state.

One characteristic relaxation occurs upon the single removal of an electron. The III-Si average O-Si-O bond angle increases to $\sim 109^\circ$ [(0/+).i], which implies a neutral paramagnetic

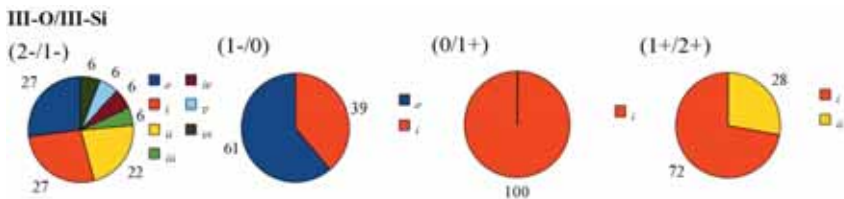


FIG. 5.3. Geometric relaxation occurrence percentages for the III-O/III-Si.

dangling bond consisting of a single an unpaired electron.

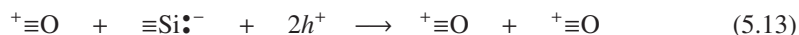


This is further supported by the (1+) KS-spectrum which contains a singly occupied mid gap state.

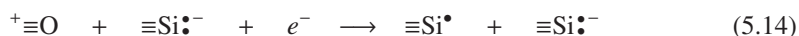
Removal of a second electron leads to two different relaxations. The III-Si moving into the plane of its O neighbors with a O-Si-O angle of $\sim 120^\circ$ [(+/2+).i]. As mentioned in above, we propose this is a sp^2 hybridized III-Si with a $3p$ trapped hole.



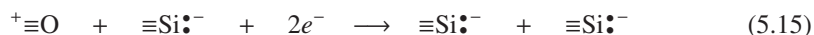
A second relaxation, (+/2+).ii, involves a local network rearrangement in which a oxygen in the the vicinity of the III-Si overcoordinates with the III-Si, resulting in a second positively charged III-O center.



Adding an electron results in two relaxations. The first, (1-/0).o, is no change in local geometry or energy of the occupation effected KS eigenvalue. The second, (1-/0).i, involves the breaking one of the Si-O bonds involved in the III-O leaving a second III-Si with $\sim 106^\circ$ average O-Si-O bond angle, pointing to an unpaired electron on a sp^3 orbital.



A second electron addition leads to seven different relaxations. Firstly, (2-/1-).o, a relaxation identical to (1-/0).o. Secondly, the III-Si involved in (1-/0).i contracts its average O-Si-O bond angle to $\sim 99^\circ$ [(2-/1-).i]. This relaxation, just as seen in other relaxations, implies that the sp^3 hybridization is very small and the two electrons occupy a filled approximate $3s$ orbital with the Si-O bonds formed on the three $3p$ orbitals.

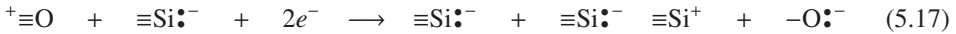


A third relaxation, (2-/1-).ii is analogous to (1-/0).i. Another relaxation, (2-/1-).iii, involves a bond breaking between an O nearest neighbor of a properly coordinated Si atom and forming a new bond with a different fully coordinated network Si. The resulting atomic geometry then contains a two III-Si, one V-Si, and one III-O.



For both III-Si the average O-Si-O bond angle is $\sim 103^\circ$, thus we presume paired electrons to be localized on an sp^3 orbital of both of these. We also predict that the III-O has given up

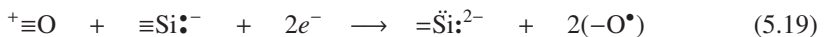
an electron which localizes near the V-Si, stabilizing this defect pair and resulting in the net system charge of (2-). A relaxation [(2-/1-).iv] in which the III-Si involved in (1-/0).i becomes roughly planar with an average O-Si-O bond angle of $\sim 116^\circ$ (it becomes sp^2 hybridized and traps a hole on its 3*p* orbital).



This relaxation also involves a nearby fully coordinated silicon breaking a Si-O bond and becoming undercoordinated with an average O-Si-O bond angle of $\sim 95^\circ$, which suggests it is not hybridized with a filled 3*s* orbital and forms its three O bonds with its singly occupied 3*p* orbitals. In addition to these unique rearrangements, (2-/1-).iv also involves a nearby bridging oxygen breaking one of its bonds, leaving a negatively charged NBO with a pair of electrons localized on its 2*p* orbital. Yet another relaxation, (2-/1-).v, involves a tetrahedrally bonded silicon near the III-O breaking one of its Si-O bonds and becoming planar with its three remaining oxygen neighbors with an average O-Si-O bond angle of $\sim 120^\circ$ (again a sp^2 III-Si with a 3*p* trapped hole).



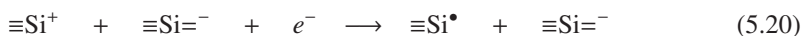
The O involved in the broken Si-O bond becomes a singly coordinated NBO which has trapped an electron. Another bonding change which occurs in (2-/1-).v involves another nearby fully coordinated silicon with an average Si-O bond distance of 1.7 Å, three of its O-Si-O bond angles are $\sim 99^\circ$, and its fourth bond angle is an obtuse $\sim 130^\circ$. The relaxation that occurs involves widening the obtuse angle to $\sim 165^\circ$ and stretching the Si-O bonds which are a part of that angle to ~ 1.9 Å. The other two bonds also stretch to ~ 1.8 Å but their O-Si-O angles undergo negligible changes. We interpret this as a II-Si with very little hybridization containing a filled 3*s* orbital and weak 3*p* bonds with the two nearer oxygens, it is here that the (-2) charge is localized. The remaining two oxygen atoms can each be thought of as an NBO with an unpaired electron, this is the well known and experimentally characterized electrically neutral paramagnetic non-bridging oxygen hole center (NBOHC) [5]. Finally, (2-/1-).vi, is a relaxation synonymous with (2-/1-).v except the positively charged sp^2 III-Si and negatively charged NBO relaxations do not occur (i.e. the doubly negative II-Si with two nearby NBOHC's is the only relaxation which occurs).



III-Si/V-Si. Analysis of the neutral charge state of this defect showed that all the III-Si were in a planar arrangement with their three neighboring oxygens with $\sim 120^\circ$ O-Si-O bond angles. As aforementioned, this is the same resulting positively charged diamagnetic III-Si defect that is observed in several oxygen deficient samples after the removal of an electron. This is thought to be sp^2 hybridized with a 3*p* localized hole, hence the electron localized somewhere else in the structure. We predict that the electron localization takes place near the V-Si allowing for the extra Si-O bond, thus the positively charged III-Si stabilizes the V-Si.

Upon the removal of an electron two relaxations occurred. Either no geometry change and many partially occupied KS eigenvalues [(0/1+).o], or movement of one or multiple V-Si nearest neighbor oxygen atoms [(0/1+).i] (supporting the hypothesis of the missing III-Si electron localizing at the V-Si). A removal of another electron results in no geometry change and many partial KS eigenvalue occupations, likely a VB transition.

Adding an electron results in the planar III-Si moving through the O plane and becoming roughly tetrahedral with an average O-Si-O bond angle of $\sim 107^\circ$ [(1-/0).i]. Implying sp^3 hybridization with an unpaired electron localized on the non-bonding orbital.



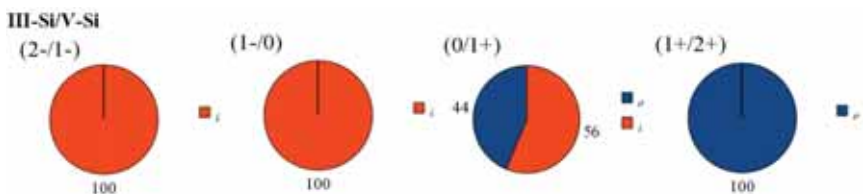


Fig. 5.4. Geometric relaxation occurrence percentages for the III-Si/V-Si.

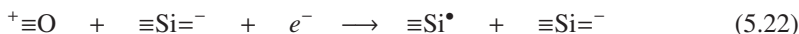
Addition of a second electron leads to an O-Si-O angle of $\sim 102^\circ$ [(2-/1-).i], suggesting paired electrons on the non-bonding hybrid orbital with Coulomb repulsion bending the other three bonds away.



The fact that only one characteristic relaxation occurs upon the single and double addition of electrons can be attributed to the relatively tight charge transition bands seen in Figs. 4.1 and 4.3.

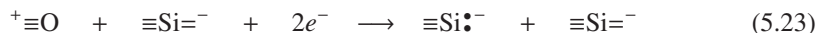
III-O/V-Si. In a manner similar to the positively charged planar III-Si stabilizing the V-Si via electron donation in the III-Si/V-Si, we hypothesize that the III-O has given up an electron which again stabilizes the V-Si. The fact that analogous relaxations as those seen in III-Si/V-Si occur for the removal of one and two electrons supports this hypothesis. Single electron removal either leads to no geometry change and partial KS eigenvalue population [(0/1+).o] or movement of at least one of the V-Si oxygen neighbors [(0/1+).i]. Double electron removal leads again to no geometry change and considerable number of partial KS eigenvalue occupations [(1+/2+).o].

Addition of an electron causes no change in geometry or occupation effected KS eigenvalue [(1-/0).o], or breaks one of the III-O bonds causing a Si to retreat away becoming undercoordinated [(1-/0).i].



This new III-Si is nearly tetrahedral with an average O-Si-O bond angle of $\sim 107^\circ$, again sp^3 hybridized with an unpaired electron on the empty orbital.

A second electron induces four relaxations. No structural change or occupation effected KS eigenvalue energy shift [(2-/1-).o]. For systems which underwent (1-/0).o a relaxation parallel to (1-/0).i occurs [(2-/1-).i], indicative of a negative- U behavior. As for the systems that were involved in (1-/0).i, the average O-Si-O angle for the effected III-Si further decreases to $\sim 102^\circ$ [(2-/1-).ii],



the second electron pairs on the singly occupied orbital and electrostatics repel the remaining three bonds. Finally, (2-/1-).iii, consists of the same re-bonding as (2-/1-).i however it is accompanied by a significant rearrangement to the local network around the new III-Si.

It is important to note here that the final geometries after the occurrence of the III-Si/V-Si(1-/0).i and III-O/V-Si(1-/0).i are virtually indistinguishable from one another, a single unpaired electron on a III-Si sp^3 orbital and a V-Si. Likewise the final geometries from the III-Si/V-Si(2-/1-).i and III-O/V-Si(2-/1-).ii are also equivalent, a doubly occupied III-Si sp^3 orbital and a V-Si. This is consistent with the observation that removal of an unpaired sp^3

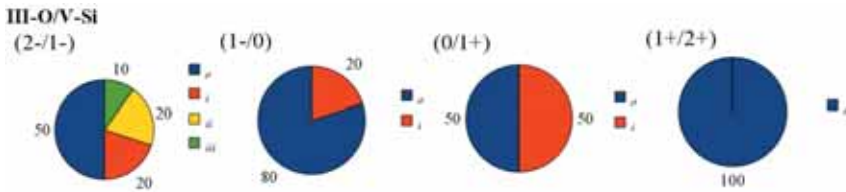


FIG. 5.5. Geometric relaxation occurrence percentages for the III-O/V-Si.

electron from a III-Si in the oxygen deficient cells yielded two results with very high frequency: a planar sp^2 III-Si with a $3p$ trapped hole (identical to the III-Si in the III-Si/V-Si systems) or an positively charged III-O center (akin to the III-O in the III-O/V-Si systems).

II-Si. A fully occupied mid-gap state exists for the neutral species, likely two electrons that are each localized on separate sp^3 orbitals. This conclusion is drawn from the average O-Si-O bond angle of 110° . Upon the removal of one electron the local network undergoes a rearrangement [(0/1+).i] in such a matter that the II-Si and a neighboring oxygen become coordinated,



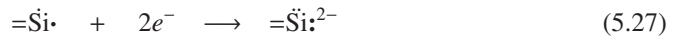
leading to a III-Si and a positively charged III-O. Removal of a second electron furthers network rearrangement [(1+/2+).i] inhibiting a new bond between the III-Si and another network oxygen, resulting in a fully coordinated silicon and two positively charged III-O centers.



Addition of electrons leads to two relaxations for each transition. For the single electron addition, (1-/0).o has no local geometry change and no occupation effected KS eigenvalue energy shift, assumed to be a delocalized CB state. The other relaxation, (1-/0).i, the addition of a single electron leads to a O-Si-O bond angle of 105° implying that the electron pairs with one of the sp^3 electrons and the Coulomb repulsion bends the Si=O₂ bonds away from the doubly occupied orbital.



Double electron addition again either has no effect on the geometry and the occupation effected KS eigenvalue [(2-/1-).o], or the second electron causes the bond angle to dramatically shift to 96° [(2-/1-).i] which suggests that the Si hybridization disappears and the electron configuration consists of a filled $3s$ orbital and singly occupied $3p$ orbitals.



In (2-/1-).i, Si=O₂ bonds are formed with two of these $3p$ orbitals and paired electrons are left on the remaining orbital with Coulomb repulsion bending the Si=O₂ bonds perpendicular to the idealized O-Si-O plane causing the angle to be slightly larger than 90° .

6. Discussion. In this section we separately discuss the implications of the statistically prevalent seen relaxations which were observed as they apply to each isolated defect.

III-Si. Perhaps the most significant result seen here involves the III-Si. This defect is seen to occur isolated and in three different charge states: a neutral paramagnetic state in which an unpaired electron is localized on a sp^3 hybrid orbital ($\equiv\text{Si}\cdot$), a diamagnetic (+1)

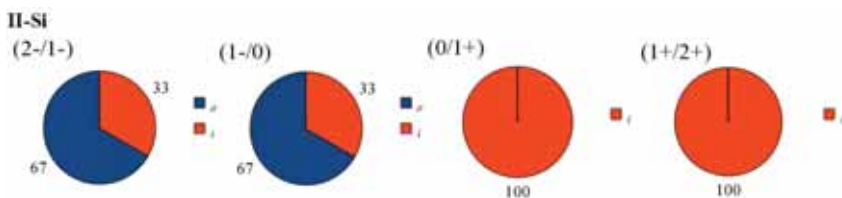


FIG. 5.6. Geometric relaxation occurrence percentages for the II-Si.

state in which the the III-Si is sp^2 hybridized and planar with its three O neighbors with a hole localized on its remaining $3p$ orbital ($\equiv\text{Si}^+$), or a diamagnetic (-1) state in which the non-bonding sp^3 orbital is doubly occupied by a pair of electrons ($\equiv\text{Si}^{\bullet-}$). Thus it is the neutral state which is EPR susceptible. These neutral isolated $\equiv\text{Si}^{\bullet}$ fragments can be viewed as the commonly proposed NOV derived E'_γ center [19] in the absence of the puckered III-Si. Some authors believe that these fragments correspond to the E'_β center [1], but there exist other suggestions that the E'_β involves a nearby hydrogen atom at a $\text{H-Si}\equiv$ center (the additional proton shifts the hyperfine splitting of the EPR signal) [5]. Depending on the distance of this $\text{H-Si}\equiv$ unit from the $\equiv\text{Si}^{\bullet}$ this hyperfine shift could be very small and essentially negligible in experiment, therefore we conclude that the neutral isolated $\equiv\text{Si}^{\bullet}$ fragments observed in this study can be thought of as either E'_γ or E'_β centers. Calculation of the spectroscopic and hyperfine splitting factors for each of the $\equiv\text{Si}^{\bullet}$ defects found could offer distinction, but presently we emphasize the claim of neutral isolated E' defects, regardless of the presence of an oxygen vacancy [3], is firmly supported by this work.

Additionally, removing or adding an electron to the $\equiv\text{Si}^{\bullet}$ yields respective $\equiv\text{Si}^+$ or $\equiv\text{Si}^{\bullet-}$ fragments. Likewise starting with $\equiv\text{Si}^{\bullet-}$ and doubly removing electrons leads to $\equiv\text{Si}^+$, and starting with $\equiv\text{Si}^+$ and doubly adding electrons leads to $\equiv\text{Si}^{\bullet-}$. This provides an indication of a hysteresis among E'_γ and E'_β centers in which they can switch between being hole, electron/hole, and electron capture centers. This reversible behavior of E' centers has been demonstrated experimentally [19], however the atomic relaxations seen here contradict previous proposals which account for this behavior that require the presence of an NOV.

NOV. Considering the pure NOV defects (not the small percentage of EST-NOV defects) an addition of a single electron universally results in an increase in the Si-Si dimer bond length of $\sim 0.1\text{-}0.2$ Å. This implies a $\equiv\text{Si}^+ \cdot \text{Si}\equiv$ center is created through the capture of a hole by the dimer bond. The weaker bond is likely due to the delocalization of the unpaired electron over the remaining pair of sp^3 hybrid orbitals. Previous theoretical studies have reported this structure as having EPR signals in good experimental agreement with the E'_δ center [4, 18]. Hence, our results further the support of the creation of positively charged paramagnetic E'_δ centers as a result of NOV hole trapping.

III-O. We find that this defect exists in a positively charged diamagnetic state ($^+\equiv\text{O}$). The neighboring silicon atoms lie essentially in the plane of the overcoordinated oxygen with Si-O-Si angles of $\sim 120^\circ$, suggesting that the O is sp^2 hybridized with a doubly occupied $2p$ orbital and forming the three bonds in each of the sp^2 orbitals. The Si-O bonds are weaker in comparison to those involved in two-fold bridging oxygen with lengths roughly 0.2 Å longer, we therefore hypothesize that the hole is delocalized over the three nearly D_{3h} symmetric hybrid orbitals, accounting for the reason that the positive charge is located on the three silicons in our adopted notation of $^+\equiv\text{O}$ for this defect. This defect is found to be an electron capture center, upon which one of the three Si-O bonds breaks leaving behind an isolated neutral paramagnetic III-Si (i.e. an E'_γ or E'_β center). The reverse reaction is also

seen to take place, an E'_γ or E'_β center that is nearby a network oxygen will capture a hole and form $^+\equiv\text{O}$ center with its proximal network O neighbor. This relaxation can be interpreted as nearly analogous to the proposed NOV capture of a hole and one of the silicon atoms becoming puckered by back-bonding to another network oxygen, the originally accepted E'_γ center formation mechanism [17]. However, here no oxygen vacancy is necessary and the puckered silicon precursor is an isolated $\equiv\text{Si}^\bullet$ fragment. We note that the formation of $^+\equiv\text{O}$ centers does occur upon removing two electrons from a NOV defect. As stated above the removal of one electron from the NOV results in a positively charged E'_δ center. Removal of a second electron leads to either two $^+\equiv\text{O}$ centers or a $^+\equiv\text{O}/\equiv\text{Si}^+$ pair, both with negative- U behavior. In a fashion similar to the III-Si, this implies a hysteric behavior in which $^+\equiv\text{O}$ centers are converted into $\equiv\text{Si}^\bullet$ and vice-versa through the respective capture of an electron and a hole.

V-Si. This defect is a negatively charged diamagnetic center ($\equiv\text{Si}^-$). An extra electron allows for the creation of the fifth Si-O bond. This additional bond is known as a floating bond and has been hypothesized to exist both theoretically and experimentally in amorphous silicon (*a*-Si) [11, 8], and tentatively identified experimentally in $\equiv\text{Si}^-$ [10]. We find that these $\equiv\text{Si}^-$ can act as hole traps resulting in a neutral defect centers with weaker Si-O bands, likely indicative that the hole is delocalized over the five bonds. Taking into account the relatively low formation energy of this defect [3], we conclude that this defect is a positive charge trap that has not previously been well characterized.

II-Si. We find that this defect exists in three different charge states: neutral, negative, and doubly negative. The (0) state has an unpaired electron in each of its two non-bonding sp^3 orbitals. Upon electron capture the (-1) state involves the an unpaired electron on one of these orbitals and a pair of electrons on the other orbital. A capture of a second electron leads to a (-2) state and the disappearance of hybridization with a filled $3s$ orbital, a pair of electrons on one $3p$ orbital, and the Si-O bonds formed with the remaining two orthogonal $3p$ -orbitals. This indicates that the II-Si is an electron trapping center, however the small amount of these defects studies does not lend statistical support to this hypothesis. We also find this defect is involved with trapping positive charge as one and two $^+\equiv\text{O}$ centers result from a local network rearrangement involving nearby oxygen atoms upon the respective capture of one and two holes.

7. Conclusion. We have used density functional techniques which properly account for the electrostatics in the presence of PBCs to study an array of charge states for numerous isolated defect sites *a*-SiO₂. In addition to identifying the charge transition levels of various charge traps within *a*-SiO₂, specific atomic geometry relaxations corresponding to each level have been reported. We found that isolated paramagnetic E'_γ and E'_β defects occur in the neutral charge state and are capable of trapping both electrons and holes. Statistical support for the NOV originated dimerized model of the E'_δ defect has also been demonstrated. As the result of hole trapping by an undercoordinated silicon we showed a significant number of stable $^+\equiv\text{O}$ defects result and are likely a prevalent source of trapped positive charge. Finally, support for a not well established hole trap associated with the an overcoordinated silicon floating bond defect has been presented.

8. Acknowledgments. The authors thank Harold P. Hjalmanson, Arthur H. Edwards and Richard P. Muller for useful discussions. This work was partly supported by Purdue's Center for the Prediction of Reliability, Integrity and Survivability of Microsystems (PRISM) funded by the US Department of Energys National Nuclear Security Administration under contract Award No. DE-FC52-08NA28617. Sandia National Laboratories is a multi-program laboratory operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin company, for the U.S. Department of Energys National Nuclear Security Administration

under contract DE-AC04-94AL85000.

REFERENCES

- [1] V. V. AFANAS'EV AND A. STESMANS, *Charge state of paramagnetic E' centre in thermal SiO₂ layers on silicon*, J. Phys.: Condens. Matter, 12 (2000), p. 2285.
- [2] N. L. ANDERSON, *Contact author for Supplementary Material*, nlanders@purdue.edu.
- [3] N. L. ANDERSON, R. P. VEDULA, P. A. SCHULTZ, R. M. VAN GINHOVEN, AND A. STRACHAN, *First-Principles Investigation of Low Energy E' Center Precursors in Amorphous Silica*, Phys. Rev. Lett., (Submitted August 2010).
- [4] J. R. CHAVEZ, S. P. KARNA, K. VANHUESDEN, C. P. BROTHERS, R. D. PUGH, B. K. SINGARAJU, W. L. WARREN, AND R. A. B. DEVINE, *Microscopic structure of the E'_δ center in amorphous SiO₂: A first principles quantum mechanical investigation*, IEEE Trans. Nucl. Sci., 44 (1997), p. 1799.
- [5] D. L. GRISCOM, *The nature of point defects in amorphous silicon dioxide*, Defects in SiO₂ and Related Dielectrics: Science and Technology, ed G. Pacchioni, L. Skuja, and D.L. Griscom, Kluwer, Dordrecht (2000), p. 122.
- [6] P. HOHENBERG AND W. KOHN, *Inhomogeneous electron gas*, Phys. Rev. B, 136 (1964), p. B864.
- [7] W. JOST, *Diffusion and Electrolytic Conduction in Crystals (Ionic Semiconductors)*, J. Chem. Phys., 1 (1933), p. 466.
- [8] S. KNIEF AND W. VON NIESSEN, *Electron spin resonance signals in a-Si: Dangling versus floating bonds*, Phys. Rev. B, 60 (1999), p. 5412.
- [9] W. KOHN AND L. J. SHAM, *Self-Consistent Equations Including Exchange and Correlation Effects*, Phys. Rev., 140 (1965), p. 1133.
- [10] J. MACHACEK, O. GEDEON, AND M. LISKA, *Molecular approach to the 5-coordinated silicon atoms in silicate glasses*, Phys. Chem. of Glasses, 48 (2007), p. 345.
- [11] S. T. PANTELIDES, *Defects in Amorphous Silicon: A New Perspective*, Phys. Rev. Lett., 57 (1986), p. 2979.
- [12] J. P. PERDEW, K. BURKE, AND M. ERNZERHOF, *Generalized Gradient Approximation Made Simple*, Phys. Rev. Lett., 77 (1996), p. 3865.
- [13] P. A. SCHULTZ, *Unpublished: see*, <http://dft.sandia.gov/Quest>.
- [14] ———, *Local electrostatic moments and periodic boundary Conditions*, Phys. Rev. B, 60 (1999), p. 1551.
- [15] ———, *Charged Local Defects in Extended Systems*, Phys. Rev. Lett., 84 (2000), p. 1942.
- [16] ———, *Theory of Defect Levels and the "Band Gap Problem" in Silicon*, Phys. Rev. Lett., 96 (2006), p. 246401.
- [17] P. V. SUSHKO, S. MUKHOPADHYAY, A. S. MYSOVSKY, V. B. SULIMOV, A. TAGA, AND A. L. SHLUGER, *Structure and properties of defect in amorphous silica: new insights from embedded cluster calculations*, J. Phys.: Condens. Matter, 17 (2005), p. S2115.
- [18] B. R. TUTTLE AND S. T. PANTELIDES, *Vacancy-related defects and the E'_δ center in amorphous silicon dioxide: Density functional calculations*, Phys. Rev. B, 79 (2009), p. 115206.
- [19] H. S. WITHAM AND P. M. LENAHA, *Nature of the E' deep hole trap in metal-oxide-semiconductor oxides*, Appl. Phys. Lett., 51 (1987), p. 1007.

WHEN NANO-PARTICLES COLLIDE VIA MOLECULAR DYNAMICS

YOICHI TAKATO*, JEREMY B. LECHMAN†, AND SURAJIT SEN‡

Abstract. We perform nanoparticle collision simulations using molecular dynamics, and calculate the coefficient of restitution for two identical nanoparticles made of an fcc lattice of Lennard-Jones atoms whose size ranges from 603 to 44,403 atoms per particle. Varying the collision velocity we find distinct regimes of particle behavior — elastic and plastic deformations. In the plastic deformation region there appear to be three distinct deformation modes; dislocations of atoms in a particle, large permanent deformations of a particle, and phase transition effects. We make preliminary comparisons of nanoparticle behavior to that of macroscopic particles.

1. Introduction. The collision of particles is a fundamental process that occurs in many fields of applied science: aerosol aggregation and collection, formation of planetary systems, powder and ceramics processing, initiation of energetic materials and blast mitigation. It is a simple process that is often used in undergraduate physics classes to introduce the concepts on momentum and energy balance in the context of classical mechanics. It is in this context that many have been introduced to the “coefficient of restitution” which is simply the ratio of the relative velocities of the two particles before and after the collision. The coefficient of restitution is practically a convenient physical quantity because without knowing any microscopic details about the collision process, it is possible to determine how much energy is dissipated or converted to other forms during the collision. If the relative velocity of the particles is the same after the collision, the collision is said to be elastic and the coefficient of restitution is unity. If the relative velocity is different, the coefficient of restitution becomes less than one. Therefore, many experiments involving collisions of macroscopic objects have been performed, and their results have been analyzed in terms of the coefficient of restitution.

Many theoretical studies of quasi-static contact between two macroscopic objects have been carried out. Hertz [1] derived a nonlinear contact force between elastic spheres, namely, Hertzian contact theory. For particles colliding under the influence of the quasi-static force, the coefficient of restitution is unity; no energy lost from the center-of-mass translational degrees of freedom during the collision. In the case of plastic deformation, the coefficient of restitution is less than unity. Johnson [2] has shown that the relation between the coefficient of restitution and collision impact velocity can be described by a power law; the coefficient of restitution decreases with increasing collision velocity beyond the collision velocity at which plastic yield begins.

However, from the microscopic point of view, the collision process is still poorly understood. In particular the dissipation mechanism associated with plastic collisions (coefficient of restitution less than one) are poorly understood. The kinetic energy loss (from macroscopic, center-of-mass point of view) can be distributed into internal energy (e.g. heat loss and/or a change in the total potential energy) or kinetic energy other than center-of-mass translational kinetic energy (e.g. rotational or vibrational kinetic energy of particles). To our knowledge, no studies that analyze the energy transformation and dissipations mechanisms and relate them to the coefficient of restitution over a wide range of collision velocities for a simple atomistic system have been performed.

In principle, the molecular dynamics (MD) simulation can calculate any instantaneous microscopic physical quantities such as position and velocity of each atom, energy and force associated with atom’s motion and position. The collision process takes place in a very short

*The State University of New York at Buffalo, ytakato@buffalo.edu

†Sandia National Laboratories, jblechm@sandia.gov

‡The State University of New York at Buffalo, sen@buffalo.edu

time, and hence, it is quite difficult to track it by an experiment. In contrast, MD can easily track any moment of collisions and can recover macroscopic phenomena such as deformations, vibrations of particles from microscopic quantities. It can show the coefficient of restitution as well, and can even connect instantaneous phenomena and coefficient of restitution. Unfortunately, due to computational limitations, MD can calculate particles of atoms far smaller than macroscopic particles within a finite computation time.

While some nanoparticle collision simulations have been studied by other researchers utilizing MD, they are limited to the elastic collision regime or very high collision velocity regime and do not analyze the details of the energy balance. Most studies using MD have dealt with collisions of a particle onto a flat surface, while a few studies have reported particle-particle collisions. For the latter case, Kalweit *et al.* [4, 3] have reported dynamics of collisions of large particles up to 10,973 atoms at high collision velocities so that particles coalesce or break into fragments. At the other extreme, Kuninaka *et al.* [5, 6] have shown that a nanoparticle's coefficient of restitution for 682 Lennard-Jones atoms in a particle can exceed unity at very low impact velocities.

In this study, we utilize LAMMPS, an MD code developed by Sandia National Laboratories, to simulate collisions of two identical spherical nanoparticles. The largest particle we report results for contains 44,403 atoms. We present coefficient of restitution of the particles over a wide range of impact velocities so that the particles' behavior ranges from elastic deformation to plastic deformation. We compare the calculated coefficient of restitution with that derived by Johnson's macroscopic plastic deformation theory. We also show energy balance before and after collision and relate it to the coefficient of restitution.

2. Simulation Method. The Nanoparticles considered here are approximate spheres of various radii R constructed from face-centered cubic (fcc) lattice of Lennard-Jones atoms. We generate seven different sizes of nanoparticles ranging from 603 to 44,403 atoms in a single particle, which correspond to approximately $R = 5.4\sigma$ to $R = 22.7\sigma$. The particles are not perfect spheres; there exist faceted faces in the particles due to the underlying fcc lattice of atoms. To simulate collisions, two identical particles are placed with facets facing each other (see Fig. 4.6(a)) and they are given equal and opposite center-of-mass velocities.

We use classical MD for this study and consider nanoparticles constituent atoms whose potential is expressed by Lennard-Jones (L-J) potential:

$$V(r_{ij}) = \begin{cases} 4\epsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^6 \right] & (r_{ij} \leq r_c), \\ 0 & (r_{ij} > r_c), \end{cases} \quad (2.1)$$

where r_{ij} is an inter-atomic distance between atoms labeled by i and j , and r_c is a cut-off distance. We set $r_c = 2.5\sigma$ for atoms interacting within a single particle, which is typically used for the L-J potential. In addition, we ignore particle-particle adhesive interactions, and therefore, we set $r_c = 2^{1/6}\sigma$ (purely repulsive) for interactions between atoms belonging to different particles.

Note that the units used here are L-J units, i.e., the unit for energy: ϵ , the unit for distance: σ . In addition to these units, the unit for velocity: $\sqrt{\epsilon/m}$, the unit for temperature: ϵ/k_B , where k_B is Boltzmann constant, and the unit for time: $1/\sqrt{\epsilon/m\sigma^2}$ are used.

The full simulation procedure follows a number of steps. First, two identical particles are placed and thermally equilibrated over 10,000 MD steps, controlling temperature $T = 0.02\epsilon/k_B$ using the Nose-Hoover thermostat. After reaching an equilibrium state, the particles are given equal and opposite center-of-mass velocities and are thus set to collide with

a relative collision velocity of v_{coll} . The collision process is carried out in the microcanonical ensemble (NVE). For most system sizes, v_{coll} is varied from $0.03 \sqrt{\epsilon/m}$ to $3.3 \sqrt{\epsilon/m}$, where m is mass of an atom. The time step is set to be $\Delta t = 0.005 / \sqrt{\epsilon/m\sigma^2}$ for all cases. The relative residue of the total energy of a system during calculations is conserved to around 10^{-5} . This is achieved for finite cutoffs by shifting the potential to zero at r_c . In order to quantify errors in calculated quantities, we perform multiple simulations for each collision velocity by setting different initial thermal velocities. The systems and number of independent runs carried out for ensemble averaging are given in Table 2.1.

TABLE 2.1
The systems for simulations

Radius R / σ	Number of atoms N	Number of data
5.4	603	10
6.5	1,055	10
8.1	2,093	10
11.4	5,481	10
14.6	11,849	10
18.9	25,315	1
22.7	44,403	10

3. Background.

3.1. Coefficient of restitution. The coefficient of restitution e is defined by momentum balance using relative velocity v'_r after collision and relative velocity v_r before collision:

$$e \equiv \frac{v'_r}{v_r}. \quad (3.1)$$

This is a standard definition and is widely used. This definition is practically convenient to obtain coefficient of restitution by an experiment because to measure velocities of objects before and after collision is relatively easy. It, however, does not tell anything about the mechanisms of energy dissipation in collisions, especially regarding the internal degrees of freedom. In order to investigate these details of collisions, it is more advantageous to consider energy balance substituting in the coefficient of restitution as defined by momentum balance. Our study utilizes classical MD which computes microscopical quantities (internal degrees of freedom), positions and velocities, of each individual atom and allows us to recover macroscopic quantities such as the center-of-mass translational kinetic energy of the particle and temperature of the system.

Consider the energy balance of a collision process. The total energy E of the system is composed of the energy associated with average motion of the particles: center-of-mass translational kinetic energy K_{cm} and rotational kinetic energy around center-of mass of each particle K_{rot} as well as other forms of macroscopic kinetic energy K_{else} (e.g., vibrational modes of an elastic sphere); and energy of internal modes (e.g., fluctuations about the average motion, and internal energy of particles) E_{int} .

$$E = K_{\text{cm}} + K_{\text{rot}} + K_{\text{else}} + E_{\text{int}}. \quad (3.2)$$

Since we perform simulations in the microcanonical ensemble, the system's total energy is conserved. Therefore, when the time elapses long enough after collision, the energy difference before and after collision is zero:

$$E' - E = \Delta K_{\text{cm}} + \Delta K_{\text{rot}} + \Delta K_{\text{else}} + \Delta E_{\text{int}} = 0. \quad (3.3)$$

The coefficient of restitution in the center of mass reference frame can be written as

$$e = \sqrt{\frac{K'_{\text{cm}}}{K_{\text{cm}}}}. \quad (3.4)$$

Using the kinetic energy loss $\Delta K_{\text{cm}} = K'_{\text{cm}} - K_{\text{cm}}$, it follows that

$$\begin{aligned} e &= \sqrt{\frac{K_{\text{cm}} - \Delta K_{\text{rot}} - \Delta K_{\text{else}} - \Delta E_{\text{int}}}{K_{\text{cm}}}}, \\ &= \sqrt{1 - \frac{\Delta(K_{\text{rot}} + K_{\text{else}} + E_{\text{int}})}{K_{\text{cm}}}}. \end{aligned} \quad (3.5)$$

The term $\Delta(K_{\text{rot}} + K_{\text{else}} + E_{\text{int}})$ represents loss of translational kinetic energy during collision. In general, for head-on collisions $\Delta K_{\text{rot}} = 0$. In this study, however, we consider rotational kinetic energy about center of mass of each particle. This is necessary even though we consider nominally head-on collisions; results shown in Fig. 4.3(b) indicate some particles rotate around their own center of mass after collision. This is possible since the particles before collision have thermal (fluctuating) velocities; hence they can acquire rotational energy due to the symmetry breaking in collision from fluctuations of individual atoms. We, therefore, include rotational kinetic energy in the energy balance.

3.2. Yielding velocity. Johnson [2] has derived a relationship for the velocity at the onset of material yield and plastic deformation v_Y ,

$$\frac{1}{2} M v_Y^2 = \frac{53 R'^3 Y^5}{E^{*4}}, \quad (3.6)$$

where Y is yield stress of a particle, $M = Nm/2$, $R' = R/2$, $E^* = E/[2(1 - \nu^2)]$, Nm , R , E are the mass, radius, and Young's modulus, and ν is Poisson's ratio. The theoretical yield stress for an ideal fcc lattice is given by

$$Y \approx \frac{G}{10}, \quad (3.7)$$

where G is the shear modulus. Quesnel *et al.* determined Young's modulus, Poisson's ratio, and shear modulus in [100] direction for the fcc Lennard-Jones solid using molecular dynamics [7]. Their values $E = 61.1\epsilon/\sigma^3$, $G = 57.2\epsilon/\sigma^3$, $\nu = 0.347$ and Eq. 3.6 give the yielding velocity for fcc lattice,

$$v_Y = 0.945 \sqrt{\frac{(R/\sigma)^3}{N}} \sqrt{\epsilon/m}. \quad (3.8)$$

Note, the yielding velocity does not depend on the particle size because the number of atoms N grows cubically as the radius of a particle R increases, then the ratio R^3/N in Eq. 3.8 stays constant.

4. Results.

4.1. particle temperature pre-collision and post-collision. The temperature T of the system is related to the kinetic energy associated with velocity fluctuations about the average velocity. We calculate the temperature by

$$\frac{m}{2} \left\langle \sum_{i=1}^N (\mathbf{v}_i - \mathbf{v}_{\text{cm}})^2 \right\rangle = \frac{3}{2} N k_B T, \quad (4.1)$$

where N represents the number of atoms in a system, \mathbf{v}_i atom's velocity, and \mathbf{v}_{cm} center-of-mass velocity of the particle. To obtain the temperature of the system, the center-of-mass velocity of the particle to which an atom belongs is subtracted from individual atom's velocity. In Fig. 4.1, the bar graph shows the distribution of atom speeds obtained in our simulation for $N = 5481$, $T = 0.02\epsilon/k_B$ after it is equilibrated. The continuous line depicts the theoretical distribution (Maxwellian) for the given temperature.

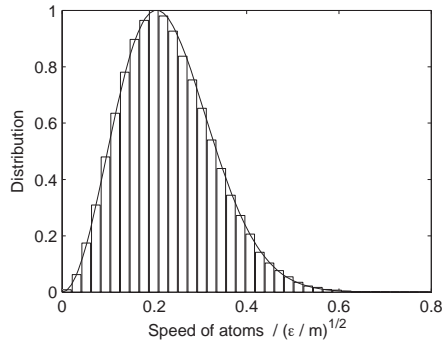


FIG. 4.1. The distribution of atoms' speed shows Maxwellian ($N = 5481$, $T = 0.02\epsilon/k_B$). The solid line is for the theoretical distribution.

The velocity distributions of the atoms after collision are shown in Fig 4.2. The x -axis is taken along the collision axis of two particles, and the other axes, y and z , are perpendicular to it. The velocity distributions are very nearly Gaussian, and little difference is seen in the three component distributions.

4.2. Energy conservation of the system. The time evolution of total energy, kinetic energy, and potential energy of the system for $N = 5481$, $v_{\text{coll}} = 0.66$, and $0.6\sqrt{\epsilon/m}$ is shown in Fig. 4.3. The calculation for equilibrating the system until 10,000 MD time step is done in the canonical ensemble (NVT). After the equilibration, the particles are given a translational velocity in order to collide each other. In this collision state, the simulation is run in the NVE . The fluctuations of the total energy up to 10,000 MD step come from controlling system's temperature. At 10,000 MD time step, the sudden step in both total energy and kinetic energy are due to the kinetic energy associated with particles' center-of-mass motions. The increase of potential energy and the drop of kinetic energy at 12,500 MD time step indicate the beginning of collision process. Both kinetic and potential energies long enough after collision (time step $> 20,000$ MD step) stay constant in time. For this system, the particles start to rotate at the beginning of the collision, which is shown a sharp peak in Fig. 4.3(b), and the rotational energies of both particles are conserved from the end of the contact at about 16,000 MD step. The loss of translational kinetic energy of this system during the collision is roughly $4,000\epsilon$, and rotational energy of the system post-collision is about 4ϵ , which is much less than the translational kinetic energy loss. Therefore, the rotation of particles is not significant for this system. The temperature of each particle is plotted on Fig. 4.3(c). The temperature is controlled to be $0.02\epsilon/k_B$ until 10,000 step. As the collision begins, the temperature starts to increase, and at the end of the collision it reaches another equilibrium state, gaining energy from part of the translational kinetic energy loss.

4.3. Coefficient of restitution. Now we consider the coefficient of restitution for $N = 5481$. In Fig. 4.4 three methods of determining the coefficient of restitution are shown. The circles are for the coefficient of restitution computed from center-of-mass velocities defined in

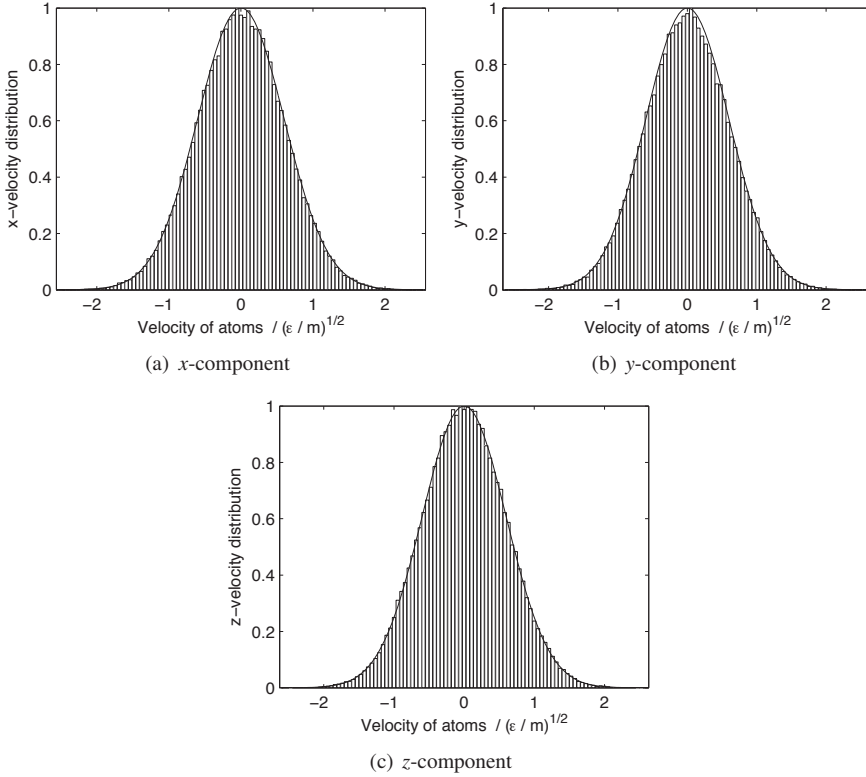


FIG. 4.2. Velocity distributions of a system, $N = 5481$, $v_{\text{coll}} = 3.3 \sqrt{\epsilon/m}$, after collision shows Gaussian distribution.

Eq. 3.1. The triangles are for the coefficient of restitution from kinetic energy K_{cm} and “loss of energy” into rotational kinetic energy and internal degrees of freedom $\Delta(K_{\text{rot}} + E_{\text{int}})$. The stars are for the coefficient of restitution from translational kinetic energy K_{cm} and “loss of energy” into internal degrees of freedom ΔE_{int} only, which considers only gain in thermal kinetic energy as “loss of energy” as seen in the macroscopically observed reduction in translational center-of-mass velocity of the particles. Both are defined in Eq. 3.5. It can be seen that the coefficient of restitution calculated from center-of-mass velocity only compares well with the one that considers rotational kinetic energy gain and internal kinetic energy gain as the energy losses. The coefficient of restitution that counts only thermal energy as the energy loss is a bit higher at $0.25 \sqrt{\epsilon/m} < v_{\text{coll}} < 0.5 \sqrt{\epsilon/m}$. It is interesting that in this region, particles that rotate about their center of mass are observed after collision, therefore this form of “energy loss” must be considered.

The reason some particles rotate after collision is that random motions of atoms in a particle exert force on the other particle during collision. If the direction of total force between the particles does not coincide with center of mass, the force generates torque to rotate the particle. If many simulations are done with different initial conditions, the direction of rotations should be random since it is derived from random motions. To verify this, the initial thermal velocity is generated randomly, and ten simulations whose initial thermal velocities are different have been performed. The results exhibit various rotational directions, and hence rotations of particles after collisions appear to be consistent with random motions of atoms.

The coefficient of restitution obtained in our numerical simulations for all system sizes

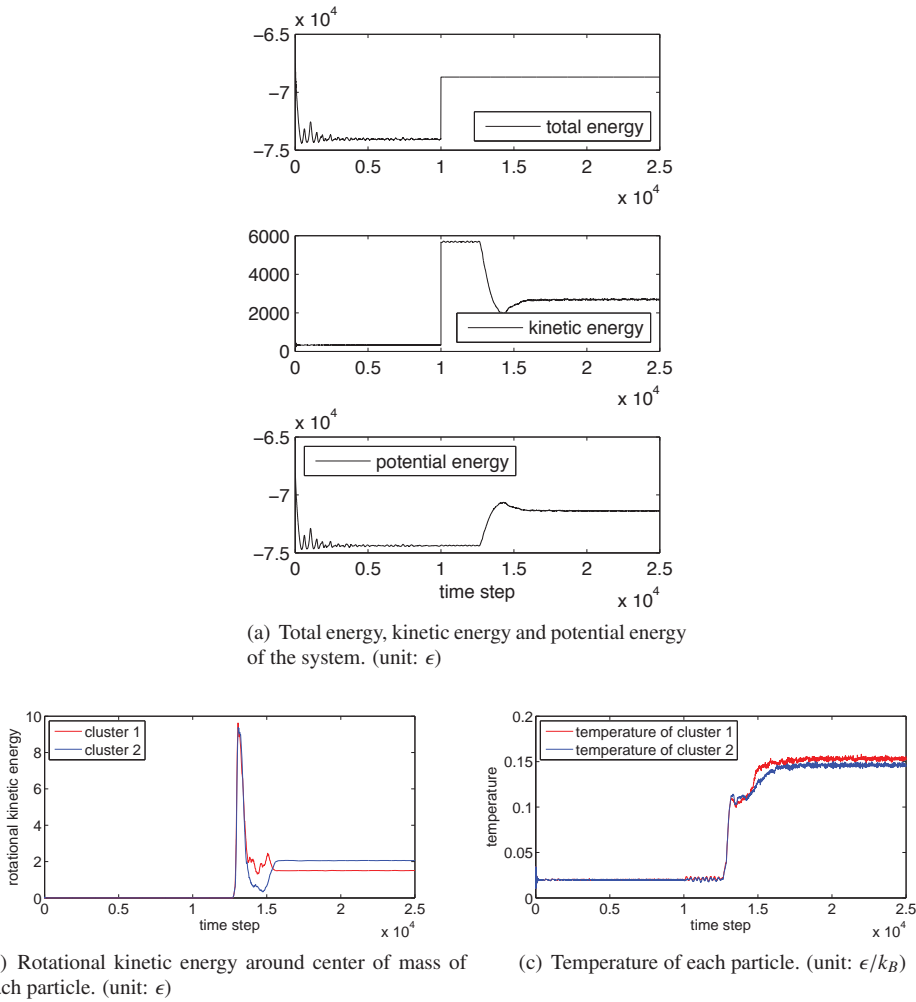


FIG. 4.3. A sample of energy transformation. $N = 5481$ and $v_{\text{coll}} = 2.0 \sqrt{\epsilon/m}$.

is shown in Fig. 4.5. Fig. 4.5(c) is for just $N = 5481$ taken out from Fig. 4.5(b).

In our simulations, the collision velocity is varied from $v_{\text{coll}} = 0.03 \sqrt{\epsilon/m}$ to $3.3 \sqrt{\epsilon/m}$. In this range of velocity, several regimes in the coefficient of restitution in terms of a particle's deformation modes are observed: an elastic deformation regime, a plastic deformation regime, and, in addition to them, there appears to be phase transition effects at the higher end of the plastic regime. The elastic region is in $v_{\text{coll}} < 0.5 \sqrt{\epsilon/m}$, and the plastic region is in $v_{\text{coll}} \geq 0.5 \sqrt{\epsilon/m}$. The phase transition region at which the particle surface appears to melt is at $v_{\text{coll}} \geq 2.2 \sqrt{\epsilon/m}$. There exists one more significant transition at $v_{\text{coll}} 1.3 \sqrt{\epsilon/m}$ that shows distinct deformation modes.

The coefficient of restitution in the elastic regime is closer to unity than in the plastic deformation regime as expected. For most system sizes, the coefficient of restitution takes a peak at around $v_{\text{coll}} = 0.05 \sqrt{\epsilon/m}$. The system of $N = 11,849$ indicates a different behavior; the coefficient of restitution keeps on increasing below $v_{\text{coll}} = 0.05 \sqrt{\epsilon/m}$ and goes above unity. Although these results are intriguing, the error bars on data for $v_{\text{coll}} < 0.015 \sqrt{\epsilon/m}$ are

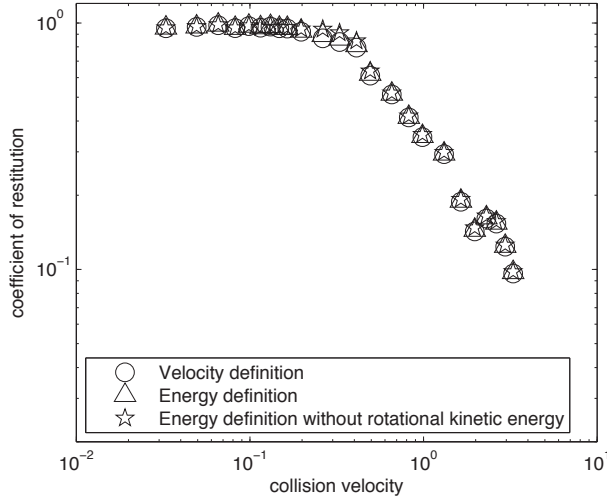


FIG. 4.4. Comparison with coefficient of restitution defined by energy and velocity. (System size $N = 5481$)

quite large, therefore further discussion should be made after collecting more data to reduce the error bars. For a velocity of $0.2 \sqrt{\epsilon/m} \leq v_{\text{coll}} \leq 0.33 \sqrt{\epsilon/m}$, it seems that the particle-size dependence is observed, comparing the coefficient of restitution at the same velocity, $v_{\text{coll}} = 0.2, 0.26$, and $0.33 \sqrt{\epsilon/m}$. As the particle size gets larger, the coefficient of restitution becomes lower.

Next, we look at the plastic regime, $0.5 \sqrt{\epsilon/m} \leq v_{\text{coll}} \leq 3.3 \sqrt{\epsilon/m}$. In this regime, different deformation modes are observed, dislocation and large deformation as well as surface melting. There appear to be several kinks at $v_{\text{coll}} = 0.4, 1.0, 2.0$ and $2.6 \sqrt{\epsilon/m}$. For $0.5 \sqrt{\epsilon/m} \leq v_{\text{coll}} \leq 1.0 \sqrt{\epsilon/m}$, particles with $N = 5481$ have a single dislocation along its crystal face (see Fig. 4.6(b)), while particles with $N = 44,403$ have multiple dislocations due to the collision. In the range of $1.2 \sqrt{\epsilon/m} \leq v_{\text{coll}} \leq 2.0 \sqrt{\epsilon/m}$, the collision causes large deformation of the particles, and they have more than simple dislocation bands; their shapes are no longer spherical (see Fig. 4.6(c)). In the case of collision velocity $v_{\text{coll}} > 2.0 \sqrt{\epsilon/m}$, particles have very little of their original crystal structure left after collision. Atoms on the surface move freely, and the surface of particles melts (see Fig. 4.6(d)). Thus, the plateau at about $v_{\text{coll}} = 2.0 \sqrt{\epsilon/m}$ seems to coincide with a phase transition of L-J atoms.

In the plastic regime, the coefficient of restitution appears to be inversely proportional to the collision velocity to some power (i.e., the coefficient of restitution follows a power law with exponent -1). Therefore, it is expressed in a form of $e = c' v_{\text{coll}}^n$. In Table 4.1, the exponent and coefficient for each region between adjacent separated by kinks are shown. A theoretical approach for deriving the relation between the coefficient of restitution and the collision velocity of macroscopic particles that plastically deform has been carried out by Johnson [2], and gives:

$$e = c' v_{\text{coll}}^{-1/4}, \quad (4.2)$$

where c' is determined by the geometry and material properties of the particles. The coefficient of restitution depends on the $-1/4$ power of collision velocity, which differs from our results. The possible reasons for the disagreement may stem from the system sizes, the L-J potential, or the NVE ensemble. The system sizes of our simulations are much smaller than macroscopic ones. If the sizes are the major cause of the disagreement, to increase the

sizes of simulations would resolve it. We are carrying out much larger calculations to determine this. The L-J potential is one of the simplest potential functions, and it may yield unrealistic simulation results when compared to other materials. To employ more realistic potential function may reproduce results of coefficient of restitution close to the theoretical ones. Finally, the *NVE* ensemble keeps the system energy constant, as a result, the loss of the center-of-mass translational kinetic energy is considerably converted into the system's temperature as shown in Fig. 4.3(c). However the energy of a real system during collision may not be kept constant. In reality, the initial kinetic energy could be sucked out and transferred into the environment during the collision, and the system's temperature may not increase as much as the temperature that the figure shows. A more realistic model that shows energy dissipation into the environment could lead to higher restitution of coefficient. Simulations using the *NVT* ensemble that keeps system's temperature constant would incorporate similar effect of the energy dissipation and may improve the coefficient of restitution in the plastic deformation regime. We are also performing simulations with a weak Langevin thermostat applied to *NVE* to determine the nature of this effect.

TABLE 4.1
Theoretical and computational coefficient and exponent for the power law, $e = cv_{\text{coll}}^n$.

$v_{\text{coll}}/\sqrt{\epsilon/m}$	Simulation		Theory
	Exponent n	Coefficient c	Exponent n
0.5–1.0	−0.91	0.34	−1/4
1.3–2.0	−1.79	0.47	−1/4
2.6–3.3	−2.09	1.16	−1/4

The yielding velocity obtained in our system is about $0.5\sqrt{\epsilon/m}$ for $N = 5481$ in Fig. 4.5(b). Although we need more data points around the yielding velocity to determine it precisely, it appears that all system sizes show the same yielding velocity. The theoretical yielding velocity derived by Johnson is obtained using Eq. 3.8, and we get $v_Y = 0.166\sqrt{\epsilon/m}$ for our fcc lattice particles made of L-J atoms. This value is smaller than our computational

TABLE 4.2
Yielding velocity v_Y obtained from the theory and our simulations.

	Simulation	Theory
$v_Y/\sqrt{\epsilon/m}$	0.5	0.166

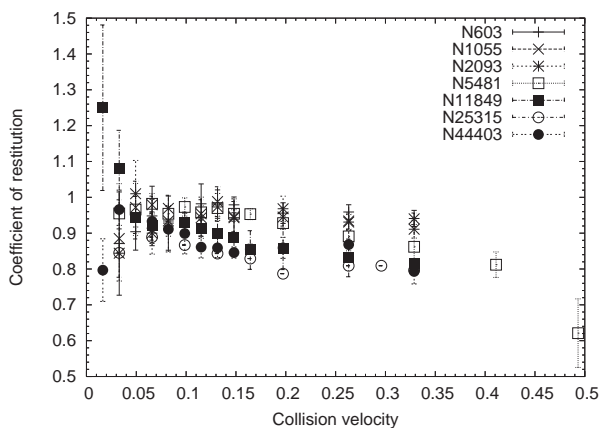
value, but the same order of magnitude. This is decent agreement, considering the use of the rough estimation of yielding stress Y in Eq. 3.7.

5. Conclusions. We have simulated nanoparticle collisions via the molecular dynamics code LAMMPS and we have obtained coefficients of restitution for two identical particles up to the system size $N = 44,403$ in a single particle under various collision velocities. We see distinct regimes in the coefficient of restitution ranging from elastic to plastic deformations. We have shown that there exists a peak of coefficient of restitution in elastic region, and there are three distinct regions in plastic deformation; dislocations of atoms in a particle, large deformations of a particle, phase transition effects. There exists large discrepancy in the behavior of the coefficient of restitution comparing nanoparticle and macroscopic particle, and hence further simulations for larger systems will be performed to examine the size effect as well as simulations of amorphous particles.

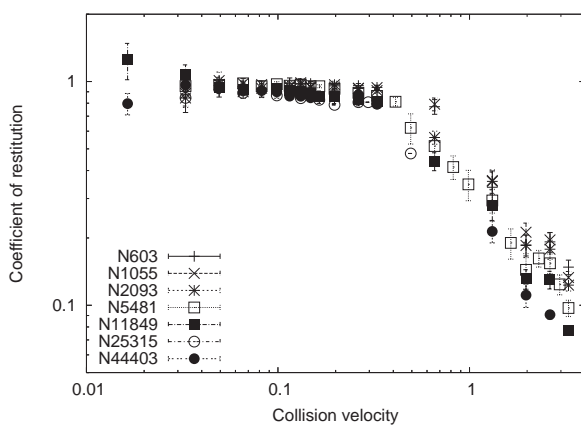
6. Acknowledgments. YT would like to thank Sandia National Laboratories for providing facilities and financial support for this research. SS thanks the Army Research Office for research support.

REFERENCES

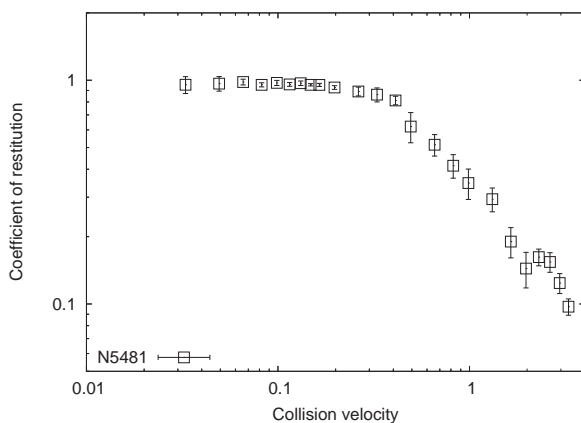
- [1] H. HERTZ, *J. Rein Angew Math.*, 92 (1881), pp. 156–171.
- [2] K. JOHNSON, *Contact mechanics*, Cambridge University Press, 1985, pp. 361–366.
- [3] M. KALWEIT AND D. DRIKAKIS, *Computational nanoclusters*, in ECCOMAS 2004, European congress on computational methods in applied science and engineering, July 2004.
- [4] ———, *Collision dynamics of nanoscale Lennard-Jones clusters*, *Physical Review B*, 74 (2006), p. 235415.
- [5] H. KUNINAKA AND H. HAYAKAWA, *Simulation of cohesive head-on collisions of thermally activated nanoclusters*, *Physical Review E*, 79 (2009), p. 031309.
- [6] ———, *Super-elastic collisions in a thermally activated system*, *Progress of Theoretical Physics Supplement*, 178 (2009), pp. 157–162.
- [7] D. QUESNEL, D. RIMAI, AND L. DEMEJO, *Elastic compliances and stiffnesses of the fcc Lennard-Jones solid*, *Physical Review B*, 48 (1993), pp. 6795–6807.

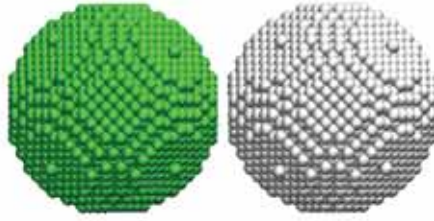


(a) Coefficient of restitution in linear-linear scale (Note: horizontal axis scale is different from 4.5(b).)

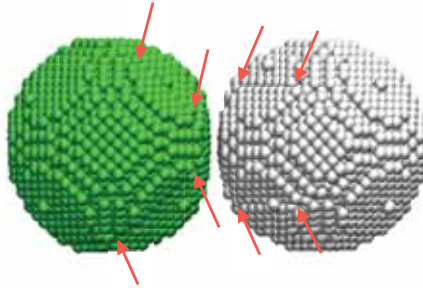
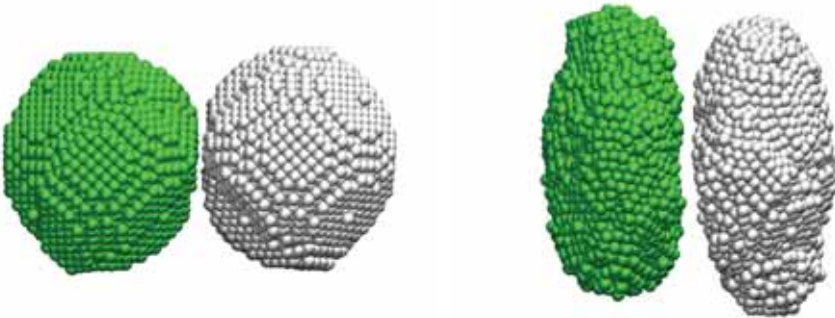


(b) Coefficient of restitution in log-log scale





(a) Before collision.

(b) After collision. $v_{\text{coll}} = 0.4 \sqrt{\epsilon/m}$. The dislocations along a single lattice face are seen in each particle.(c) After collision. $v_{\text{coll}} = 1.3 \sqrt{\epsilon/m}$. The particles are no longer spheres. (d) After collision. $v_{\text{coll}} = 3.3 \sqrt{\epsilon/m}$. The atoms on surfaces move freely.FIG. 4.6. Snap shots of particles ($N = 5481$) before and after collision at $v_{\text{coll}} = 0.4, 1.3, 3.3 \sqrt{\epsilon/m}$.

NUMERICAL SIMULATION OF THE PERFORMANCE OF A RESONANT TUNNELING DIODE

ANNE S. COSTOLANSKI* AND ANDREW G. SALINGER†

Abstract. A method for modeling the performance of a resonant tunneling diode (RTD) using Sandia's Trilinos software is described. The equations to model the behavior of an RTD are given, along with the corresponding numerical methods. An object-oriented structure using C++ classes is defined, and the method for incorporating Trilinos's nonlinear solver and continuation packages is detailed.

1. Introduction. Resonant tunneling diodes (RTDs) are nanoscale semiconductor devices which were first proposed by Tsu and Esaki in 1973 [15]. They predicted a phenomenon known as negative differential resistance, in which the current through a tunneling barrier reaches a local maximum when the electrons injected in the material achieve certain resonant energies. In 1974, Chang et al. fabricated the first RTD device that demonstrated evidence of negative differential resistance [3]. A decade later, Sollner et al. improved upon previously achieved results[14], and research on RTDs increased.

Resonant tunneling diodes have two distinct features that make them stand out from other semi-conductor devices: their high speed operation and their ability to produce negative differential resistance. Since tunneling is a very fast phenomenon (response time is on the order of picoseconds), the RTD is among the fastest devices ever made, with a maximum operational oscillation frequency projected to be over 1 THz [11]. Due to these features and their potential to be used as components in high speed electronic devices, RTDs have been studied extensively since the mid 1980s [5, 6, 8, 13, 17, 18].

In this paper, we propose an updated method for simulating the performance of an RTD using C++ and Trilinos. The paper is organized as follows: Section 2 describes the basic structure of an RTD, section 3 details the equations that model an RTD, and section 4 specifies how these equations are discretized. Section 5 discusses how to implement the discretized equations, with section 6 listing the details of the C++ classes and section 7 the specific Trilinos implementation. Our conclusions are listed in section 8, with future upgrades to the code discussed in section 9.

2. Resonant Tunneling Diodes. A typical resonant tunneling diode is composed of the following sections (see figure 2.1):

- Two regions of narrow energy band gap material that are heavily infused with electrons on either end of the device
- Several layers (barriers) of a larger energy band gap material on the interior of the device, separated from the doped regions by thin undoped spacer regions
- Quantum well region(s) separating the barrier regions from each other

The doped regions on either end of the device are generally large in size in comparison to the barrier and well regions. In a typical RTD, the quantum well thickness is around 50Å, and the barrier layers can range from 15 to 50Å. The region in the interior of the device containing the well(s), barrier layers, and spacer regions has a significantly lower doping level than the heavily doped regions on each end of the device. The spacer regions separating the barriers from the doped regions ensure that dopants do not diffuse to the barrier layers. Bias is then applied across the device to induce current flow.

*North Carolina State University, ascostol@ncsu.edu.

†Sandia National Laboratories, agsalin@sandia.gov

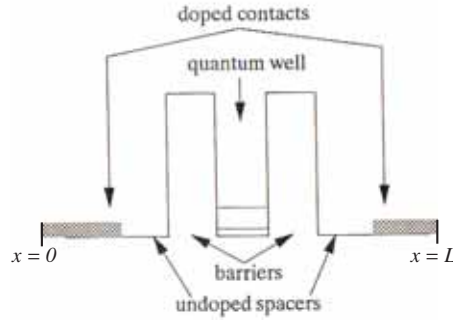


FIG. 2.1. Energy band diagram of a typical two barrier resonant tunneling diode, where the left endpoint is at the beginning of the device (i.e., corresponding to a position at $x = 0$), and the right endpoint is at the end of the device (i.e., corresponding to a position of $x = L$, where L is the length of the device.) From [11].

3. The Wigner-Poisson Equations. Since RTDs are built on the nanoscale, their physics is governed by quantum mechanics rather than classical physics. Thus, the standard drift-diffusion model no longer applies in calculating the current in the device. One of the primary models used for simulating the performance of a resonant tunneling diode is the Wigner-Poisson formulation, which couples the Wigner equation with Poisson's equation to predict the behavior of nanoscale semiconductor devices.

The Wigner function $f(x, k, t)$, which describes the distribution of electrons in the device, can be derived from the density matrix $\rho(z, z', t)$ for a multi-particle quantum mechanical system:

$$\rho(z, z', t) = \sum_j \psi_j(z) \psi_j^*(z') f(j) \quad (3.1)$$

where $\psi(\cdot)$ is an ensemble of wavefunctions that satisfy Schrödinger's equation, z, z' are two positions in coordinate space, t is time, and $f(j)$ is the Fermi-Dirac distribution function, which gives the probability that an electron will be in a given quantum mechanical state. The Wigner function is related to the density matrix via a fourier transform using a change of variables with $x = \frac{1}{2}(z + z')$ and $y = z - z'$, so that

$$f(x, k, t) = \int_{-\infty}^{\infty} e^{-iky} \rho(x + \frac{1}{2}y, x - \frac{1}{2}y, t) dy \quad (3.2)$$

where x, y are position variables and k is momentum. Taking the time derivative of the Wigner function and simplifying using trigonometric identities and the change of coordinates, we have

$$\begin{aligned} \frac{\partial f(x, k, t)}{\partial t} &= -\frac{\hbar k}{2\pi m^*} \frac{\partial f(x, k, t)}{\partial x} + \\ &- \frac{4}{\hbar} \int_{-\infty}^{\infty} f(x, k', t) dk' \int_0^{\infty} \sin(2y(k - k')) [U(x + y) - U(x - y)] dy \\ &= K(f) + P(f). \end{aligned} \quad (3.3)$$

Here $K(f)$ is defined as the kinetic energy term

$$K(f) = -\frac{\hbar k}{2\pi m^*} \frac{\partial f}{\partial x} \quad (3.4)$$

which corresponds to the effects due to kinetic energy on the distribution function f , where h is Planck's constant and m^* is the effective mass of an electron. The other term in the derivation, $P(f)$, is the potential term

$$P(f) = -\frac{4}{h} \int_{-\infty}^{\infty} f(x, k', t) T(x, k - k') dk' \quad (3.5)$$

with

$$T(x, k - k') = \int_0^{\infty} [U(x + y) - U(x - y)] \sin(2y(k - k')) dy \quad (3.6)$$

where $U(x)$ is the potential energy inside the device. To determine the potential energy, we need to incorporate both the energy band function, $\Delta_c(x)$, defined by the barriers and wells within the device as well as the electrostatic potential, $u_p(x)$, to account for the voltage applied to the device. Thus

$$U(x) = \Delta_c(x) + u_p(x) \quad (3.7)$$

where $u_p(x)$ is the solution to Poisson's equation

$$\frac{d^2 u_p(x)}{dx^2} = \frac{q^2}{\epsilon} [N_d(x) - n(x)] \quad (3.8)$$

with q the charge on a electron, ϵ the dielectric constant (which is material dependent), $N_d(x)$ the doping profile of the device, and $n(x)$ the electron density function, defined as

$$n(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} f(x, k) dk. \quad (3.9)$$

Poisson's equation has boundary conditions dependent on the amount of voltage applied across the device

$$u_p(0) = V_0, \quad u_p(L) = V_L \quad (3.10)$$

where V_0 is the initial voltage at the left side of the device, and V_L the voltage at the right side. Traditionally $V_0 = 0$ and $V_L = -V$ with $V \geq 0$.

Complete details of the derivation are shown by Buot and Jensen [2]. This derivation, however, does not take the effects of electron interactions into account. Several different methods have been tried to incorporate scattering effects into the equation, but detailed treatments create a heavy computational burden [1]. Thus we will use a low-order relaxation-time approximation in this model [16] to account for collision interactions between electrons in the device:

$$S(f) = \frac{1}{\tau} \left[\frac{f_0(x, k)}{\int_{-\infty}^{\infty} f_0(x, k) dk} \int_{-\infty}^{\infty} f(x, k) dk - f(x, k) \right]. \quad (3.11)$$

Here $S(f)$ is termed the scattering function, with τ the relaxation time of an electron in the device. $f_0(x, k)$ is the equilibrium Wigner distribution function, which is the solution to equation (3.3) with no change in the bias voltage applied across the device; i.e., $V_L = V_0$. Thus the full Wigner equation is

$$\frac{\partial f(x, k, t)}{\partial t} = K(f) + P(f) + S(f) = W(f), \quad (3.12)$$

and to solve the Wigner equation, boundary conditions are imposed on the distribution function f :

$$f(0, k) = \frac{4\pi m^* k_B T}{h^2} \ln \left\{ 1 + \exp \left[-\frac{1}{k_B T} \left(\frac{h^2 k^2}{8\pi^2 m^*} - \mu_0 \right) \right] \right\}, k > 0 \quad (3.13)$$

$$f(L, k) = \frac{4\pi m^* k_B T}{h^2} \ln \left\{ 1 + \exp \left[-\frac{1}{k_B T} \left(\frac{h^2 k^2}{8\pi^2 m^*} - \mu_L \right) \right] \right\}, k < 0 \quad (3.14)$$

where k_B is Boltzmann's constant, T is the temperature, and μ_0 and μ_L are the Fermi energies at the corresponding ends of the device.

To analyze the performance of the device, we will drop the time dependence and focus on the steady state solution, since RTDs have an incredibly fast response time (on the order of picoseconds) and we are interested in the long-term behavior ($t > 1$ sec). The main quantity of interest in modeling an RTD is the current density, which is the measure of device performance. The current density can be calculated from the Wigner distribution $f(x, k)$ as

$$j(x) = \frac{h}{2\pi m^*} \int_{-\infty}^{\infty} k f(x, k) dk. \quad (3.15)$$

4. Discretizing the Wigner-Poisson equations. Since the system of equations specified in the Wigner-Poisson formulation is too difficult to solve analytically, numerical techniques are used to approximate the solution. First, the domain of the function is discretized using a uniform grid. The spatial domain is divided into N_x equally spaced grid points, with $x_1 = 0$ and $x_{N_x} = L$, and the grid spacing defined as $\Delta x = \frac{L}{N_x - 1}$.

To discretize the momentum space, the domain is truncated from $(-\infty, \infty)$ to $(-K_{max}, K_{max})$ where K_{max} is chosen such that when $|k| > K_{max}$, $f(x, k) \approx 0$. We will use $K_{max} = 0.25$ inverse Angstroms as a best approximation, based on prior work [4, 9]. Then the momentum domain can be divided into N_k equally spaced grid points, with the grid spacing given by $\Delta k = \frac{2K_{max}}{N_k}$. We will avoid using $k = 0$ as a grid point, since at $k = 0$, the kinetic term would be identically equal to zero, making any discretized matrix singular. Thus the momentum grid points can be defined as

$$k_j = \frac{(2j - N_k - 1)\Delta k}{2} \quad \text{for } j = 1, 2, \dots, N_k. \quad (4.1)$$

We will denote a solution of the Wigner distribution function f at a grid point (x_i, k_j) as f_{ij} .

To approximate the kinetic term $K(f)$, we will use an upwinding scheme since there are one-sided boundary conditions on the Wigner function. The standard second order approximation is

$$K(f_{ij}) \approx \begin{cases} -\frac{hk_j}{2\pi m^*} \left(\frac{-3f_{ij} + 4f_{i-1,j} - f_{i-2,j}}{2\Delta x} \right) & , \quad k_j < 0 \\ -\frac{hk_j}{2\pi m^*} \left(\frac{3f_{ij} - 4f_{i+1,j} + f_{i+2,j}}{2\Delta x} \right) & , \quad k_j > 0. \end{cases} \quad (4.2)$$

A first order Euler approximation is used at $x = \Delta x$ or $x = L - \Delta x$ as appropriate.

The potential $P(f)$ term is discretized using the second order composite trapezoidal rule

$$P(f_{ij}) \approx -\frac{4}{h} \sum_{j'=1}^{N_k} f_{ij'} T(x_i, k_j - k_{j'}) w_{j'} \quad (4.3)$$

where the $w_{j'}$ are weighting terms that are defined as:

$$w_{j'} = \begin{cases} \Delta k & \text{for } j' = 2, 3, \dots, N_k - 1, \\ \frac{\Delta k}{2} & \text{for } j' = 1, N_k \end{cases} \quad (4.4)$$

For the $T(x_i, k_j - k_{j'})$ term, we must make additional approximations to discretize this integral. Since the upper limit of the integrand in equation (3.6) is infinity, we must truncate the limit at a value $L_c \leq L$, where L_c is termed the correlation length. L_c may or may not correspond to a grid point in the spatial domain, so the weights used in the approximation for $T(x_i, k_j - k_{j'})$ must take this into account in order for the solution to be second order accurate. A derivation of these weights can be found in [9]. Thus, the $T(x_i, k_j - k_{j'})$ term can be discretized as:

$$T(x_i, k_j - k_{j'}) \approx \sum_{i'=1}^{N_c+1} [U(x_i + x_{i'}) - U(x_i - x_{i'})] \sin(2x_{i'}(k_j - k_{j'})) w_{i'} \quad (4.5)$$

where the $w_{i'}$ are the modified trapezoidal weights

$$w_{i'} = \begin{cases} \frac{\Delta x}{2} & \text{for } i' = 1 \\ \Delta x & \text{for } i' = 2, 3, \dots, N_c - 1 \\ \frac{\Delta x + h_{N_c}}{2} + \frac{h_{N_c} h_{N_c+1}}{2\Delta x} & \text{for } i' = N_c \\ \frac{h_{N_c}^2}{2\Delta x} & \text{for } i' = N_c + 1. \end{cases} \quad (4.6)$$

To compute the electrostatic potential $u_p(x)$ which is part of the potential energy $U(x)$, Poisson's equation can be discretized using a center difference formula, for which the second order method is

$$\frac{u_p(x_{i-1}) - 2u_p(x_i) + u_p(x_{i+1}))}{\Delta x^2} = \frac{q^2}{\epsilon} [N_d(x_i) - n(x_i)] \quad (4.7)$$

with $u_p(0) = V_0$ and $u_p(L) = V_L$. Within Poisson's equation, the doping density $N_d(x)$ is piecewise linear, but the electron density is a function of the Wigner distribution f . Thus, it can be approximated using the composite trapezoid rule

$$n(x_i) \approx \frac{1}{2\pi} \sum_{j=1}^{N_k} f_{ij} w_j \quad (4.8)$$

where the weights are the standard composite trapezoidal weights from equation (4.4).

The final term in the Wigner equation is the scattering term $S(f)$. We again discretize the integrals using the composite trapezoidal rule and the associated weights from equation (4.4):

$$S(f_{ij}) \approx \frac{1}{\tau} \left[\frac{f_0(x_i, k_j)}{\sum_{j'=1}^{N_k} f_0(x_i, k_{j'}) w_{j'}} \sum_{j'=1}^{N_k} f_{ij'} w_{j'} - f_{ij} \right]. \quad (4.9)$$

Once the Wigner function has been computed, the current density $j(x)$ can be calculated

$$j(x_i) \approx \frac{h}{2\pi m^*} \sum_{j=1}^{N_k} k_j f_{ij} w_j \quad (4.10)$$

where the weights are again those for the standard composite trapezoidal rule.

5. Numerical Implementation. The Wigner equation (3.12) is a standard nonlinear equation which can be solved using Newton's method, which finds a stationary point of the iterative equation $u_{i+1} = u_i - W'(u_i)^{-1} W(u_i)$, where $W'(u_i)$ is the Jacobian of $W(u)$ at u_i . When

$\|u_{i+1} - u_i\| \leq \epsilon$ where ϵ is an error tolerance, u_{i+1} is an approximate fixed point of the Wigner equation.

However, creating the iteration step $W'(u_i)^{-1}W(u_i)$ for the discretized Wigner equation is not computationally efficient, particularly when the $W'(u_i)$ matrix is large (typical implementations have on the order of 125,000 to 1 million grid points, which can put the Jacobian matrix at approximately 1 trillion elements). Instead, we will use an inexact Newton method to solve the system. (For more information on inexact Newton methods, see [7] and references therein.) With inexact Newton methods, the nonlinear equation is approximated by

$$\|W'(u_i)s + W(u_i)\| \leq \eta_i \|W(u_i)\| \quad (5.1)$$

where η_i is a forcing term that controls the size of the relative residual. Newton's method is used to compute the u_i , and the step s is approximated by a linear iterative method. For Wigner-Poisson, we will use GMRES to solve for the steps, since GMRES is a Krylov subspace method for solving non-symmetric systems that does not require the computation of an iteration matrix [7].

This method can be efficiently implemented using Sandia's Trilinos software, which has a variety of built-in inexact Newton method solvers, including Newton-GMRES. In addition, Trilinos allows the option to use a matrix-free implementation involving finite differences to approximate the Jacobian-vector product $W'(u_i)s$, since computing and discretizing the Jacobian of the Wigner-Poisson formulation would be extremely complicated.

6. C++ Class Descriptions. Since Trilinos is written in C++, the main program code is also written in C++ to take advantage of the object oriented structure. A number of classes and structs were created, each representing a different piece of the Wigner-Poisson equations. Each class has a Compute function to calculate either the appropriate action on the Wigner function f or to compute terms that will be used in calculating f . The two classes that involve finite difference approximations for the kinetic term and the Poisson term each have an Operator function that sets up the sparse matrix associated with the derivative term in the equation.

Data Inputs Three structs and two classes hold the constants associated with the Wigner-Poisson formulation. Note that the magnitudes of the data in each is dependent on the units used.

The three structs are Constants, MatProps, and DevProps, which contain standard mathematical constants, constants related to the materials used, and constants related to the device, respectively. Most of the C++ classes reference one or more of the structs as part of their Compute method. The two classes, Barrier and Doping, create the barrier and doping profiles that are used in the solution of the potential term $P(f)$. The Compute method calculates the relative energy band profile, $\Delta_c(x)$, and doping profile $N_d(x)$ at each grid point along the length of the device, which are used in equations (4.5) and (4.7) respectively.

Wigner classes The next set of classes were created to calculate the specific terms in the Wigner distribution function:

- KineticMethod for the kinetic term $K(f)$
- ScatterMethod for the scattering term $S(f)$
- PotentialMethod for the potential term $P(f)$

The Operator method of the KineticMethod class creates the sparse matrix associated with the discretization of the derivative term in equation (4.2), and applies the matrix to the Wigner distribution function f in the Compute method. ScatterMethod calculates the result of equation (4.9), and the PotentialMethod calculates the result of equation (4.3) for a given f . However, to compute the Wigner potential term, $T(x, k - k')$ must be known for all values of k' . Thus, several more classes were created to break up the computation of $T(x, k - k')$ into

more manageable pieces. These pieces must be computed before the Compute method in the `PotentialMethod` class is executed.

Potential term classes The classes that feed into the calculation of the Wigner potential term are:

- `TcMethod` and `TpMethod` to calculate each respective portion of the T integral
- `SinMat` to calculate the $\sin(2yk)$ terms
- `PoissonMethod` to calculate the electrostatic potential $u_p(x)$ used in the `TpMethod` class
- `EleDensMethod` to calculate the electron density function $n(x)$ used in the `PoissonMethod` class

Since the potential energy $U(x)$ in the computation of $T(x, k - k')$ is the sum of two functions, we can break $T(x, k - k')$ into two separate integrals and assign a C++ class to each:

$$T(x_i, k_j - k_{j'}) = \sum_{i'=1}^{N_c+1} [U(x_i + x_{i'}) - U(x_i - x_{i'})] \sin(2x_{i'}(k_j - k_{j'}))w_{i'} \quad (6.1)$$

$$= \sum_{i'=1}^{N_c+1} [u_p(x_i + x_{i'}) - u_p(x_i - x_{i'})] \sin(2x_{i'}(k_j - k_{j'}))w_{i'} \quad (6.2)$$

$$+ \sum_{i'=1}^{N_c+1} [\Delta_c(x_i + x_{i'}) - \Delta_c(x_i - x_{i'})] \sin(2x_{i'}(k_j - k_{j'}))w_{i'} \\ = T_p(x_i, k_j - k_{j'}) + T_c(x_i, k_j - k_{j'}) \quad (6.3)$$

where $T_p(x, k - k')$ represents the integral associated with the electrostatic potential $u_p(x)$ and $T_c(x, k - k')$ the integral associated with the energy band function $\Delta_c(x)$. The associated C++ classes are `TpMethod` and `TcMethod` respectively. The values of $\sin(2yk)$ for each $(y, k) \in [0, L] \times (-2K_{max}, 2K_{max})$ are computed and stored by the `SinMat` class, and then used to compute the integrals evaluated in the Compute methods of `TcMethod` and `TpMethod`. Due to the fixed nature of the inputs to `TcMethod`, $T_c(x, k - k')$ is calculated once and stored for use throughout the remaining computations. However, $T_p(x, k - k')$ depends upon $u_p(x)$, which in turn depends on the electron density $n(x)$, so additional classes were created to compute these functions.

The `PoissonMethod` class has an `Operator` method which creates the sparse matrix associated with the second order central difference approximation in equation (4.7), and the associated `Compute` method solves the linear system $Ax = b$. Here A is the sparse matrix and b includes the piecewise continuous doping profile $N_d(x)$ and the electron density $n(x)$ which is computed via the `EleDensMethod` class. The `Compute` method uses Trilinos' Amesos package to solve the system via a serial direct solver called KLU.

Finally, once the Wigner distribution function has been fully calculated, a separate class `CurrentMethod` computes the current density using equation (4.10).

7. Solution Process using Trilinos. To solve the Wigner-Poisson system using Trilinos, matrix and vector elements used in the computations are compatible with Epetra data structures, such as `Epetra.CrsMatrix` and `Epetra.Vector`, and the equations are solved using the nonlinear solver package NOX and the associated continuation package LOCA. A `Problem` class is created to call the `Compute` methods for each class involved in the continuation process, and a `ProblemInterface` class is created to implement the `LOCA::Epetra::Interface::Required` interface. The kinetic and potential terms (and when the ending voltage $V_L \neq V_0$, the scattering terms) are then combined to solve for the Wigner

function $f(x, k)$. The flowchart depicted in figure 7.1 represents these processes, where the classes above the dotted black line are called in the constructor, and the classes below the line are called in the ComputeF method.

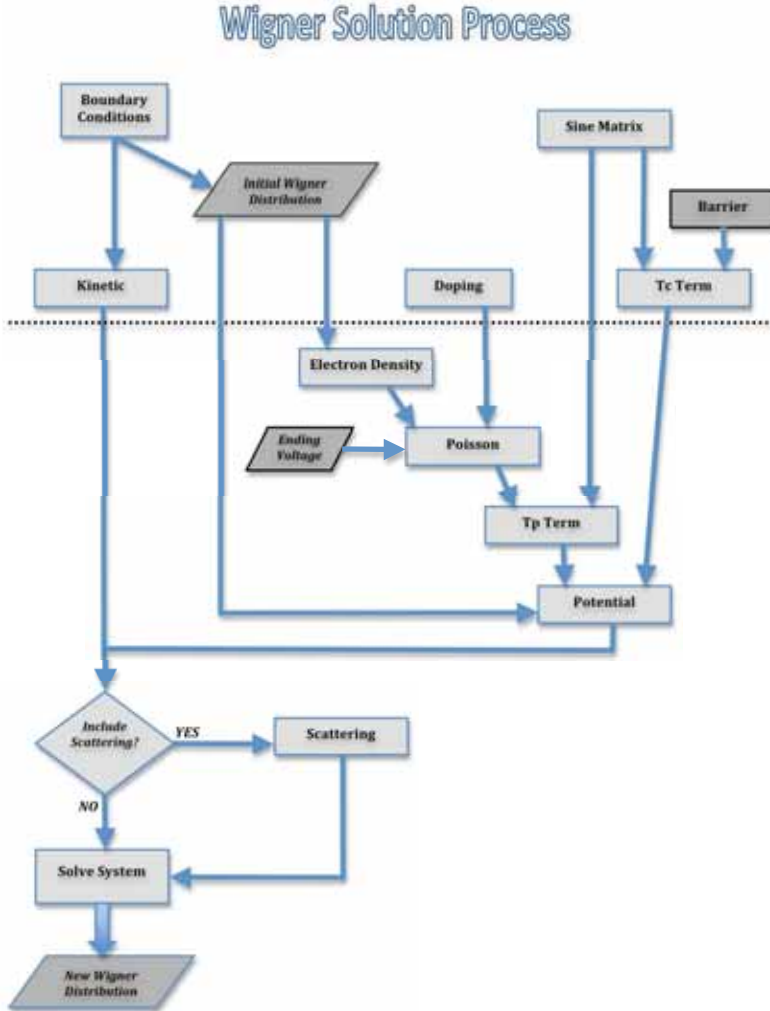


FIG. 7.1. Flowchart of the detailed object oriented solution process. The classes above the dotted black line are computed in the ProblemInterface constructor, and those below are part of the continuation. The dark grey shaded boxes indicate inputs that are subject to change via the continuation process.

The system of equations will be solved as two homotopy problems. The first involves continuation on the barrier profile to calculate the initial Wigner distribution f_0 . Continuation is started using a barrier profile $\Delta_c(x) \equiv 0$ and increases up to the full $\Delta_c(x)$. The second continuation run is to calculate the current density as a function of the bias voltage applied to the device, which starts at 0 and increases up to V_L . These two processes are depicted in the

second flowchart in figure 7.2.

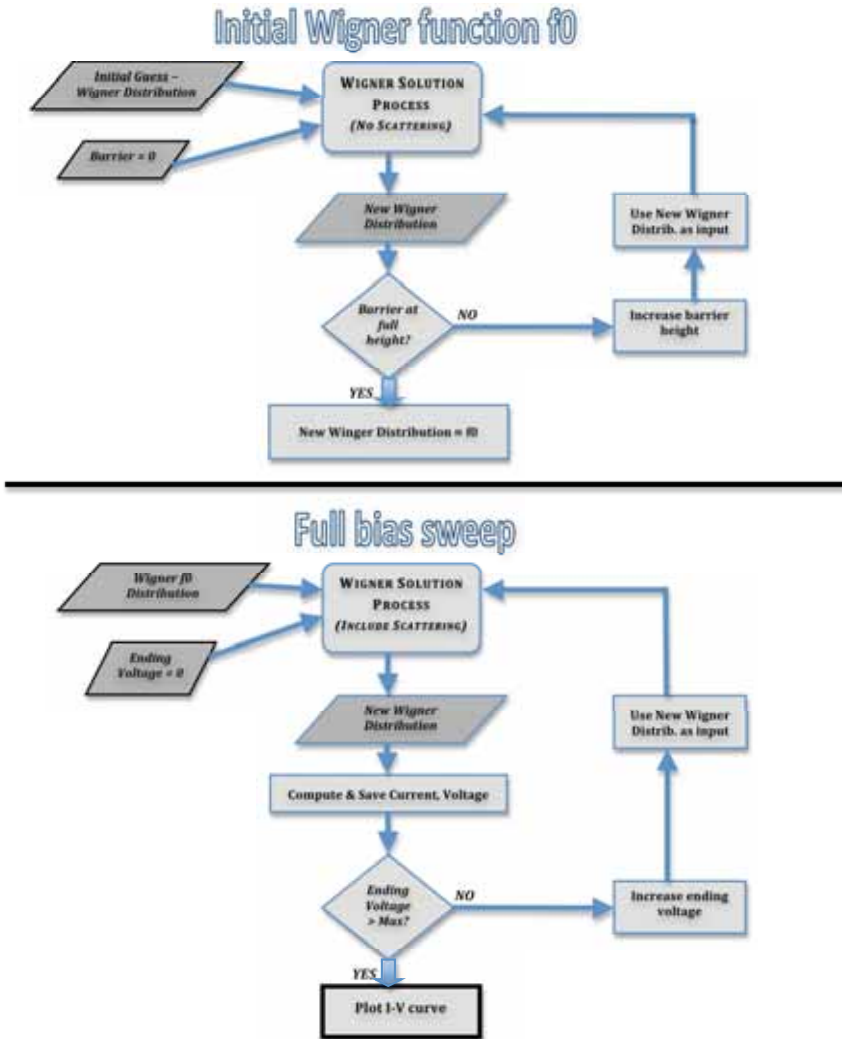


FIG. 7.2. High level flowcharts for the continuation processes. The top chart describes the continuation on the barrier profile to compute the initial Wigner distribution f_0 ; the bottom chart is the continuation process to compute the full voltage sweep from $V_0 = 0$ to V_L .

To start the computation, an initial guess for f must be chosen, taking the boundary conditions on the Wigner equation into account. Then equation (3.3) is solved with $\Delta_c(x) \equiv 0$ and $V_L = V_0 = 0$. The barrier profile is increased by multiplying $\Delta_c(x)$ by a constant and then increasing the constant incrementally from 0 to 1, solving for f at each incremental step. This is done *via* Trilinos by calling LOCA. The final Wigner solution f using the full barrier profile $\Delta_c(x)$ is the initial Wigner distribution f_0 .

Once f_0 has been computed, continuation is performed again, this time by including $S(f)$

in the Wigner computation and solving equation (3.12) for f using the ending bias value as the continuation parameter. Once each f is calculated, the current density can be computed and stored along with the associated voltage for use in producing a current vs. voltage curve. The voltage is then increased up to the final voltage value V_L .

8. Conclusions. The C++/Trilinos implementation defined above provides more computational efficiency as well as greater flexibility in modeling a variety of devices than previous models do. Although other implementations exist in both Matlab and Fortran, Trilinos is written in C++ to handle memory allocation and computations more efficiently, which will allow finer meshes to be simulated in shorter run times than those in previous work [12, 17]. Coupling Trilinos with a C++ implementation of the Wigner-Poisson formulation is more efficient than the previous Fortran/Trilinos implementation [9, 10], and Trilinos' ability to handle parallel computation will decrease run times even more.

The object oriented design of the C++ implementation gives flexibility for investigating new algorithms. For example, changing the discretization of the Poisson equation is a local change in one class. Similarly, tradeoffs between memory usage and FLOPS, such as storing the entire $\sin(2yk)$ matrix, can be easily investigated with a local change to one class.

Also, due to the flexibility inherent in the data input structures (such as the `Barrier` class and the `DevProps` struct), changes to the device can be made easily and consistently. Previous models hard-coded most of this information [9, 10, 12], such as the number of barriers and the material properties, and thus limited the types of devices that could be analyzed. Therefore, the C++/Trilinos implementation will increase the breadth of knowledge about RTDs and potentially other nanoscale devices that can be modeled using the Wigner-Poisson formulation.

9. Future Work. Once the code outlined in this paper is complete, there are a variety of enhancements to be made that will improve performance and increase the amount of information available on resonant tunneling diodes:

- **Incorporate fourth order methods.** Currently, most of the numerical approximations use second order accurate methods since they are slightly easier to code. The first improvement to be made will be to switch out the second order methods for fourth order methods, which will require minor changes in the Compute methods for several classes.
- **Incorporate analytic solutions.** In this and previous Fortran versions of the RTD model, numerical approximations are used to compute solutions involving the piecewise linear terms $\Delta_c(x)$ and $N_d(x)$. Partial solutions to the equations involving these terms can be computed analytically, and these solutions incorporated into the Wigner-Poisson formulation. This will entail changes to the `TcMethod` and the `PoissonMethod`.
- **Implement a non-uniform grid.** The current uniform grid places a large number of grid points in regions where the physics of the problem changes very little – mainly in the doped areas on either end of the device, and away from the $k = 0$ line in the momentum dimension. Implementing a non-uniform grid which concentrates the grid points in the areas with greater variety in the Wigner function will refine the solution in those regions while decreasing computation time.
- **Solve the time dependent Wigner equation.** Although equation (3.12) shows the time dependent nature of the Wigner equation, the solution process outlined here computes the steady state version of the problem. We would like to solve the full time dependent version to determine the oscillatory nature of the solution in the region of negative differential resistance, and to search for bifurcation values.

10. Acknowledgements. The work for this paper was partially supported by Army Research Office Grants W911NF-0710112 and W911NF-0910159, National Science Foundation Grant DMS-0707220, and NanoRTD LLC.

REFERENCES

- [1] P. BORBONE, M. PASCOLI, R. BRUNETTI, A. BERTONI, AND C. JACOBONI, *Quantum transport of electrons in open nanostructures with the Wigner-function formalism*, Phys. Rev. B, 59 (1999), pp. 3060–3069.
- [2] F. A. BUOT AND K. L. JENSEN, *Lattice Weyl-Wigner formulation of exact many-body quantum-transport theory and applications to novel solid-state quantum-based devices*, Phys. Rev. B, 42 (1990), pp. 9429–9457.
- [3] L. CHANG, L. ESAKI, AND R. TSU, *Resonant tunneling in semiconductor double barriers*, Applied Physics Letters, 22 (1973), pp. 562–564.
- [4] W. R. FRENSLEY, *Wigner-function model of a resonant-tunneling semiconductor device*, Phys. Rev. B, 36 (1987), pp. 1570–1580.
- [5] H. GRUBIN AND R. BUGGELN, *RTD relaxation oscillations, the time dependent Wigner equation and phase noise*, Journal of Computational Electronics, 1 (2002), pp. 33–37.
- [6] K. L. JENSEN AND F. A. BUOT, *Numerical simulation of intrinsic bistability and high-frequency current oscillations in resonant tunneling structures*, Physical Review Letters, 66 (1991), pp. 1078–1081.
- [7] C. T. KELLEY, *Iterative Methods for Linear and Nonlinear Equations*, no. 16 in Frontiers in Applied Mathematics, SIAM, Philadelphia, 1995.
- [8] N. C. KLUKSDAHL, A. M. KRIMAN, D. K. FERRY, AND C. RINGHOFFER, *Self-consistent study of the resonant-tunneling diode*, Phys. Rev. B, 39 (1989), pp. 7720–7735.
- [9] M. S. LASATER, *Numerical Methods for the Wigner-Poisson Equations*, PhD thesis, North Carolina State University, Raleigh, North Carolina, 2005.
- [10] M. S. LASATER, C. T. KELLEY, A. SALINGER, P. ZHAO, AND D. L. WOOLARD, *Simulating nanoscale semiconductor devices*, International Journal of High Speed Electronics and Systems, 16 (2006), pp. 677–690.
- [11] K. NG, *Complete Guide to Semiconductor Devices*, Wiley-Interscience, New York, NY, 2 ed., 2002.
- [12] G. RECINE, *Numerical Simulation of Quantum Electron Transport in Nanoscale Resonant Tunneling Structures*, PhD thesis, Stevens Institute of Technology, Hoboken, New Jersey, 2004.
- [13] T. SOLLNER, E. BROWN, W. GOODHUE, AND H. LE, *Observation of millimeter-wave oscillations from resonant tunneling diodes and some theoretical considerations of ultimate frequency limits*, Applied Physics Letters, 50 (1987), pp. 332–334.
- [14] T. SOLLNER, W. GOODHUE, P. TANNENWALD, C. PARKER, AND D. PECK, *Resonant tunneling through quantum wells at frequencies up to 2.5THz*, Applied Physics Letters, 43 (1983), pp. 588–564.
- [15] R. TSU AND L. ESAKI, *Tunneling in a finite superlattice*, Applied Physics Letters, 22 (1973), pp. 562–564.
- [16] P. ZHAO, H. L. CUI, AND D. L. WOOLARD, *Dynamical instabilities and I-V characteristics in resonant tunneling through double barrier quantum well systems*, Phys. Rev. B, 63 (2001), pp. 075302–1–075302–14.
- [17] P. ZHAO, H. L. CUI, D. L. WOOLARD, K. L. JENSEN, AND F. A. BUOT, *Simulation of resonant tunneling structures: Origin of the I-V hysteresis and plateau-like structure*, Journal of Applied Physics, 87 (2000), pp. 1337–1349.
- [18] P. ZHAO, D. L. WOOLARD, AND H. L. CUI, *Multisubband theory for the origination of intrinsic oscillations within double-barrier quantum well systems*, Physical Review B, 67 (2003), p. 085312.

Architecture and Networking

Computational algorithms frequently require large numbers of reliable operations and ever increasing amounts of power. The articles in this section present novel architectures to better utilize available resources and new simulation techniques to better understand the reliability and power requirements of future machines.

Curry et al. present a new RAID capability for storage in high performance computing that offloads error correction calculations to a graphics processing unit (GPU). The performance of this new technology out performs a software based RAID controller and, due to the commodity nature of the GPU, compares favorably to expensive hardware based controllers. *Van Vorst et al.* discuss progress made in the development of a real-time large-scale network simulation testbed on Cray XT supercomputers. This work promises to provide network researchers with an environment capable of running large-scale simulations. *Ruiz Varela et al.* use simulations to compare various techniques for resilient computations on exascale platforms. The authors also propose a new simulation method for exploring efficient resilience mechanisms on future machines. *Vaughan et al.* discuss the use of root cause analysis to better understand the cause and source of failures in supercomputing environments. They demonstrate the effectiveness of these techniques using a combination of system logs and statistical analysis. *Merritt and Pedretti* evaluate several techniques for data distribution for multi-threaded multi-core computers. Based on this evaluation, the authors propose a kernel level dynamic runtime data distribution system. *Thompson et al.* discuss extending the Structural Simulation Toolkit to handle and analyze power consumption data. Additionally, using existing power modeling toolkits results, the authors present sample results of using this new capability. *Kersey and Rodrigues* explore the use of process layers for large-scale discrete event simulation of computer hardware. Based on a use case study, they propose performance improvement methods that avoid some of the common costs associated with process layers. *Williams and Rodrigues* discuss and motivate the use of the Structural Simulation Toolkit for reliability simulation in the context of high-performance computing. The authors also describe the need for reliability simulation and the components that go into constructing such a simulation.

E.C. Cyr
S.S. Collis

December 17, 2010

A LIGHTWEIGHT, GPU-BASED SOFTWARE RAID SYSTEM

MATTHEW L. CURRY*, H. LEE WARD†, ANTHONY SKJELLUM‡, AND RON BRIGHTWELL§

Abstract. While RAID is the prevailing method of creating reliable secondary storage infrastructure, many users desire more flexibility than offered by current implementations. Traditionally, RAID capabilities have been implemented largely in hardware in order to achieve the best performance possible, but hardware RAID has rigid designs that are costly to change. Software implementations are much more flexible, but software RAID has historically been viewed as much less capable of high throughput than hardware RAID controllers. This work presents a system, Gibraltar RAID, that attains high RAID performance by offloading the calculations related to error correcting codes to GPUs. This paper describes the architecture, performance, and qualities of the system. A comparison to a well-known software RAID implementation, the md driver included with the Linux operating system, is presented. While this work is presented in the context of high performance computing, these findings also apply to a general RAID market.

1. Introduction. RAID (Redundant Array of Independent Disks) is a technology that allows for groups of disks to be joined in order to be viewed as a single logical block device [4]. Chen *et al.* describe RAID as a method of combining several disks in order to have larger, single volumes that are presented in a familiar way, improving the overall reliability of the storage, and increasing the overall speed of operations on the volume by exploiting parallel transfers from all disks. They define RAID levels 5 and 6, which are described as obtaining parallelism between disks by splitting data across disks in a block-cyclic manner using a fixed chunk size. They show that transfers significantly larger than this chunk size can exploit multiple disks simultaneously for different parts of the transfer, while small transfers that are not close together can also exercise several disks independently.

Fault tolerance is a necessary part of RAID because combining multiple components increases the average rate of failure of components. For example, RAID 0 is the only RAID level that does not contain redundant information. A RAID 0 array composed of n disks has the Mean Time To Data Loss (MTTDL) $MTTDL = MTTF/n$, where MTTF is the mean time to failure of a single disk. Single disks can have an MTTF on the order of one million hours [18], yielding an expected lifetime of 114 years. However, even a moderately sized RAID 0 array of twenty disks can have a MTTDL of less than an installation's desired lifetime.

Most RAID configurations include some level of redundancy, including RAID 5 and RAID 6, which contain parity. These parity-based levels generally implement systems of $k + m$ disks where any m disks may fail without losing data. RAID 5 and 6 have fixed values $m = 1$ and $m = 2$, respectively. Reliability is incorporated into the array by interleaving m chunks of parity for every the k chunks of data, spreading them evenly over all $n = k + m$ disks at the same offset to create a stripe. The documents describing RAID [4] specify that the parity is generated such that any k chunks of data and parity are required to recover the k original chunks of data. The means of generation is left up to the implementer, with many specialized erasure codes existing for RAID 6 [5, 14]. Reed-Solomon coding [16] can be used to perform $k + m$ codings for $m \geq 2$.

RAID implementations for high performance workloads are typically implemented in

*University of Alabama at Birmingham, curryml@cis.uab.edu

†Sandia National Laboratories, lee@sandia.gov

‡University of Alabama at Birmingham, tony@cis.uab.edu

§Sandia National Laboratories, rbbright@sandia.gov

hardware, because hardware has historically provided much higher performance in many situations. RAID performs erasure correcting coding on all data written [4], necessitating a high-speed implementation of erasure correction. This work aims to achieve greatly improved software RAID performance by offloading erasure correction operations onto commodity graphics processing units (GPUs).

The authors have already shown that GPUs are more well-suited to performing Reed-Solomon coding than CPUs [6, 8, 7]. This capability is attributable to a lack of appropriate x86-64 instructions for performing Reed-Solomon coding [3]. While GPUs also lack these instructions, the memory architecture and the large number of cores both contribute to much higher rates of coding and decoding. This work integrates the Gibraltar library [7], a GPU-based Reed-Solomon coding library, into a system called Gibraltar RAID.

2. Motivation. This work occupies an interesting space between hardware- and software-based RAID controllers, inheriting attributes from each. The overall goal is to provide increased flexibility over hardware-based RAID implementations, and increased speed over software-based implementations. This section details the benefits of a high-performance software RAID implementation over hardware RAID.

2.1. Lightweight. One reason for the expense of high-performance RAID controllers is the lack of HPC-specific controllers in the marketplace. Instead, HPC customers purchase products aimed at the larger enterprise customer base. The enterprise market demands a controller that often has many more features than are strictly needed for HPC, like deduplication [9]. By only implementing the features that are required for high-performance streaming I/O (that is, storing data reliably and serving data to a parallel file system), costs can remain relatively low while servicing HPC (and similar) needs just as effectively.

2.2. Inexpensive. Commodity hardware has impacted HPC for the better, yielding systems that are much more powerful while maintaining cost-effectiveness. For example, today's primary architecture for supercomputers is the cluster of workstations, which are largely created from commodity components. Initial investigations into this architecture were performed to field an economical alternative to expensive, proprietary systems [20]. As the commodity computing market grew, improvements to the processing capacity of home and business computers were leveraged by new clusters, forming more powerful supercomputers. The result is that the largest machines are built with commodity processor technology [2].

GPU vendors have been continuing the commodity HPC trend by opening a general-purpose API to program processors that are usually intended to render graphics for interactive animation [10]. GPUs have mostly been used to improve on CPU performance, with several important applications accelerated significantly with GPUs [11]. Applying GPUs to RAID is unusual because the GPU is intended to replace a non-commodity part – the erasure correction engines used in RAID controllers. The benefits of large production and continual improvements for the graphics market allow this application to benefit from improving GPUs without much effort. This situation is in direct contrast to the improvement of RAID controllers, which usually require redesign to incorporate improvements in technology.

2.3. Extended RAID Features. While a reduced set of features is certainly a benefit, another goal behind this project is to provide exactly the features that are required, including those not offered with less full-featured RAID controllers. Certain applications require high levels of reliability and data integrity, but RAID controllers often do not provide features beyond simple RAID 6. We have integrated two uncommon features into Gibraltar RAID: Extra parity and read verification.

2.3.1. Extra Parity. Recently, there has been much written that indicates how RAID 6, which allows for two disks to fail without data loss, is not going to be sufficient to ensure data reliability in the near future. A specific example questions the validity of hard disk manufacturers' reliability estimates of their own disks [17]. Others also have shown that many hard disk failures are correlated, resulting in an increased probability of array failure once the first disk has failed [13]. Further, the growing discrepancy between disk size and disk speed is increasing the chances of disk failure during array reconstruction once a disk has already failed.

An additional problem is encountering an unrecoverable read error (URE) during reconstruction after redundancy has been eliminated from the array. UREs can occur for many reasons, including media damage, a reduction of magnetic coating on the platters, or component wear, and result in data loss for the affected sector. A RAID array operating without any failed disks can recover from this loss, but an array that lacks redundancy cannot. For example, if a RAID 6 array with two failed disks attempts to rebuild onto clean disks, but receives a URE from one of the remaining disks, the contents of the affected sector are completely lost. If the two corresponding sectors on the failed disks are used for data blocks, then those sectors are not recoverable.

UREs have not historically been a great concern, but the typical hard disk size has increased while the URE rate has not decreased. Given a typical SATA disk with a URE rate of one sector per 10^{14} bits [19], a read error can occur (on average) once every 12.5 TB. Many large RAID arrays are significantly larger than 12.5 TB and are heavily loaded, implying that such arrays will have a high probability of encountering a URE within context of a rebuild, causing data loss.

One mitigating strategy for UREs is scrubbing [12]. During scrubbing of an array, the RAID controller reads all disks from beginning to end, attempting to detect UREs and parity mismatches as a background process during normal operation of a RAID array. When a URE is encountered, the controller recovers the contents of the sector with parity and requests the disk to rewrite the information. The disk will either rewrite the data (if the sector is not damaged) or remap the sector to another within a pool of spare sectors. This process is not possible without redundancy in the array to use for data recovery. Scrubbing is not the perfect solution: It consumes bandwidth, reducing the bandwidth available to client applications; and scrubbing does not work if parity in the array is exhausted and the array is rebuilding, when the array is likely to encounter a URE.

One way to lessen the above problems of UREs, less reliable disks, and batch-correlated failures is to introduce higher degrees of parity. Current hardware RAID controllers are statically configured with industry-standard and vendor-specific RAID levels, and do not allow for arbitrary $k+m$ redundancy within an array. The primary distinguishing feature of Gibraltar RAID is the ability to use the values of m that users require for their applications, constrained only by space utilization and performance requirements. An array implemented with the controller described here can be dynamically tuned to provide the desired amount of parity for each installation to make efficient trade-offs in performance, reliability, capacity, and scale.

2.3.2. Read Verification. In large computer installations, component failure is unavoidable. Some of the failures that occur are not easily diagnosable because of the lack of detection for certain trouble conditions. For example, a faulty disk cable can be responsible for a large amount of silent data corruption between a RAID controller and its disks. However, a controller that does not verify reads is not capable of detecting the problems imposed by a bad cable, creating mysterious data corruption and failures in an application.

Furthermore, increasingly small feature sizes and the resulting highly dense computer installations increase the probability of single events (like an alpha particle or neutron strike)

causing silent corruption of data. While most components can and do include their own error correction between different components to guard against data corruption, missing error correction circuitry (and less robust types) can allow corrupted data to be propagated to a user application.

Gibraltar RAID effectively detects corruption arising from these problems by verifying reads at high speeds. Read verification can be used to effectively recover from single events, detect intermittent or chronic problems with components, and continue operating in spite of less severe problems. Additionally, using read verification can correct UREs without requesting extra data or seriously reducing the speed of operation, as the data required to recover the sector is initially requested for data verification. This reduces the latency between encountering a URE and having available data to recover the associated sector.

High-speed recovery of data lends itself to read verification when some disks within an array have failed and a rebuild has not been completed. Gibraltar RAID supports this by recovering more of the data than is strictly necessary based on the operating mode of an array; specifically, during recovery of a missing chunk, a different chunk in the same stripe that is available on disk is read and also recovered in the same recovery operation that regenerates the missing chunk. The recovered chunk and the read chunk can be compared to ensure correctness. This process does not work without extra redundancy in the array, indicating a need for enough parity to ensure that redundancy is always present.

3. Architectural Description. Because a GPU is used for coding (generating parity) and decoding (recovering lost data), Gibraltar RAID was implemented entirely in user space. Interactions between the RAID array and network clients can be managed through the Linux SCSI Target Framework [1]. The interface between Gibraltar RAID and the target is general, allowing Gibraltar RAID to be used with other network storage packages that have user space operation. Gibraltar RAID cannot take advantage of many facilities for I/O within the operating system kernel, so many portions of the underlying secondary storage stack had to be reimplemented within Gibraltar RAID in user space. This section details the mechanics of the components that have been reimplemented, and how the components communicate.

Figure 3.1 has been provided to aid with the description following. Different data paths are outlined as follows: Writes are denoted with a “w,” reads with an “r,” and victimization with a “v.” Each interaction is noted with a letter indicating the data path and a number indicating the order in which the interactions occur.

3.1. Architectural Overview. There are two main restrictions that govern the overall design of a GPU-based RAID system. First and foremost, an application using the NVIDIA CUDA toolkit must run in user space. No public, documented interfaces to the NVIDIA run-time exist that are available from a driver that runs in kernel space. Given that the iSCSI target used for this project runs in user space, the requirement that the coding occurs in user space presents little logistical difficulty. However, if desired, a small driver that passes block requests to a user space daemon is not difficult to consider. Regardless, GPU operations must be performed in user space. To simplify design and development, and to minimize user mode to kernel mode transitions, the other components are also located in user space.

Second, as a user space service that requires knowledge of the contents of cache in order to verify reads, Gibraltar RAID must bypass the Linux buffer cache when accessing disks. If this is not done, a user application may request data that is still in the buffer cache, and Gibraltar RAID will have no means of knowing whether that data came from the disk or was already in RAM. Parity would then be re-verified, which is expensive compared to simply serving the request from the buffer cache.

It is possible to keep a large cache in user space and still use the buffer cache to schedule I/O to and from disk. However, an application that is required to deal with large amounts of

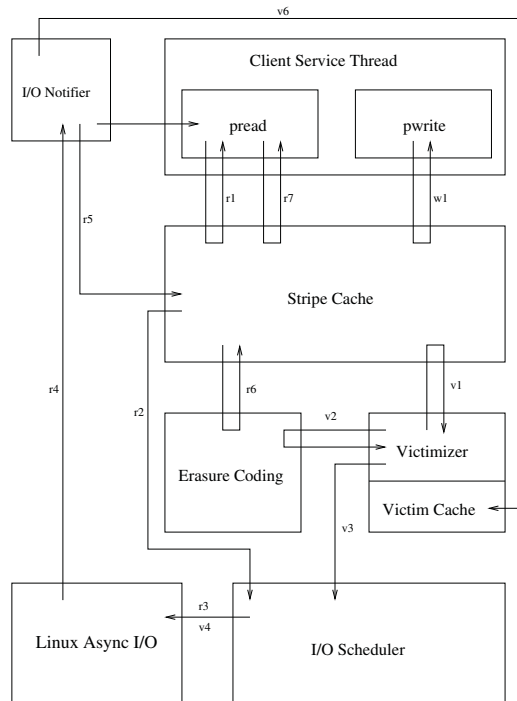


FIG. 3.1. Gibraltar RAID Architecture and Data Flow Diagram

information in its own cache needs a high degree of control over the memory it is using (and memory used by the kernel on the application's behalf) in order to prevent running afoul of Linux's optimistic memory allocation. If the system is low on memory and an application calls `malloc()`, virtual memory can be allocated to the process in absence of physical memory. The assumption is that the memory either will not be completely used, or more memory will become available to support this allocation. For storage applications like RAID, the general pattern is to write data to disk, then acquire more data from users or other input devices. If the buffer cache is using a large amount of memory, as it tends to do, it can interfere with the application's ability to maintain high bandwidth.

The requirement to bypass the Linux buffer cache necessitates using the `O_DIRECT` flag when opening the devices, which allows a user to perform I/O operations directly from memory in user space rather than through the Linux buffer cache. `O_DIRECT` unfortunately conflicts with some CUDA functionality, which includes special memory allocators for optimizing memory transfers between host memory and GPUs. Such allocators perform operations like map host memory into the GPU, which provides increased overlap between PCI-Express transfers and computation within the GPU; and prevent memory regions from being paged out by the virtual memory system, which saves a memory copy when transferring data to a GPU. Direct I/O operations from memory regions allocated with these allocators fail, so these optimizations are unavailable. This decreases the effectiveness of the GPU significantly because of the need for increased host memory copies, but GPU operations still maintain significant speed improvements over a CPU-based software RAID.

3.2. System/RAID Interface. All interaction between the target and Gibraltar RAID occurs through functions implementing the interfaces for the standard C library calls `pread()` and `pwrite()`. Each function takes a pointer to a user buffer, an offset into the RAID device,

and the length of the desired data to be copied into the user buffer. Each request is mapped by these functions to the stripes affected, and asynchronous requests for those stripes are submitted to the Gibraltar RAID cache. These requests are filled without knowledge of whether the stripes are present in the cache, so the stripe can be requested in one of two ways:

- Request stripe with its full, verified contents; or
- Request a “clean” stripe, which may return an uninitialized stripe and does not incur disk I/O.

The second option is useful if the stripe is to be completely overwritten by a write request, or if it is expected to be overwritten. Client service threads, threads created by the target to satisfy read and write requests from network clients, call `pread` and `pwrite`.

For reads, additional stripes may also be requested in order to provide read ahead capability that takes advantage of spatial locality of disk accesses. In the case of streaming reads, there is a high probability that a read for a block of data will be quickly followed by a request for the next block of data. The Linux kernel buffer cache does this; read ahead is necessary to get the best performance out of a storage system under streaming workloads.

After all read or write requests have been registered, the client service thread waits for the requests to be fulfilled. In the case of a read, the routine passes each stripe to the erasure coding component to be verified or recovered. Writes can be recorded as incomplete updates to a stripe, anticipating that the whole stripe will eventually be overwritten. If this does not happen before a stripe is chosen for victimization, the stripe must be read, verified, modified with the contents of the incomplete stripe, updated with regenerated parity, and written.

3.3. Stripe Cache. Gibraltar RAID includes a stripe cache that operates asynchronously with the I/O thread and the user’s requests for reads and writes. In the event of a read request to the cache, the cache submits requests to the I/O thread to read the relevant stripes from disk in their entirety, including parity blocks for read verification. This full-stripe operation can be viewed as a slight mandatory read ahead.

Writes have a simple implementation in the stripe cache, as they do not require immediate disk operations. The cache is optimistic; if the write request does not fill an entire stripe, the cache assumes that the stripe will be completely populated before being flushed to disk. Therefore, no reads are required before a partial update to a stripe is made. If a stripe is in the process of being read, the cache will force the client thread to wait until the read has been completed. If the thread has already been previously read, it will be simply given to the client thread to be updated with the write contents. Otherwise, a blank stripe is returned, and the client is responsible for maintaining the clean/dirty statistics related to the writes requested.

A victimizer thread occasionally (because of an external stimulus from the cache related to memory pressure) deletes clean stripes and flushes dirty stripes. The interface between the victimizer and the cache is flexible, allowing many different types of algorithms to be implemented. The default mode of operation is the Least Recently Used (LRU) caching algorithm, but higher quality algorithms may be used. Furthermore, the victimizer can have an internal timer used to implement write-behind caching.

3.4. I/O Scheduler. The scheduler receives requests from the cache for reading stripes, and receives requests from the victimizer for writing stripes. The scheduler accumulates these requests in a queue until it is ready to service them. All of the requests are received as a batch to facilitate combining of requests that are adjacent on disk. Only write requests are combined for the following reasons:

- Clients are affected by latency by having to wait longer for a read request to be serviced. Reads are necessarily synchronous, so a large combined read request will force all clients to wait until the entire combined request is serviced.

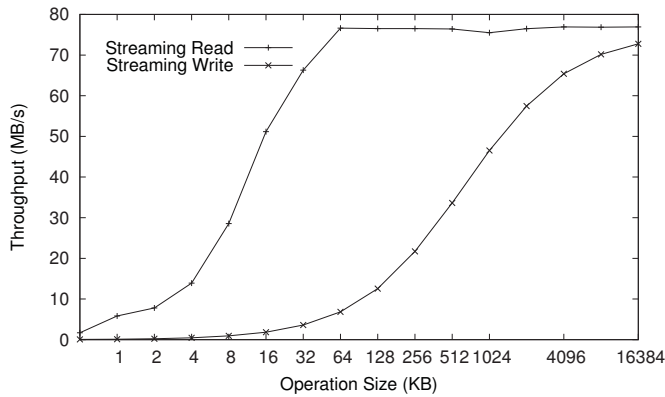


FIG. 3.2. Performance of a single disk in a RS-1600-F4-SBD switched JBOD over 4Gbps Fibre Channel

- Our experiments show that performing short contiguous reads easily obtains good performance from a disk. Writes, however, can be significantly slower (depending on the qualities of the system, such as hardware and configuration) unless a comparatively large amount of data is accumulated for writing at once. Figure 3.2 demonstrates this property. Notice that contiguous writes of 16 megabytes are slightly slower than contiguous reads of 64 kilobytes. This performance degradation is only apparent for files or devices opened with the `O_DIRECT` flag, which bypasses the Linux kernel buffer cache. For normal disk operation, write requests are combined in the buffer cache, hiding this potential performance issue.

The scheduler receives all waiting requests as a batch, ordering and combining them as necessary to achieve the highest possible disk bandwidth. The ordering algorithm currently used is the circular elevator algorithm (C-SCAN).

Combining requests is conceptually simple. If there are two writes that are adjacent on disk, it is possible to use a vector write in order to combine the requests into a single write call. One implementation of this type of call available to applications is `pwritev()`, a vector version of `pwrite()`. Vectored operations allow a contiguous area of a disk to be the target for a combined write operation from non-contiguous areas of memory. However, using `pwritev()` is not the best strategy for a RAID implementation.

Asynchronous I/O, which allows the Linux kernel to manage read and write requests in the background, is more sensible for storage-intensive applications. Initially, this implementation used the `pthread` library to perform synchronous reads and writes with one thread assigned per disk. Switching to asynchronous reads and writes allowed for more efficient use of resources than CPU-intensive `pthread` condition variables with a high thread-to-core ratio allow. While asynchronous I/O has been implemented in the Linux kernel and C libraries for some time, the methods for performing asynchronous vector I/O are not well-documented.

There is a method of using `io_submit()` to submit `iovec` structures in an unconventional and difficult-to-discover way. The typical usage of `io_submit()` takes an array of `iocb` structures as a parameter, which describes individual I/O operations to submit asynchronously. However, to use the relatively new vectored read and write capabilities, one passes an array of `iov` structures within the `iocb` structure instead of `iocb.common` structures. These will be used to perform a vectored I/O operation. This is not noted in system documentation.

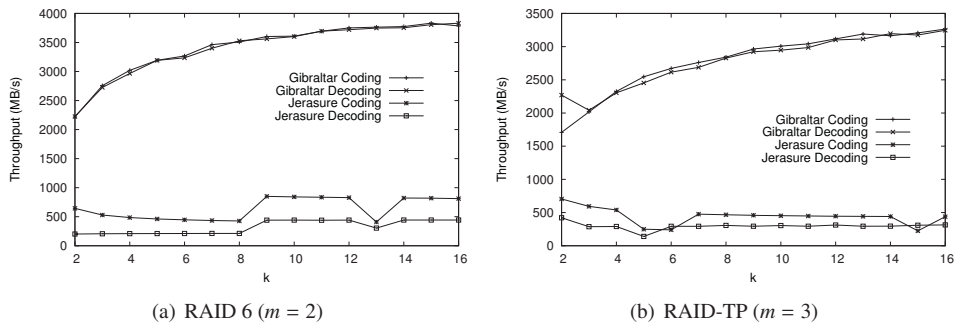


FIG. 3.3. Performance of GPU for Coding, IMB Chunk Size

3.5. I/O Notifier. The I/O Notifier is a thread that collects events resulting from the asynchronous I/O calls, and performs record-keeping on a per-stripe basis. Once all of the asynchronous I/O calls for a stripe have been completed, the notifier notifies other threads that depend on the stripe associated with the I/O. If the stripe is undergoing eviction from the cache, this thread will initiate destruction of the stripe at I/O completion.

3.6. Victim Cache. When considering the speed that new I/O requests can arrive at the RAID controller, there is a significant delay between the decision to victimize a dirty stripe and the completion of the write associated with it. Canceling a write in progress is an inefficient action because of the combining of writes and the asynchronous completion of the writes. To aid in victimization, a victim cache is included that allows for a client read or write request to “rescue” a stripe from being deleted before it has been written, or even while the write is still in progress. Maintaining a separate cache can allow for many flush requests to be in flight simultaneously without overloading the hash table for the main cache. A separate cache also allows fewer look-up operations and, thus, locking, during the victimization of a stripe.

3.7. Erasure Coding Component. The erasure coding component uses a the Gibraltar library, which was designed to perform Reed-Solomon encoding and decoding at high rates using GPUs [7]. Briefly, the Gibraltar library operates by accepting k buffers of data, such as striped data for a RAID array, and returns m buffers of parity. This GPU-based parity calculation can encode and decode at speeds of well over four gigabytes per second for RAID 6 workloads on commodity GPUs. Figure 3.3(a) shows RAID 6 performance over a variety of stripe sizes as compared to a CPU implementation of Reed-Solomon coding, Jerasure [15]. A unique feature of Gibraltar RAID is the ability to extend far beyond RAID 6 in the number of disks that may fail without data loss. Figure 3.3(b) shows performance for RAID TP, a triple-parity RAID level that can tolerate three disk failures. These tests were performed with a GeForce 285 and an Intel Extreme Edition 965. The most compelling feature is that the GeForce 285, though costing approximately 60% less than the Intel processor, demonstrated a significantly increased performance.

GPU parity calculations entail transfer of significant amounts of data across the PCI-Express bus to the GPU. This implies that using the Gibraltar library in this system also incurs significant PCI-Express traffic. This traffic can be a significant concern if other hardware, like network adapters and host bus adapters, also heavily uses the PCI-Express bus.

4. Performance. In order to test the performance of Gibraltar RAID, an experiment has been constructed to measure the speed of streaming raw I/O to a RAID array managed by

Performance			
	Linux md, RAID 6	Gibraltar, RAID 6	Gibraltar, RAID TP
Write	612	637 (min: 438)	727 (min: 589)
Read	878	774	720
Write (sf)	369	669 (min: 470)	743 (min: 438)
Read (sf)	262	795	721
Write (df)	369	602 (min: 501)	759 (min: 462)
Read (df)	253	789	742
Write (tf)	N/A	N/A	649 (min: 489)
Read (tf)	N/A	N/A	767
sf: Single Failure, df: Double Failure, tf: Triple failure			

TABLE 4.1

Performance of Gibraltar RAID compared to Linux md, for streaming operations to raw device. Measured in MB/s, higher is better.

Linux md, the software RAID driver included with the Linux operating system, and Gibraltar RAID. All measurements were taken without use of the iSCSI target, instead targeting the base devices. With both reads and writes, requests are submitted to the array starting with offset zero, with an I/O size of one megabyte (2^{20} bytes). The operations are continued for 100 gigabytes (100×2^{30} bytes). All configurations are tested in normal mode (*i.e.*, no disks failed) and all degraded modes (*i.e.* at least one disk unavailable) supported by the RAID levels tested. Each test has been run three times, and the maximum bandwidths are reported. For cases with high variation, minimum bandwidths are also reported.

Each array was configured to have 16 disks of data capacity, with the addition of the necessary number of disks to support the RAID level implemented (*i.e.*, two extra disks for RAID 6, and three extra disks for RAID TP). A chunk size of 64KB was used. With Linux md, the array was built with the “assume-clean” option because md would otherwise attempt to calculate all parity blocks for the array. While the data in the md array will not be consistent at first, the inconsistency does not cause any difficulty with the tests as executed. Writes are also performed prior to reads, so all data is consistent when read.

Tests for both Gibraltar RAID and md were performed on the same server. It has two quad-core Intel Xeon X5550 CPUs and 24 GB of DDR3 RAM clocked at 1333 MHz. The GPU used is an NVIDIA Tesla C1060. Samsung Spinpoint F1 HE103UJ disks are connected to an LSI SAS1068E disk controller.

Table 4.1 gives the result of the performance testing. One notable feature is that Gibraltar RAID’s read bandwidth is slightly lower than that of md in normal mode. This is expected, as Gibraltar RAID performs read verification with all available parity, which consumes more bandwidth. As the above configuration is bandwidth-limited for reads by the disk controller, the bandwidth consumed by reading parity slightly reduces the performance. However, total bandwidth can be calculated for Gibraltar RAID by dividing its rate by the number of data blocks per stripe and multiplying by the number of total blocks per stripe, yielding comparable total bandwidths (achieving 99.2% and 97.3% of md’s rate for Gibraltar RAID 6 and Gibraltar RAID TP, respectively). This is an important result: Parity verification does not significantly reduce the speed of reads in the above configurations. Only the extra bandwidth requirement for parity reads causes a performance decrease.

One notable trend is that the variability between results in the write test on Gibraltar RAID could be quite high, with up to a 40% difference between runs. Read bandwidth tests had almost no variability. As this is an early prototype that focuses first on functionality over

performance, some variations are to be expected. We estimate that the performance variation results from a potential serialization based on memory usage constraints. While the cache is filling, the amount of free memory available to the cache decreases. When the victimizer is awakened by the cache as it runs out of memory, the nearly full cache will cause the writes accepted into the system to slow to the rate of disk and GPU coding occurring serially, as this is the rate at which memory is freed for new requests. A batch must proceed through the GPU parity generation component and be written out to disk before a new set of writes can be accepted into the memory that has just been released. In many situations, this condition does not occur and writes proceed at the same speed as reads; however, this is not always the case, and the tests reflect this undesired behavior. This can be considered a lesson learned, and an opportunity for future improvement.

Degraded mode performance for Gibraltar RAID is significantly better than `md`, showing no performance reduction over normal mode. In fact, performance typically *improves* in degraded mode because less data is read for parity verification. In comparison, `md` suffers a performance decrease of 70% for reads. Our previous work in developing the Gibraltar library focused on making coding and decoding similar operations, allowing degraded mode to be as fast as normal mode [7].

5. Conclusion. High-performance software RAID with commodity hardware assist has the potential to provide an economical means of high-speed storage for a multitude of uses. Its flexibility provides for increased capabilities that allow it to enable applications that even many hardware RAID controllers are not capable of servicing. For example, environments that are particularly harsh and space-constrained can apply the extra parity and read verification to provide much more reliable storage in the face of difficult conditions.

However, the work presented in the paper does not require extraordinary conditions to show benefits to users. High-performance streaming I/O is a basic requirement of many applications, but typical software RAID and inexpensive hardware RAID controllers may not be capable of supporting that level of performance. Many applications *only* require high-performance streaming I/O and data reliability, but historically only expensive RAID controllers can provide both capabilities. Gibraltar, when paired with an economically priced GPU, can also fill that niche.

This work demonstrates that parity generation and data recovery after disk failure can occur at line speed for many disk installations by using a GPU. Further, as Gibraltar can verify data at high speeds, users are protected from a whole class of errors that cannot be prevented without similar measures in other controllers. As many software RAID packages do not verify reads, the type of technology presented here would be a natural upgrade. This is a significant point because hardware RAID controllers with this capability are much more expensive than the GPU required to perform these computations. Furthermore, hardware RAID controllers that do support read verification are rare.

While this work currently uses GPUs, GPUs are not the only type of device that can be used to calculate parity. As technology evolves, different computation devices will be preferred for computation of parity based on analysis of cost versus performance. The Gibraltar library, the underlying library for Reed-Solomon coding and decoding for Gibraltar RAID, is flexible enough to allow for different (or multiple) devices to be tasked with parity computation. The changing computational landscape, with a multitude of accelerators and CPU types available, will be well-served in the future by a user space RAID infrastructure. We intend to test Gibraltar with a variety of other computation devices and methods.

6. Future Work. It has been shown that GPUs are fully capable of sustaining parity generation and data reconstruction beyond the rates sustainable by the software RAID implementations available for use within this study. However, previously mentioned issues such

as memory fragmentation, pinning/mapping of memory, and direct I/O have made parity operations much slower than they could be. In the future, we plan to investigate methods to solve such memory problems within the controller. Extra computational capacity obtained from such improvements could be used to provide other I/O-related services or service other applications entirely.

While the tests in this paper cover the speed of the raw device, it is important to determine the achievable performance through the means of access. This RAID infrastructure is currently implemented as an easily integrated modification to the Linux Target Framework, which allows network access to storage via iSCSI, iSER, FCoE, and other protocols. The current task with top priority as of this writing is to integrate machines running this software into clusters with high-speed interconnects. We can then perform multi-client tests over a variety of protocols. Further access options that can be provided include a small driver allowing direct access to the array as a block device.

7. Acknowledgements. This work was supported by the United States Department of Energy under Contract DE-AC04-94AL85000. This work was also supported by the National Science Foundation under grant CNS-0821497.

REFERENCES

- [1] *The Linux SCSI Target Framework*. <http://stgt.sourceforge.net/>.
- [2] *Top500 supercomputing sites*. <http://www.top500.org>.
- [3] R. BHASKAR, P. K. DUBEY, V. KUMAR, AND A. RUDRA, *Efficient Galois field arithmetic on SIMD architectures*, in SPAA '03: Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures, New York, NY, USA, 2003, ACM Press, pp. 256–257.
- [4] P. M. CHEN, E. K. LEE, G. A. GIBSON, R. H. KATZ, AND D. A. PATTERSON, *RAID: High-performance, reliable secondary storage*, ACM Computing Surveys, 26 (1994), pp. 145–185.
- [5] P. CORBETT, B. ENGLISH, A. GOEL, T. GRACANAC, S. KLEIMAN, J. LEONG, AND S. SANKAR, *Row-diagonal parity for double disk failure correction*, in In Proceedings of the 3rd USENIX Symposium on File and Storage Technologies (FAST 04), 2004, pp. 1–14.
- [6] M. L. CURRY, A. SKJELLUM, H. WARD, AND R. BRIGHTWELL, *Accelerating Reed-Solomon coding in RAID systems with GPUs*, in IEEE International Symposium on Parallel and Distributed Processing, 2008, April 2008, pp. 1–6.
- [7] M. L. CURRY, A. SKJELLUM, H. L. WARD, AND R. BRIGHTWELL, *Gibraltar: A library for RAID-like Reed-Solomon coding on programmable graphics processors*, (2010). Submitted to *Concurrency and Computation: Practice and Experience*.
- [8] M. L. CURRY, H. L. WARD, A. SKJELLUM, AND R. BRIGHTWELL, *Arbitrary dimension Reed-Solomon coding and decoding for extended RAID on GPUs*, in 3rd Petascale Data Storage Workshop held in conjunction with SC08, November 2008.
- [9] D. GEER, *Reducing the storage burden via data deduplication*, Computer, 41 (2008), pp. 15–17.
- [10] NVIDIA CORPORATION, *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*, Santa Clara, CA, 2007.
- [11] J. OWENS, M. HOUSTON, D. LUEBKE, S. GREEN, J. STONE, AND J. PHILLIPS, *GPU computing*, Proceedings of the IEEE, 96 (2008), pp. 879–899.
- [12] J.-F. PARIS, A. AMER, D. LONG, AND T. SCHWARZ, *Evaluating the impact of irrecoverable read errors on disk array reliability*, nov. 2009, pp. 379–384.
- [13] J.-F. PÂRIS AND D. D. E. LONG, *Using device diversity to protect data against batch-correlated disk failures*, in StorageSS '06: Proceedings of the Second ACM Workshop on Storage Security and Survivability, New York, NY, USA, 2006, ACM Press, pp. 47–52.
- [14] J. S. PLANK, J. LUO, C. D. SCHUMAN, L. XU, AND Z. WILCOX-O'HEARN, *A performance evaluation and examination of open-source erasure coding libraries for storage*, in FAST-2009: 7th Usenix Conference on File and Storage Technologies, February 2009.
- [15] J. S. PLANK, S. SIMMERMAN, AND C. D. SCHUMAN, *Jerasure: A library in C/C++ facilitating erasure coding for storage applications - Version 1.2*, Tech. Rep. CS-08-627, University of Tennessee, August 2008.
- [16] I. S. REED AND G. SOLOMON, *Polynomial codes over certain finite fields*, Journal of the Society for Industrial and Applied Mathematics, 8 (1960), pp. 300–304.

- [17] B. SCHROEDER AND G. A. GIBSON, *Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you?*, in Proceedings of the 5th USENIX Conference on File and Storage Technologies, Berkeley, CA, USA, 2007, USENIX Association, pp. 1–1.
- [18] SEAGATE TECHNOLOGY LLC, *Barracuda ES.2 data sheet*. http://www.seagate.com/docs/pdf/datasheet/disc/ds_barracuda_es_2.pdf, 2008.
- [19] ———, *Barracuda LP SATA datasheet*. <http://www.seagate.com/www/v/index.jsp?name=st32000542as-bcudalpsata2tbhd&vgnextoid=1f70e5daa90b0210VgnVCM1000001a48090aRCRD&locale=en-US#tTabContentSpecifications>, 2010.
- [20] T. STERLING, D. J. BECKER, D. SAVARESE, J. E. DORBAND, U. A. RANAWAKE, AND C. V. PACKER, *Beowulf: A parallel workstation for scientific computation*, in In Proceedings of the 24th International Conference on Parallel Processing, CRC Press, 1995, pp. 11–14.

SUPER-SCALE REAL-TIME NETWORK SIMULATION ON THE CRAY XT

NATHANAEL VAN VORST [†], KEVIN PEDRETTI , AND RON OLDFIELD [‡]

Abstract. The success of the Internet in recent decades is undeniable. However, it is equally clear that the Internet has ossified to the point where even small changes to its structure are very difficult to implement. Many research efforts have been funded in recent years with the purpose of fostering network innovations. GENI is a prime example of the tremendous push of academia and industry for building a comprehensive experimentation platform where structural changes to a future Internet can be tested in large-scale settings. In this report, we present our progress in building a large-scale real-time network simulation testbed on Cray XT supercomputers. Specially, we detail our efforts on improving the scalability of PRIME and creating a testbed environment on the Cray XT architecture. Our testbed promises network researchers an environment capable of running experiments at orders of magnitude larger than any current testbed.

1. Introduction. The success of the Internet depends on continuing innovations in the networking research community and on the successful transition of laboratory research projects into products and services. However, the Internet is an operational environment with distributed ownership. This makes the Internet a difficult place to effectively prototype, deploy, and evaluate new designs or services [14]. As such, experimental testbeds are indispensable tool for network researchers.

We can generally classify available experimental networking research testbeds into physical, emulation, and simulation testbeds. While physical and emulation testbeds provide the ability to conduct extensive live experiments, they are not completely satisfactory on all accounts. The performance and scale is inherently bound to the physical resources. For example, PlanetLab currently supports thousands of experiments; the facility is heavily loaded and is already stretching the capability of its underlying resources [16]. Another potential issue is the lack of controllability. Although emulation allows flexible network experiments (e.g., [2]), emulated network conditions are subject to the physical setup of the testbeds (e.g., nodal processing speed, link bandwidth and latency, and traffic situation).

Network simulation tools, in contrast, allow examining large-scale phenomena more expediently. However, simulation fares poorly in other aspects, particularly in its operational realism. Additionally, model development is labor intensive and error-prone. Reproducing realistic network topologies, representative network traffic, and diverse operational conditions in simulation is known to be a substantial undertaking [6]. Despite this, network simulation is effective at capturing high-level design issues and providing preliminary analysis of complex system behaviors.

Emulab [19] and GENI [8] are currently the state of the art in emulation network testbeds. These testbeds are designed as shared resources where many experiments are concurrently run. Users of these testbeds can only expect to gain access to a small fraction of the resources at any given time because of this multiplexing. Most experiments are of moderate size consisting of dozens or maybe hundreds of compute nodes in a single experiment.

In this work we aim to create an experimental testbed that combines the advantages of both simulation and emulation which is capable of utilizing the resources of the Cray XT architecture. Systems such as the Jaguar supercomputer at ORNL and the RedStorm supercomputer at SNL offer environments with tens of thousands of compute nodes connected by high speed interconnects. Furthermore, these systems offer far more computing power per compute node when compared with the state of the art clusters mentioned above. We have chosen to use the Parallel Real-Time Immersive Modeling Environment [1](*PRIME*) as our

[†]Florida International University, nvanv001@cis.fiu.edu

[‡]Sandia National Laboratories, {ktpedre,raoldfi}@sandia.gov

real-time network simulator. PRIME is used because of its ability to execute network models in parallel on both distributed and shared memory architectures.

Studying computer networks and network applications at large-scales is critical to the design and implementation of next generation Internet technologies. The combination of operating system virtualization, network simulation, and emulation technologies on Cray XT supercomputing hardware opens the possibility of executing networking experiments that comprise hundreds of thousands of network hosts and routers. This allows the study networking at scale and in a fully controlled environment.

Our contributions can be summarized as follows:

- We identify three areas that can limit the scalability of PRIME on HPC systems and present solutions for two of these issues.
- We present a framework for running a real-time simulation testbed on the Cray XT architecture.

The remainder of this report is structured as follows. In section 2 we describe the concept of real-time network simulation and the work required to make it scale to thousands of compute nodes. In section 3 we discuss issues we encountered and progress that has been made in running OpenVZ on the Cray XT architecture. Lastly, in section 4 we summarize some our work and detail our on-going efforts.

2. Addressing Issues of Scalability in Real-Time Network Simulation. Real-Time network simulation allows unmodified applications to interact with simulated models in real-time. We use the Parallel Real-Time Immersive Modeling Environment [1](*PRIME*) as our real-time network simulator. PRIME is built on top of the Scalable Simulation Framework [5](*SSF*); a standard API for discrete event simulators. PRIME is both multi-threaded and distributed. Pthreads are used to run PRIME as a multi-threaded application on shared-memory multi-processors and MPI is used to run PRIME on distributed-memory machines. The Cray XT architecture is constructed of many compute nodes, each with one or more multi-core CPUs connected by a high speed interconnect. On systems such as this, PRIME runs as a multi-threaded application on each compute-node. In remote compute nodes, it uses MPI to coordinate the simulation among the individual simulation instances.

To run an experiment using PRIME, the experimenter must develop a network model. The network model is then partitioned into N partitions, where N is the number of compute nodes on which the experiment will run. An instance of PRIME will run on each compute node and execute the portion of the network model described in its associated partition. In addition to executing its portion of the network model, each simulation instance is responsible for importing and exporting network traffic with *emulated* applications running on the virtual hosts or routers. Emulated applications run unmodified on the compute nodes. Traffic generated by the emulated applications is captured by the simulator instance and is injected into the virtual network where the traffic is subjected to the conditions of the virtual network. Likewise, when traffic in the virtual network is targeted at an emulated application, the simulator exports and delivers the packet.

Each compute node may run many emulated applications. The primary reason each compute node will run multiple emulated applications is that the network model is typically so large that even on a very large machine, the number of emulated applications exceeds the number of compute nodes. Each application expects to be run in a standard, unshared environment. PRIME opts to use operating system virtualization techniques to shield the applications from the fact that many emulated applications are multiplexed onto one compute node. Virtual nodes that are running emulated applications will be assigned their own virtual environments. Because many virtual nodes may be assigned to one compute node, a lightweight virtualization platform must be used. For this project we have chosen OpenVZ [13] as the

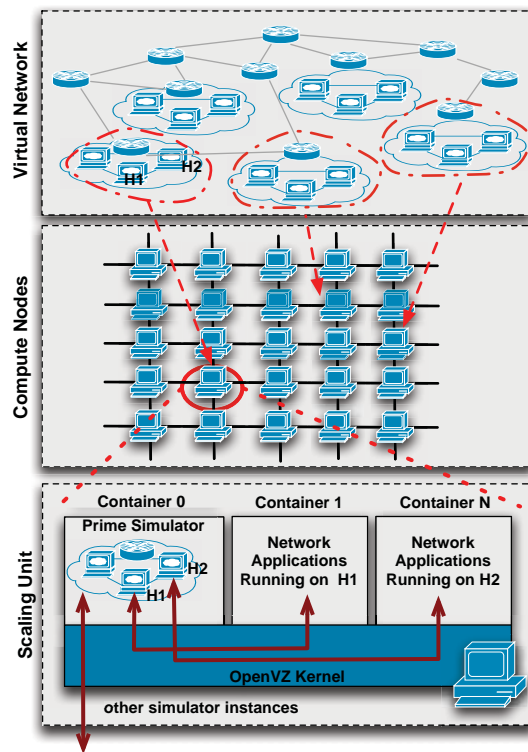


FIG. 2.1. The PRIME architecture. A virtual network is mapped onto a grid of scaling units where each unit simulates a portion of the virtual network. Hosts H1 & H2 are running emulated applications that are run in separate OpenVZ containers.

platform. OpenVZ uses *containers* to virtualize system resources. A container is an isolated environment in which applications can run as if they were running on a dedicated system. Each container has its own set of processes, file systems, users (including root), network interfaces and routing tables. Hardware can either be virtualized and shared between containers, or a single container can be given exclusive access. An OpenVZ kernel hosts multiple containers at once and provides a mechanism to control how much CPU, RAM, and disk space each container is allowed to consume.

The high level real-time simulation infrastructure for PRIME is shown in Figure 2.1. The infrastructure is composed of *scaling units*. A scaling unit is responsible for simulating a portion of the virtual network and managing a set of emulated applications which are co-hosted on the same physical node. All network traffic that is produced by an emulated application is captured by the simulator and injected into the virtual network. Once the traffic is in the virtual network, it will make its way to its final destination which may be another emulated or virtual application. We have identified three areas of concern regarding the scalability of the PRIME network simulator: (1) routing in the virtual network; (2) the performance of transferring packets between the simulator and applications; (3) load balancing of the network model among scaling units.

The remainder of this section is organized as follows. In section 2.1 we discuss our efforts in addressing the amount memory required to route in the virtual network. A brief discussion of our initial efforts at addressing the performance of transferring packets between the simulator and applications is found in section 2.2. Finally in section 2.3 we discuss issues

with load balancing the execution of network models across compute nodes.

2.1. Routing. The current implementation of PRIME requires that a static forwarding table for all hosts or routers in the model exist on each scaling unit. The space complexity of this table is approximately $O(N^2)$ where N is the number of hosts or router nodes in the network model. Lets assume that each scaling unit can manage about 10^2 hosts and/or routers. As the number of scaling units increases, so does the amount of memory required to store the routing tables on each scaling unit. Assuming each scaling unit can manage about 10^2 hosts and/or routers, we find that when there are 10^4 scaling units the entire model consists of about 10^6 nodes. We also see that there would be 10^{12} routing entries per scaling unit. Further assuming each routing entry took just a single byte, it would still take over 931 gigabytes per scaling unit just to store routing information for the model.

There were two major problems with the previous structure of routing tables in PRIME. As mentioned previously, the first problem is that even though a network model may be partitioned across many compute nodes, each compute node must have access to the global forwarding table (which stores the routing information). For a shared memory machine this issue is moot since one forwarding table could be shared among all of the compute nodes. However, for a distributed memory machine the issue is quite severe. The second problem is the space complexity of the forwarding tables was $O(N^2)$. We have devised a new strategy for routing within network simulations called *spherical routing*. Spherical routing addresses both of these issues.

Spherical routing pre-calculates the forwarding tables and conducts routing within so-called *routing spheres*, each with a user-specified routing strategy. A routing sphere is defined internally as a network graph - with vertexes and edges - on which a single routing strategy is applied. A routing strategy can be either based on shortest paths, which is commonly used for intra-domain routing (such as OSPF and RIP), or based on routing policies (such as those used in BGP for inter-domain routing). Using spherical routing, a network can be viewed as a hierarchy of routing spheres: a routing sphere of a sub-network is enclosed by the routing sphere of its parent network; the routing sphere of the sub-network is represented as a vertex in the network graph of the parent network's routing sphere. Within each sphere a static forwarding table is calculated according to its routing strategy before simulation, and during simulation the simulator conducts packet forwarding according to the forwarding table and the location of the routing sphere with respect to its parent network's routing sphere.

The structure of spherical routing can be explained using the following example. The user specifies a network model in a hierarchical fashion, where networks serve as containers for routers, hosts, links and sub-networks. Fig. 2.2(a) shows a network consisting of two sub-networks with the same network structure (Net1 and Net2), connected by three links (L1, L2 and L3). Because Net1 and Net2 have an identical structure the simulator simply marks Net2 as a replica of Net1 in its internal tree representation of the network, as shown on the right of Figure 2.2(b). Net1 consists of two sub-networks (Net3 and Net4), one router (R1), and two links (L4 and L5). Again, Net4 is simply a replica of Net3. The latter consists of four identical hosts (H1 to H4), one router (R3), and four links (L6 to L9). H2, H3 and H4 are replicas of H1. Hosts and routers also serve as containers for network interface cards (NICs); we ignore them in the figure for simplicity.

For each network in the model the user can assign a routing sphere. A sub-network can also inherit the routing sphere of its parent network. For network entities within a network, the routing sphere is called the *owning routing sphere*. Each routing sphere must specify a routing strategy; it's either based on shortest paths or routing policies. In the example, all routing sphere are assigned a shortest-path routing policy. Net1 and Net2 are not specified with routing spheres and their network graphs are flattened into the network graph of their owning

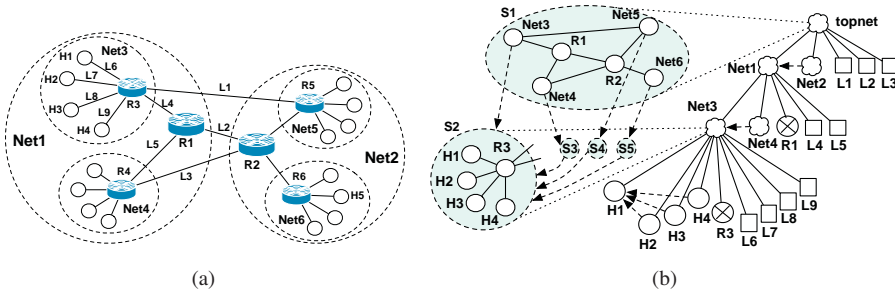


FIG. 2.2. An example network is shown in (a) and (b) shows the internal representation of the same model with routing spheres and replicated sub-structures.

routing sphere (topnet). Net3 through Net6 each has its own routing sphere specified for shortest-path routing, although the routing spheres for Net4 through Net6 are actually replicas of Net3.

If a sub-network is assigned a routing sphere, the sub-network is represented as a “super-node” in the network graph of the routing sphere for the parent network. The routing sphere for the parent network is called a *parent routing sphere*; the routing sphere for the sub-network is called a *child routing sphere*. Like routers, a super-node can have multiple network interfaces. Unlike routers, the distance between the interfaces on the same super-node is not always zero, since the interfaces may actually belong to different hosts or routers. In this case, the distance should be calculated based on the network topology and the associated routing strategy of the child routing sphere. The example defines five routing spheres, S1 through S5, as shown in the left of Figure 2.2(b). Because of replication, there are actually only two types of routing spheres, topnet and S2; S2 through S5 are replicas of each other.

Due to space considerations we omit the details of our algorithm to route packets across and between spheres at run-time. A detailed study of spherical routing is presented in [17]. The important point in terms of this report is that once a packet is within a routing sphere, the simulator is able to route the packet across the sphere using only information from its forwarding tables and the table of its parent. This means that we do not need to include all forwarding tables in all compute nodes (as was our previous approach). For example if we had three compute nodes, we would partition the model into three partitions: P1, P2, and P3. Assume that P1 includes Net3 and R1, P2 includes Net5 and R2 and P3 includes Net4 and Net6. All of the partitions would need to include S1 because our forwarding algorithm requires that routing sphere be able to query its parent sphere. P1 would need to include sphere S2, and P2 would need to include S4. P3 would need to include spheres S3 and S5, but since S3 and S5 are replicas of each other we need only include one of them.

Spherical routing addresses our first issue by breaking the single large forwarding table into a hierarchy of smaller forwarding tables. The space complexity of the forwarding tables is addressed in two ways. First, by breaking up the forwarding table into a hierarchy, each table in the hierarchy is much smaller than the single table. Second, replicated routing spheres do not need to store their copy of their forwarding table because they can share the forwarding table of the routing sphere which they replicate.

There has been much research into reducing the space complexity of routing information for network simulation. Of the previous approaches, hierarchical routing [9] is the approach which is most similar to spherical routing. Suppose a network topology consists of a number of domains, a few clusters within each domain, and a few nodes in each cluster. Each node only maintains routing information for nodes within its local cluster, to each other cluster

in the same domain, and to the other domains in the topology. By storing the information in this way the average space complexity is $O(kN \log_k N)$, where k is the number of clusters per domain, and $\log_k N$ is the height of the hierarchy. This provides much better scalability than a flat forwarding table, however some routes will be in-accurate in comparison to global shortest path. These in-accuracies stem from the fact that each cluster or domain can only have one link that carries traffic between other clusters or domains. Spherical routing has two distinct advantages over hierarchical routing. First, spherical routing considers all links between clusters and domains which allows for a better approximation of global shortest path routing. However, spherical routing does introduce some minimal path inflation when routing across multiple spheres. Second, spherical routing can take advantage of structural similarities to reduce the space complexity of the forwarding tables far more than hierarchical routing alone.

To show the overall effectiveness of spherical routing we extended one of the tier-1 ISP topologies provided by Rocketfuel [15] to build a moderately large network model. The chosen ISP contains 640 routers and 1,382 links. We use a graph partitioner to partition the ISP network into 16 clusters of similar size. Each cluster is assigned a routing sphere. We then attach 192 campus networks [12] (each having 17 routing spheres) evenly distributed among the clusters, resulting in a network model with a total of 103,935 hosts and routers in 289 routing spheres organized in 5 levels. For spherical routing, the total memory consumed by the forwarding tables measures about 7.3 MB. Our previous method for routing would have required approximately 356 GB to store the forwarding tables. These numbers show the promise our spherical routing strategy.

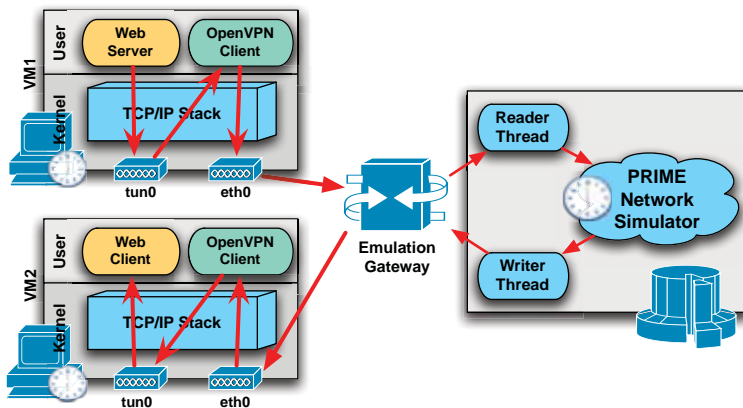


FIG. 2.3. The current PRIME I/O infrastructure. Emulated application traffic is captured by TUN/TAP devices on host machine and sent to the simulator via the emulation gateway.

2.2. Packet Transfer Performance. The second area of concern is the speed at which the simulator can exchange packets with the emulated applications. The current PRIME I/O infrastructure, seen in Figure 2.3, uses an emulation gateway to exchange packets with emulated applications. The approach has the advantage that the simulation of the virtual network and the execution of the emulated applications are fully decoupled. However, the overhead required to exchange packets between the simulator and the emulated applications is quite high. This limits the bandwidth to about a 100MB/s [11].

To cope with this, we designed a new API for PRIME which can support a diverse set of emulation strategies. The new API introduces the concept of an *emulation device* which

abstracts how PRIME imports and exports packets. There are two general categories of emulation devices. The first type of device is an interrupt driven device. These devices require a separate thread and block until a packet is ready to be imported or exported. The second type of device is a polling device. Polling devices do not require a separate thread for execution, instead, the device directly schedules when it should be allowed to import and/or export packets. The scheduling is most often periodic, but our API does not require it.

The emulation device abstraction allows us to explore new methods of exchanging import and exporting packets. Using our new API we wrote an emulation device which uses an emulation gateway to exchange packets with emulation applications. We are currently implementing an efficient transport which directly exchanges packets with emulated applications without the need for a gateway.

2.3. Load Balancing. The third issue is related to the partitioning of the network model. Currently, the model is partitioned into N partitions and each partition is assigned to run on a single compute node (scaling unit). When partitioning the model we opt to use Metis [10] to partition the graph. During partitioning we maximize the delay of the links that cross between the partitions while trying to balance the number of nodes in each partition. This approach has many advantages, however, it fails in a number of significant ways. First, it assumes that all of the hosts and routers in the model take a similar amount of effort to simulate. This assumption is clearly not true. The amount of work required to simulate a partition is not only related to the number of entities in the partition but also the amount of traffic that is generated within partition. A small portion of an experiment's partitions could easily generate a majority of its traffic, thereby causing a severe load imbalance. Because each compute node has a limited amount of resources, the load imbalance may cause a severe over subscription of the resources on some of the compute nodes. This hinders PRIME's ability to simulate the network in real-time. Second, the partitioning does not account for the difference between purely simulated network entities and emulated entities. Emulated entities require more CPU resources because an entire virtual environment must be created and run to host the emulated applications. Furthermore, there are limits as to the number of emulated entities that can exist on a single compute node. Due to the limited time for this project, we opted to not to address this issue. We only mention it as an open issue that needs resolving.

3. A Testbed for Large-Scale Real-Time Simulation on Cray XT. The Cray XT architecture is a platform used by many DOE laboratories (e.g. SNL, ORNL, ANL) to run large-scale parallel distributed simulations. The PRIME Real-Time simulation infrastructure in Figure 2.1 requires two things: (1) a collection of compute nodes that are connected by a high speed network and (2) a virtualized environment to run applications. The Cray XT is a very good match in terms of the first requirement. The second requirement is where the Cray XT is not yet adequate. Virtualization is needed for two reasons. First, emulated applications need to be run in a "standard" environment and should require no modifications to run. Second, in order to support the execution of network models where there are more applications than compute nodes, it must be possible to run multiple emulated applications on a each compute node. Virtualization is clean solution that allows us to run multiple applications in isolated environments on the same hardware.

The default operating system for the compute nodes of Cray XT supercomputers is a modified version of the SUSE SLES distribution called *Compute Node Linux* [18](CNL). CNL has many advantages over a vanilla Linux distribution but there are a few serious drawbacks. First, CNL has been heavily modified and is difficult to compile or modify. Second, because of the propriety extensions introduced by Cray, CNL is essentially a closed source environment. These draw backs have lead to the development of a patch for vanilla Linux kernels to run on the Cray XT hardware. The patch is primarily composed of an open source

driver for the Cray SeaStar and Resiliency Control Agent (RCA) hardware. The RCA is responsible for routing hardware events and console output to the login node(s). The SeaStar driver emulates Ethernet over the SeaStar interconnect. Details of the Ethernet emulation are discussed in section 3.1.

The basic idea behind OpenVZ is to support separate process name spaces and resource isolation and control. OpenVZ is available as a patch to vanilla Linux kernels. The latest patch is for the 2.6.32 kernel and is quite large, currently 132,710 lines. Rather than back-porting the OpenVZ patch to match the our Cray XT patch, we ported the Cray patch to the 2.6.32 kernel. The port of the Cray XT patch was straightforward. Modifying the OpenVZ patch to work with our Cray XT patch was also fairly straight forward. We did however run into difficulties with a number of bugs in the OpenVZ kernel which were exposed by not using the default set of kernel options.

Modern Linux distributions often require hundreds of megabytes of disk space for even the most bare-bones of installs. Compute nodes are run disk-less. Their root file system is an in-memory file system, a so called *initramfs*. In order to minimize the amount of memory that is used for the *initramfs* we opted to build our own distribution of Linux based on BusyBox [4]. BusyBox is a minimal Linux distribution that is commonly used for embedded Linux devices. BusyBox provides many of the system tools one would expect in a full-fledged distribution but there are some exceptions. It is, for instance, expected the applications be statically compiled as the distribution does not come with any pre-compiled dynamic libraries.

In addition to the kernel patch, OpenVZ requires a set of tools to create and manage containers. The OpenVZ tool set is a combination of c-programs and shell scripts which coordinate construction and control of containers. The shell scripts make extensive use of system utilities such as *awk*, *grep*, and *df*. BusyBox provides all of these tools as a single static binary and wraps the binary with scripts to make it appear as if there were distinct applications. It turns out that the BusyBox implementation of many of the system utilities is either incomplete or not strictly compatible with the standard (or common) implementations. We were forced to rework most of the scripts related to the construction of containers in response to these incompatibilities.

OpenVZ has a concept called *templates*. A template is a base install of a specific Linux flavor, including all of the system files for that distribution. There are templates for Fedora, RedHat, Ubuntu, Suse, CentOS, as we well as many others. When a new container is installed, the system basically copies the files from a template into a directory for that container. For each container that is installed, an additional copy of these files must be made. The average size of a template is around 150 megabytes. It is not feasible to store files for hundreds containers in the *initramfs*; there would very little memory remaining to run the simulator or applications. Instead we opted to use a union file system which allows us to store a single read-only copy of the system files and separately store modifications each container makes to the read-only files. However, the union file system alone is not adequate. The whole point of having containers is so applications can be run in the containers during an experiment. The reason to run experiments is to gather data. Applications in each container will generate data of possibly significant volumes. We needed somewhere to store the experimental data generated by the simulator and applications. This led us to use a shared file system to store the experimental data. Section 3.2 details the structure of the file system for each compute node.

3.1. Networking. The SeaStar is basically a small switch with seven ports. One port is connected to a host CPU. The other six ports are connected to SeaStar chips on other compute nodes. In general, the compute nodes are connected in a 3D torus. Each SeaStar in the torus is loaded with a routing table. When a message is received from a neighboring SeaStar, a

check is made to see if the message is destined for the connected host. If the local host is not the target, another table look up is done to ascertain out of which port to forward the message. If the message is meant for the attached CPU, the SeaStar raises an interrupt to the CPU and sets up a DMA transfer to the host's memory.

SNL's open source SeaStar Ethernet emulation driver allows the standard Linux TCP/IP stack to use the SeaStar as it would any Ethernet device. In essence, fully formed Ethernet frames are given to the driver to be delivered. The driver re-writes the Ethernet header with a SeaStar header by mapping the MAC addresses to SeaStar addresses. It is expected that the MAC address of a SeaStar encodes the underlying SeaStar address. Additionally, the driver does not support Ethernet broadcast. As such, ARP cannot be implemented. Instead of running ARP, the ARP tables are preloaded at boot with all of the IPs and MACs of the SeaStars in the torus. Section 3.3 discusses some of the details of the IPs, MACs, and ARP tables are configured at boot time.

The SeaStar supports message sizes that are much larger than the standard Ethernet frame size of 1500 bytes. The default MTU for the driver is 16000 bytes (16KB). Because the SeaStars raises interrupts for each message - which in this case maps to a frame - the frame size can have an impact of the performance of the driver. Using the default driver, we were able to get about 1.86 GB/s between two nodes using iperf's TCP bandwidth test. A previous study measured the peak bandwidth of the SeaStar using portals at 2.2 GB/s [3]. Given the overheads of the TCP/IP stack we are pleased with the available bandwidth.

3.2. File System. The root file system of the compute nodes is an in-memory file system (*initramfs*) because the compute nodes do not have attached storage devices. It is therefore important to keep the initram small because any memory used for the initram is not available for applications.

As mentioned previously, the storage required for the OpenVZ containers can be substantial. To minimize the amount of space required for the containers we decided that we would only support a single type of container. Since all containers will use the same template image, a majority of the files will be identical. If we are able to keep a single copy of all of the files that are the same between all the containers we can save a substantial amount of space. We use funionfs [7], a FUSE based union file system. Funionfs allows for two directories to be "unioned" and presented as a single, mountable, file system. One directory is set to be the read-only and the other is set to be the read-write. When a file is accessed, if the file exists in the read-write directory the read-write version is used. If the file is not in the read-write directory, the read-only version is returned. When read-only files are modified, the modified version is saved in the read-write directory. By using funionfs we are able to store a single template for all containers, and only store the system files which are modified during the execution of each of the individual containers. As part of our reworking of the vzctl scripts we modified them to use funionfs when creating and starting containers.

Funionfs does not entirely solve the storage problem. Files that are unique to a container must be stored separately in the read-write directory. As mentioned earlier, it is expected that applications running within a container will generate data. The data should be available to the experimenters after an experiment so the data must also be persistent. Typically a parallel file system such as Lustre or Panasas is used. We have three target environments on which to run our system. The first system, *xtp*, is an XT5 with around 1024 compute nodes and uses a Panasas based shared file system. The second system, *rsopen* is a much smaller 32 node XT4 running a Lustre based shared file system. The end goal is to run PRIME on a very large XT system such as Jaguar. Jaguar is an XT5 with around 18K compute nodes running a Lustre based file system. There are known issues with using simpler shared file systems such as NFS so our initial goal was to leverage one of these parallel file systems.

It turns out that our ambition was greater than our ability. All of the systems have highly customized installations of either Panasas or Lustre. There was no obvious path to supporting Panasas or Lustre in one - much less all - of our target environments. However, we are able to easily re-export a Panasas or Lustre file system using NFS. The issue is that a single NFS server will be hard-pressed to handle hundreds, and certainly not thousands of clients. To solve that issue we decided we could use a network of NFS servers. The exact number of NFS servers will depend on the particular environment. We plan on running tests on our xtp system to determine how many clients an NFS server can reasonably support.

3.3. Initialization. Compute nodes use a customized initialization process during boot. Our customized initialization script sets up the SeaStar device, populates the ARP tables, mounts shared file systems, and prepares the system to create and run OpenVZ containers. Each compute node is assigned a node id (*nid*). The previously mentioned RCA driver exports the *nid* via the `proc` file system at `/proc/cray_xt/nid`. In addition to the *nid*, there are two static files: `/etc/nids` and `/etc/servers`. The *nids* file contains the IP and MAC addresses of all compute nodes in the system and is used to populate the ARP cache. The local SeaStar IP is calculated as follows: $IP = 192.168.(NID/254).(1 + NID \bmod 254)$. The MAC address is simply the NID (in hex) followed by four zeros and prepended with enough zeros to make a valid MAC address.

After the network is setup, the *prime-ovz* export is mounted from the NFS server specified in the *servers* file. The *prime-ovz* share is expected to have three directories: `template`, `nodes`, and `exp`. The base template images are stored in the `template` directory. Within the `nodes` directory each compute node creates `node_NID` where the compute node stores all its experimental data. The `exp` directory is where users place their network models and where PRIME will store data generated by the simulator itself.

4. Conclusion. Over the course of this project we: (1) successfully ported and ran OpenVZ on a Cray XT4's compute nodes; (2) designed an emulation infrastructure for PRIME and started an emulation driver for the Cray XT; and (3) implemented and evaluated a new routing strategy for large-scale network simulation. In the coming months, we hope to finish the implementation of our testbed at which time we plan to design and run a large-scale network experiment on the Cray XT5 machine at SNL (xtp).

5. Acknowledgments. We would like to extend a special thanks to Kevin Pedretti for all of his help this on this project. Even though he was working on numerous other project of his own this summer, he made the time to help us get our project off the ground. Without his help this would not have been possible.

REFERENCES

- [1] *The PRIME research*. <http://primessf.net/>.
- [2] A. BAVIER, N. FEAMSTER, M. HUANG, L. PETERSON, AND J. REXFORD, *In VINI veritas: realistic and controlled network experimentation*, SIGCOMM'06.
- [3] R. BRIGHTWELL, K. PEDRETTI, AND K. D. UNDERWOOD, *Initial performance evaluation of the cray seastar interconnect*, in In Proceedings of the 13th IEEE Symposium on High-Performance Interconnects, 2005.
- [4] *BusyBox*. <http://busybox.net/>.
- [5] J. COWIE, D. NICOL, AND A. OGIELSKI, *Modeling the global Internet*, Computing in Science and Engineering, 1 (1999), pp. 42–50.
- [6] S. FLOYD AND V. PAXSON, *Difficulties in simulating the Internet*, TON, 9 (2001), pp. 392–403.
- [7] *funionfs*. <http://funionfs.apioi.org/>.
- [8] *Global Environment for Network Innovations*. <http://groups.geni.net/>.
- [9] X. W. HUANG AND J. HEIDEMANN, *Minimizing routing state for light-weight network simulation*, 9th International Symposium on Modeling, Analysis and Simulation on Computer and Telecommunication Systems (MASCOTS), (2001), pp. 108–116.

- [10] G. KARYPIS AND V. KUMAR, *Metis - unstructured graph partitioning and sparse matrix ordering system, version 2.0*, tech. rep., 1995.
- [11] J. LIU, S. MANN, N. V. VORST, AND K. HELLMAN, *An open and scalable emulation infrastructure for large-scale real-time network simulations*, in Proceedings of IEEE INFOCOM MiniSymposium, 2007, pp. 2476–2480.
- [12] D. NICOL, *DARPA NMS baseline network topology*. <http://www.ssfn.net.org/Exchange/gallery/baseline/>.
- [13] *OpenVZ*. <http://openvz.org/>.
- [14] L. PETERSON, T. ANDERSON, D. CULLER, AND T. ROSCOE, *A blueprint for introducing disruptive technology into the Internet*, HotNets-I, (2002).
- [15] N. SPRING, R. MAHAJAN, D. WETHERALL, AND T. ANDERSON, *Measuring ISP topologies with Rocketfuel*, IEEE/ACM Transactions on Networking, 12 (2004), pp. 2–16.
- [16] N. SPRING, L. PETERSON, A. BAVIER, AND V. PAI, *Using PlanetLab for network research: myths, realities, and best practices*, Operating Systems Review, 40 (2006).
- [17] N. V. VORST, T. LI, AND J. LIU, *[SUBMITTED]How Low Can You Go? Spherical Routing for Scalable Network Simulations*, in Proceedings of IEEE INFOCOM, 2011.
- [18] D. WALLACE, *Compute node linux: Overview, progress to date & roadmap*, CUG’07.
- [19] B. WHITE, J. LEPREAU, L. STOLLER, R. RICCI, S. GURUPRASAD, M. NEWBOLD, M. HIBLER, C. BARB, AND A. JOGLEKAR, *An integrated experimental environment for distributed systems and networks*, in Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI’02), December 2002, pp. 255–270.

APPLICATION SUPPORT FOR RESILIENCE IN EXASCALE SYSTEMS

MARIA RUIZ VARELA*, KURT B. FERREIRA†, AND ROLF RIESEN‡

Abstract. Application-driven, coordinated, checkpoint-to-disk is the most prevalent fault tolerance method used in modern, large scale, high performance computing (HPC) systems. As the node count increases, the inherent synchronization of coordinated checkpoint/restart (CR) is expected to seriously limit overall application performance to a point where applications would spend most of the time checkpointing instead of executing useful work. As HPC systems grow in size, unscalable resilience methods and increased failure rate will progressively hinder application performance, hence the need for evaluating resilience performance at exascale. Since exascale systems are not available yet, we choose simulation as performance evaluation method to run experiments at the million-core level. Our choice of simulation platform, the Structural Simulation Toolkit (SST), is an event-driven framework that integrates models for different system components, such as processing elements, memory, and the network. To accurately characterize resilience overhead, we compare the performance of several resilience methods by extracting communication patterns of representative scientific applications and code them in a state machine representation suitable for SST. In this work, we propose a new simulation method that will enable us to identify the most efficient resilience mechanisms for future exascale systems and their applications.

1. Motivation. Coordinated, application-directed, checkpoint-to-disk, is widely used as resilience method in modern parallel systems to enable applications to progress in spite of hardware and software faults. Future exascale machines are predicted to have hundreds of thousands of nodes and millions of cores[1], and the number of cores per chip is expected to grow with Moore’s law. Additionally, since the system failure rate grows proportionally to the number of components, exascale machines are expected to experience more failures than current systems. Consequently, as the node count and number of cores per chip increases, the failure rate will be such that applications would spend most of its time checkpointing, restarting from failures, and recomputing lost work, rather than making progress towards the solution. Even under ideal conditions, applications executing on more than 50,000 nodes could spend more than half of their total running time on checkpointing overhead [11].

Currently there is no evidence indicating that mean time to fail (MTTF) of the individual components that comprise a node will improve in the near future. On the contrary, there is evidence that suggests manufacturing processes improvements for future chips will keep failure rates per bit similar to that of current systems [18]. This expected component constant failure rate and the dramatic increase in component count will lead to a much higher system MTTF for next generation systems.

Figure 1.1 depicts the run time of a 168-hour, weak-scaling application executing in a system with a single node mean time between failures (MTBF) of 43,800 hours (5 years), checkpoint time of 5 minutes, restart time of 10 minutes, and optimal checkpoint interval. For node count beyond 50,000 only a small fraction of the total elapsed time corresponds to actual useful computation time. This rapidly increasing overhead is caused by the amount of lost work and the number of restarts the application experiences as the failure rate grows.

In this work, we explain the simulation method we propose to evaluate resilience performance at extreme scale. In order to run the simulation experiments, we need to design the different components that need to be integrated into the event-driven SST framework; namely the network, compute cores, and communication application patterns. The network component is designed to simulate realistic network traffic. The communication pattern component we designed to plug into SST generates network traffic without incurring the processing and memory overhead of executing a full endpoint simulator. The compute core component is

*University of Delaware, mruizv@udel.edu

†Sandia National Laboratories, kbferre@sandia.gov

‡Sandia National Laboratories, rolf@sandia.gov

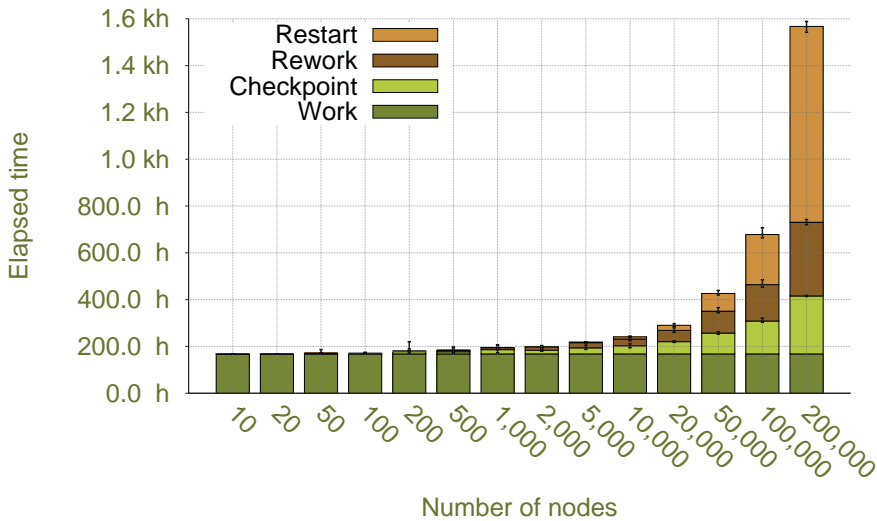


FIG. 1.1. Overhead example for a 168-hour application [11]

represented by a message pattern generator that transmits messages and implement different resilience methods. Message patterns are important because the performance of some resilience mechanisms depend on message exchange behavior. For example, in the case of uncoordinated checkpointing with message logging, only if we know at which point in time messages arrive, how large they are, and which core sent them, will we be able to faithfully emulate message logging, keep track of the log size and logging overhead, and send recovery requests to the appropriate core. Communication patterns are coded as state machine, which is a representation suitable for SST. The state machine for an application pattern will contain additional states to provide different resilience functionality. The rest of the paper is organized as follows. Section 2 provides definitions and a classification of the resilience methods we will compare in our experiments. In Section 3 we describe the simulation approach we propose, the components of the simulation framework, and our choice of design parameters to feed it. In Section 4 we detail the experiment plan, and in Section 5 we provide a current status of our work and conclude.

2. Background. Rollback-recovery fault tolerance protocols can be checkpoint-based or log-based. Based on the type of coordination that occur among processes, checkpoint-based protocols can be coordinated, uncoordinated, or communication induced. Based on the frequency with which the message log is saved, log-based methods can be pessimistic, optimistic, or causal [9].

Several optimizations to coordinated checkpoint/restart (CR), designed to ameliorate some of its negative effects, have been proposed. Some examples include incremental [17] [5], non-blocking [8], diskless [16] [10], and RAID-like distributed and multi-level checkpointing [14].

In coordinated CR all processes save their checkpoint state synchronously creating a consistent global state, which facilitates recovery. Its main drawback is the strain this synchronization puts in the stable storage making storage system bandwidth the bottleneck of the operation. Also, even if only one process fails, the entire application needs to restart from the last stable checkpoint.

In contrast to coordinated CR, uncoordinated CR only restarts the failed processes. Un-

coordinated CR logs messages at the sender side and in case of failure, replays the logged messages necessary to bring the restarted processes to a consistent state with respect to the rest of the processes. Its main disadvantages are the storage space required to log the messages, a considerably more complex restart, the need for garbage collection of message logs as the application execution progresses, and for optimistic protocols, the possibility of roll-back propagation, which could cause all processes to restart in the worst case. Although it has been proposed, uncoordinated CR has not been implemented in HPC systems.

In addition to the classical roll-back recovery protocols, there are mechanisms such as redundant computing that enable applications to continue executing if failures occur. Redundant computing has been long employed in mission-critical [13] [4] [15] and distributed systems [12] [6], but has been dismissed for HPC systems due to its perceived high cost. In Redundant computing each process is replicated a number of times across the system. It increases resilience application by increasing the application mean time between interruptions (MTBI). An application can make uninterrupted progress towards useful work provided that at least one process in the replication pool is functional. Unlike rollback-recovery protocols were a single failure causes an interrupt, a redundant application tolerates multiple failures until redundancy is exhausted, avoiding recovery overhead. This overhead reduction has the potential of increasing the application MTBI and checkpoint interval, reducing time to solution and increasing system throughput. At large node counts, these gains can offset the cost of extra nodes and the additional power and cooling requirements. [11].

Diskless techniques have been proposed as an optimization to coordinated CR to store checkpoints more efficiently [7] [16] [20]. One example are memory distributed mechanisms that save checkpoint state redundantly across a distributed system in a RAID-like manner, and write it to stable storage only when a failure occurs [14]. Its main drawback is the nearly doubled amount of memory required to store both the application state and the checkpoint file. Its main benefit is the ability to tolerate multiple failures as long as at least one of the nodes that store the checkpoint file and its copy remains functional.

Table 2.1 lists the resilience methods to compare as part of the simulation method proposed in this work.

Resilience mechanisms
1. Coordinated CR
2. Uncoordinated CR with optimistic message logging
3. Uncoordinated CR with pessimistic message logging
4. Distributed CR with RAID-like storage in remote node memory
5. Redundant computing

FIG. 2.1. *Resilience mechanisms.*

3. Approach. Our goal is to learn how the various methods that provide application resilience perform at exascale. Since these systems are not available yet, we choose simulation to run experiments at the million-core level. Our choice of simulation platform, the Structural Simulation Toolkit [2], is an event-driven simulation framework that integrates models for different system components, such as processor, memory, and the network. In order to run such large simulations, the network and compute core components integrated into SST need to be simple and frugal in memory consumption.

Concerning resilience methods, some of them might not scale because of the large number of aggregate faults expected to occur in exascale systems; thus, we will simulate faults and force the pattern generators to recover before they can continue. Recovery overhead might become prohibitively large for some resilience methods at exascale. Furthermore, the over-

head might depend on the message pattern, thus be application contingent. Our simulation experiments will enable us to evaluate these scenarios setting various parameters to configure the message pattern generators, that is, the various delays that govern writing and retrieving checkpoint and log data, and the duration of computation phases. We will adjust these parameters to get as close as possible to measurements obtained from modern large-scale systems executing real applications. The following subsections describe our design choices for the different SST components.

3.1. Network. We model the network component as a two-dimensional torus interconnection with end-around connections. A torus interconnect network is a mesh network widely used in HPC systems. In this topology, each processor is connected to four neighbors, which simplifies the implementation of grid application patterns that exchange messages between subgrids. The number of rows and number of columns are powers of two. This design constraint simplifies the collective algorithms implementation. Instead of simulating a router, we opt for a simplified model that forwards messages based on the source route computed at the sending core. Messages are wormhole-routed, i.e. a single message occupies the input and output ports at a time. The routers model internal congestion at the output ports by blocking incoming messages, however, there is no flow control among routers; thus, network congestion cannot be modeled. For many simulation studies, it is important to have realistic network traffic, but computation at the endpoints is of limited relevance. The communication pattern component of SST allows the generation of network traffic without incurring the processing and memory overhead of running a full endpoint simulator.

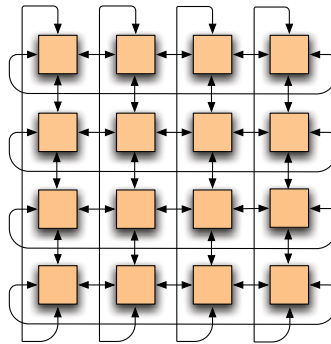


FIG. 3.1. *2D torus with end around connections*

3.2. Compute Cores. For now, each node consists of a single core, and each core is represented by a message pattern generator. The total number of cores as well as the number of cores per router must be a power of two. As in the network case, this restriction greatly simplifies the collective algorithms implementation. Simulating a full-fledged CPU core at exascale would be prohibitively expensive. For our purposes, a much faster approach will suffice, thus, we create message pattern generators that transmit messages and implement the resilience methods listed in Table 2.1.

3.3. Application Patterns. Application patterns encapsulate code key features and extract application communication patterns. We implement four message patterns that resemble the messaging behavior of a set of representative scientific applications: (1) Ghost cell, (2) Master-Slave, (3) Integer Sort (IS), and (4) Fast Fourier Transform (FT) patterns. In order to instantiate these patterns as objects in SST, we implement them as state machines, which is a

representation suitable for the event-driven simulation environment. Instead of sending messages, the pattern generators send events to the appropriate destination core informing it that a message is on its way, along with the size of that message. Local computation is emulated by simply waiting for a specified amount of time.

Each pattern generator is implemented as a state machine. They simulate compute time by suspending operations until a future event indicates elapsed time and the need to transition to another state in the state machine. The state machines contain states for waiting for messages, in case the algorithm has a dependency on incoming data. The state machines also have additional states to enable various checkpoint/restart methods, the handling of faults, and the recovery after a fault. Currently, the ghost cell pattern is implemented. Implementation of master-slave, IS, and FT patterns is in progress. In the future, we plan to extract communication patterns for more complex applications such as structured adaptive mesh refinement codes (AMR).

3.3.1. Ghost cell pattern. The ghost cell pattern is a type of structured grid computation where each point in an n -dimensional grid maps to a point in the physical space. The value of a point is a function of other neighboring points that are updated iteratively. This set of neighboring points is called a stencil. The specific stencil computation that determines a point value depends on the application. In our version of the ghost cell pattern, a point waits for the value of its four adjacent neighbors: East, west, north, and south; however, it must not be necessarily the case that the stencil points are adjacent.

3.3.2. Master-slave pattern. In the master-slave pattern one core is designated the master which distributes work to slaves running in all other cores. The slave processes do not communicate among each other and the amount of work each slave performs varies. The master is responsible for collecting results from the slaves and assigning new work as slaves become available.

3.3.3. IS pattern. This pattern is extracted from the Integer Sort (IS) of the NAS parallel benchmarks [3] and it targets integer computation and communication performance. It implements a particular bucket parallel sort of normally distributed integers generated by a random process. The input sequence is partitioned and each interval is assigned to a different process. In turn each interval is further divided into buckets to hold integers in that subinterval. The algorithm comprises three phases: (1) the sequence of integer keys are partitioned and sorted locally; (2) each participating rank sends those keys that do not belong to that rank's sorting subinterval, that is, the keys are redistributed to the appropriate buckets. This phase is communication intensive. (3) The newly integrated keys are merged with existing keys in the corresponding rank's buckets. This type of sort has application in particle method simulations.

3.3.4. FT pattern. This pattern is extracted from the Fast Fourier Transform (FT) of the NAS parallel benchmarks and targets long-distance communication performance. It implements a solution to 3D partial differential equation using Fast Fourier Transforms that are used in spectral codes.

3.4. Collective Communications. The FT and IS application patterns described in Section 3.3 perform coordinated communication operations with several processes. In the IS pattern, processes cooperate to compute the total bucket size and the number of keys each process has to send, and to send those keys to their respective processes. In the FT pattern, processes cooperate to transpose a distributed matrix. Collective operations as their name indicate are executed collectively, which means that each process in a process group calls the

```

IS()
    COMPUTE                // initialize arrays, set timers
    loop MAX-ITERATIONS
rank( iteration )
    COMPUTE                // gather timing information; verify, print results

rank( iteration )
    COMPUTE                // sort into buckets
    ALLREDUCE              // get bucket size totals
    COMPUTE                // calculate key redistribution
    ALLTOALL               // find out how many keys each process will send
    ALLTOALLV              // send key to respective processes
    COMPUTE                // calculate total number of keys, local key sort

```

FIG. 3.2. *IS pattern.*

```

FT()
loop MAX-ITERATIONS
COMPUTE                // index map, initial conditions, roots-of-unity
    // evolve to next time step in Fourier space
    ALLTOALL           // transpose planes
    REDUCE              // checksum
COMPUTE                // verify, print results

```

FIG. 3.3. *FT pattern.*

communication routine with the same parameters. Same as the communication patterns, collectives need to be implemented as state machines to integrate them into SST. In order to keep the state machines for the collective operations as simple as possible, the communication pattern component requires that the total number of ranks in a system be a power of two. There is one rank as communication pattern component in each core. The components further assume that the network is an $X \times Y$ torus with end-around connections. Each router in that torus is connected to a node which consists of a $x \times y$ network-on-chip (NoC) torus. Each router of the NoC has one or more cores attached to it, and each one of these routers corresponds to a router model component within SST. The network routers use wormhole routing, i.e. one message occupies an input and output port at a time, while the NoC routers use flit-based routing to more closely resemble a memory crossbar switch. Following is a description of the collective operations used in the communications patterns described in Section 3.3.

3.4.1. Broadcast. A broadcast performs a one-to-all operation that sends data from one process to all processes. We implement it as a binary tree, with constant message length. In the tree representation, ranks correspond to nodes and they can be either the root, interior nodes, or leaf nodes. Figure 3.4 shows the broadcast algorithm and Figure 3.5 depicts the tree representation.

3.4.2. Reduce. Reduce performs a global reduction operation and returns the result to the root. It combines the values in the input buffer of each process applying the specified operation before each send and placing the result to the output buffer of the root process. In the case of allreduce, it places the result in the output buffer of all processes. As with broadcast, the reduce operation can also be represented as a binary tree with ranks corresponding to nodes, but with the data flowing in the opposite direction. Figure 3.6 shows the reduce

```

BCAST(R)
   $\forall$  ranks  $r \in \text{group } R$ 
  if ( $r = \text{root}$ )
    send_right( $r$ )
    send_left( $r$ )
  if ( $r = \text{interior}$ )
    wait_from_parent( $r$ )
    send_right( $r$ )
    send_left( $r$ )
  if ( $r = \text{leaf}$ )
    wait_from_parent( $r$ )

```

FIG. 3.4. Broadcast algorithm.

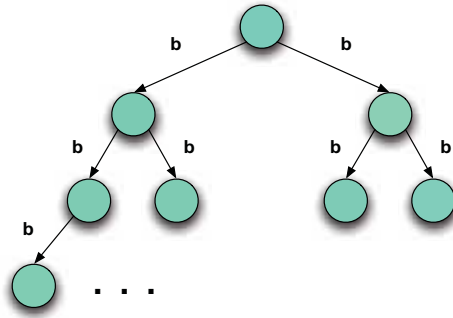


FIG. 3.5. Broadcast binary tree representation.

algorithm and Figure 3.7 shows its tree representation.

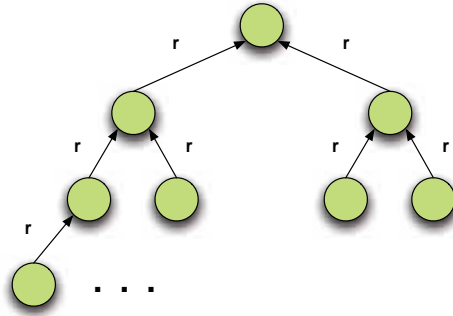
```

REDUCE(R, op)
   $\forall$  ranks  $r \in \text{group } R$ 
  if ( $r = \text{root}$ )
    wait_from_right( $r$ )
    wait_from_left( $r$ )
    compute(op)
  if ( $r = \text{interior}$ )
    wait_from_right( $r$ )
    wait_from_left( $r$ )
    compute(op)
    send_to_parent( $r$ )
  if ( $r = \text{leaf}$ )
    send_to_parent( $r$ )

```

FIG. 3.6. Reduce algorithm.

3.4.3. Allreduce. Allreduce returns the result of the global reduction operation to all processes. Allreduce is typically implemented with a reduce to the root followed by a broadcast. The IS pattern performs an allreduce to get the bucket size from all processes. Later these numbers are accumulated to perform key redistribution. Figure 3.8 shows the allreduce

FIG. 3.7. *Reduce binary tree representation.*

algorithm.

```

ALLREDUCE(R, op)
  REDUCE(R, op)
  BCAST(R)

```

FIG. 3.8. *Allreduce algorithm.*

3.4.4. Alltoall and Alltoallv. Alltoall sends a distinct message from each process to every other process. It is also known as a complete exchange. Alltoallv also sends a distinct message from each process to every other process, but messages can have different sizes and it is possible for each process to provide displacements for the input and output data.

This algorithm posts all receives followed by all sends, and waits for all communications to complete. The loop index for both sends and receives is given by $(\text{rank} + i) \bmod \text{commsize}$, otherwise, all ranks would communicate with rank i first, then rank $i + 1$ and so on, creating a communication bottleneck [19]. A nonblocking receive call indicates that the system may start writing data into the receive buffer. The receiver should not access any part of the receive buffer after a nonblocking receive operation is called, until the receive completes; similarly for the non blocking send calls, the buffer containing the data to be sent must not be changed until the operation is completed.

The FT pattern performs alltoall to compute the global matrix transposes. The IS pattern uses alltoall to find out how many keys each process will send out, and alltoallv to send those keys to the corresponding processes. Figure 3.9 shows the Alltoall algorithm.

3.4.5. Barrier. A barrier synchronizes the execution of a group of processes. No process returns from the barrier until all processes have called it, thus ensuring all processes reach the same point in the computation and are ready to proceed. Its purpose is to separate computation phases. The FT pattern calls barriers to synchronize the global matrix transposes. Figure 3.10 shows the Barrier algorithm.

3.5. Simulating Checkpoint/Restart. We aim at characterizing the overhead components associated with the different resilience methods we are comparing. To this end, we provide resilience functionality to the application by enabling Checkpoint/Restart (CR) in the simulation experiments. If resilience is enabled, the state machine for an application pattern will have additional states to provide resilience functionality. These additional states will enable various CR mechanisms, handling of faults, and recovery after a fault. In the state machine, writing checkpoints and message logs causes additional delays based on the amount of

```

ALLTOALL(R, comm_size)
  ∀ ranks  $i \in \text{group } R$ 
    DO nonblocking receive from source  $(\text{rank} + i) \bmod \text{comm\_size}$ 
  ∀ ranks  $i \in \text{group } R$ 
    DO nonblocking send to destination  $(\text{rank} + i) \bmod \text{comm\_size}$ 
  DO wait for all communications to complete

```

FIG. 3.9. *Alltoall algorithm.*

```

BARRIER(R)
  ∀ ranks  $r \in \text{group } R$ 
    if ( $r = \text{root}$ )
      send_left( $r$ )
      send_right( $r$ )
      wait_left( $r$ )
      wait_right( $r$ )
    if ( $r = \text{interior}$ )
      wait_for_parent( $r$ )
      send_left( $r$ )
      send_right( $r$ )
      wait_left( $r$ )
      wait_right( $r$ )
      send_to_parent( $r$ )
    if ( $r = \text{leaf}$ )
      wait_parent( $r$ )
      send_to_parent( $r$ )

```

FIG. 3.10. *Barrier algorithm.*

data being logged. If checkpoint data travels off-node an event that informs the destination of the checkpoint data is triggered. Processing and storing of data is emulated through delay events with a delay proportional to the amount of data being stored. Similarly, data retrieval from remote nodes during recovery is emulated with delays for retrieving local data and event traffic.

4. Experiments. The experiments consist of applying the resilience methods listed in Table 2.1 to the four communication patterns we described in Section 3.3. We will compare this results to those previously obtained using a different simulator coupled with an MPI implementation for redundant computing [11]. SST takes several input parameters for the network, node architecture, and storage. For the resilience simulation experiments, coordinated checkpointing will write checkpoints to both external storage and cabinet solid state disk (SSD). Uncoordinated checkpointing will store message logs in local flash and checkpoints on external storage or cabinet SSD. For distributed checkpointing, we store other node's checkpoints in local flash and eventual coordinated checkpoints to stable storage, either disk or cabinet SSD. For now, storage bandwidth and latency are fixed parameters, and we assume that storage is always available for recovery.

The message pattern generators assume a very specific machine configuration for the SST simulation framework. Table 4.1 summarizes the simulator input parameters.

5. Concluding Remarks. In this paper we detailed the simulation method and the experiments we propose to compare various resilience mechanisms and evaluate their perfor-

SST simulation input parameters	
Total number of cores n	2^n
Torus X dimension	2^x
Torus Y dimension	2^y
Network bandwidth	1.9 GB/s
Network latency	4.9 μ s
Crossbar switch bandwidth	12.6 GB/s
Core-router latency	150 ns
Router hop delay	25 ns
Router ports	local, north, south, east, west

FIG. 4.1. SST input parameters.

mance at extreme scale. Our objective is to characterize the different components that comprise checkpointing overhead, and determine checkpointing efficiency for several application classes. Currently the ghost pattern is implemented. It simulates ghost cell exchanges on a five-point stencil operator where each rank communicates with its east, west, south, and north neighbor. Implementations of communication patterns for FT, IS, and master/slave are under way. In addition to measuring application performance, we are interested in memory and communication overheads of the various resilience methods. For example, it is expected that the size of message logs in uncoordinated checkpointing will grow at different rates for different message patterns. We are looking for relevant performance metrics for evaluating these methods. The maximum log size could indicate the amount of extra node volatile memory that will not be available to the application or will need to be purchased. We also plan to extend our model to investigate the effect of the architecture on application resilience, more specifically how the increased number of cores in a compute node affect the overhead associated with the different resilience methods and overall application performance. We plan to validate our results by comparing the number of messages exchanged to the transmissions and network utilization of an actual application. The result will show if our application communication patterns resemble those of real applications. We also plan to compare our results to those of a message logging library currently under development. The authors would like to thank the members of the SNL Fault Tolerance group for enriching discussions on the subject, and anonymous reviewers for their valuable feedback.

REFERENCES

- [1] S. AMARASINGHE, D. CAMPBELL, W. CARLSON, A. CHIEN, W. DALLY, E. ELNOHAZY, M. HALL, R. HARRISON, W. HARROD, K. HILL, J. HILLER, S. KARP, C. KOELBEL, D. KOESTER, P. KOGGE, J. LEVESQUE, D. REED, V. SARKAR, R. SCHREIBER, M. RICHARDS, A. SCARPELLI, J. SHALF, A. SNAVELY, AND T. STERLING, *Exascale software study: Software challenges in extreme scale systems*, tech. rep., DARPA IPTO, Air Force Research Labs, September 2009.
- [2] ARUN RODRIGUES, *The structural simulation toolkit*. <http://www.cs.sandia.gov/sst/>, Aug. 2010.
- [3] D. H. BAILEY, E. BARSZCZ, J. T. BARTON, D. S. BROWNING, R. L. CARTER, L. DAGUM, R. A. FATOOGHI, P. O. FREDERICKSON, T. A. LASINSKI, R. S. SCHREIBER, H. D. SIMON, V. VENKATAKRISHNAN, AND S. K. WEERATUNGA, *The NAS Parallel Benchmarks—summary and preliminary results*, in Supercomputing '91: Proceedings of the 1991 ACM/IEEE conference on Supercomputing, New York, NY, USA, 1991, ACM, pp. 158–165.
- [4] J. F. BARTLETT, *A nonstop kernel*, in SOSP '81: Proceedings of the eighth ACM symposium on Operating systems principles, 1981, pp. 22–29.
- [5] G. BRONEVETSKY, D. J. MARQUES, K. K. PINGALI, R. RUGINA, AND S. A. MCKEE, *Compiler-enhanced incremental checkpointing for OpenMP applications*, in PPOPP '08: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming, New York, NY, USA, 2008, ACM, pp. 275–276.

- [6] J. CHAPIN, M. ROSENBLUM, S. DEVINE, T. LAHIRI, D. TEODOSIU, AND A. GUPTA, *Hive: fault containment for shared-memory multiprocessors*, in SOSP '95: Proceedings of the fifteenth ACM symposium on Operating systems principles, New York, NY, USA, 1995, ACM, pp. 12–25.
- [7] T.-C. CHIUH AND P. DENG, *Evaluation of checkpoint mechanisms for massively parallel machines.*, in Annual Symposium on Fault Tolerant Computing, Sendai, Japan, June 1996, IEEE Computer Society Press, pp. 370–379.
- [8] X. DONG, N. MURALIMANOVAR, N. JOUPPI, R. KAUFMANN, AND Y. XIE, *Leveraging 3d PCRAM technologies to reduce checkpoint overhead for future exascale systems*, in SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, New York, NY, USA, 2009, ACM, pp. 1–12.
- [9] E. N. M. ELNOZAHY, L. ALVISI, Y.-M. WANG, AND D. B. JOHNSON, *A survey of rollback-recovery protocols in message-passing systems*, ACM Comput. Surv., 34 (2002), pp. 375–408.
- [10] C. ENGELMANN AND A. GEIST, *A diskless checkpointing algorithm for super-scale architectures applied to the fast fourier transform*, in In Proceedings of the International Workshop on Challenges of Large Applications in Distributed Environments, Seattle, WA, June 2003, IEEE Computer Society Press, pp. 47–52.
- [11] K. FERREIRA, R. RIESEN, R. OLDFIELD, J. STEARLEY, J. LAROS, K. PEDRETTI, R. BRIGHTWELL, AND T. KORDENBROCK, *Increasing fault resiliency in a message-passing environment*, Technical report SAND2009-6753, Sandia National Laboratories, Oct. 2009.
- [12] F. C. GÄRTNER, *Fundamentals of distributed computing in asynchronous environments*, ACM Comput. Surv., 31 (1999), pp. 1–26.
- [13] D. McEVoy, *The architecture of Tandem's nonstop system*, in ACM '81: Proceedings of the ACM '81 conference, New York, NY, USA, 1981, ACM, p. 245.
- [14] A. MOODY, G. BRONEVETSKY, K. MOHROR, AND B. R. DE SUPINSKI, *Design, modeling, and evaluation of a scalable multi-level checkpointing system*, in Proceedings of the ACM/IEEE international conference for High Performance Computing, Networking, Storage, and Analysis, November 2010.
- [15] H. PACKARD, *HP NonStop computing*. <http://h20338.www2.hp.com/NonStopComputing/cache/76385-0-0-0-121.html>.
- [16] J. S. PLANK, K. LI, AND M. A. PUENING, *Diskless checkpointing.*, IEEE Transactions on Parallel and Distributed Systems, 9 (1998), pp. 972–986.
- [17] J. C. SANCHO, F. PETRINI, G. JOHNSON, J. FERNANDEZ, AND E. FRACHTENBERG, *On the feasibility of incremental checkpointing for scientific computing*, Parallel and Distributed Processing Symposium, International, 1 (2004), p. 58b.
- [18] B. SCHROEDER, E. PINHEIRO, AND W.-D. WEBER, *DRAM errors in the wild: a large-scale field study*, in SIGMETRICS '09: Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems, New York, NY, USA, 2009, ACM, pp. 193–204.
- [19] R. THAKUR AND R. RABENSEIFNER, *Optimization of collective communication operations in MPICH*, International Journal of High Performance Computing Applications, 19 (2005), pp. 49–66.
- [20] C. YU, D. ZHI-HUI, P. LIU, AND L. SAN-LI, *OS kernel supported fault tolerant MPI*, Journal of Shanghai University, 5 (2001), pp. 18 – 21.

STATISTICAL ANALYSIS OF HPC ALERTS AND DEVELOPMENTS IN ROOT CAUSE ANALYSIS

JOEL M. VAUGHAN ^{*}, JON R. STEARLEY [†], SCOTT A. MITCHELL [‡], AND GEORGE MICHAILIDIS [§]

Abstract. Supercomputers are inherently complex. When they fail, determining which of the many components making up the system is the cause of the fault is challenging. Solving this problem in full requires a thorough understanding of how alerts and faults are related. Building on previous work, this paper further demonstrates the effectiveness of combining of system log tools and statistical analysis to answer relevant questions about supercomputers. In the first part of this paper, we analyze two types of alerts on a real system. We then discuss several important issues in root cause analysis and potential research directions.

1. Introduction. Supercomputers, or high performance computing (HPC) systems, are made up of a complex network of various types of components, ranging from compute nodes to disks to network cables. In a more general sense, the users of the system and software version may also be considered components. Throughout this work, the term *factor* is used to distinguish the type of component, e.g. compute node, network cable, or user. The successful operation of such a machine requires all these components to both work individually and interact in certain ways. Should one or more component fail to function or interact correctly, a *fault* occurs, resulting in the failure of one or more jobs. In order to determine the cause of the problem, it is necessary to examine system *alerts*. Alerts are messages in system logs indicating a fault or potential fault. Like the logs in which they reside, alerts are semi-structured, timestamped text. This text conveys a variable amount of information about the underlying cause. The alerts themselves may be either *direct*, in which a component reports that it has experienced a fault, or *indirect*, where the component reporting the fault is not the cause. The *root cause problem* refers to the difficult problem of determining the component responsible for a fault by using the limited information available in the alerts.

Previous work by the authors ([16] and [15]) has demonstrated a working solution to the root cause problem. This solution is based on a multiple simultaneous hypothesis testing framework with a false discovery rate (FDR) control, originally developed by Benjamini and Hotchberg [2]. Although this method has shown to be effective in many situations, there are still challenges involved in the root cause problem. These challenges include determining the significance of component types, implementing the analysis tools on production systems, and developing ways to scale up the methods to ever larger systems. This paper addresses these areas. However, further development of these methods requires a more detailed understanding of how alerts are generated and related to the underlying faults. Thus, this paper examines several statistical properties of these alerts in detail.

Splunk [10] is a piece of software designed to efficiently index and search IT information. It is designed to be inherently customizable for specific situations. Splunk has been customized for and deployed on Red Sky, one of the HPC systems at Sandia National Laboratories. This customization and deployment has greatly reduced the amount of effort needed to examine the system logs and facilitated collaborations between researchers and system administrators [12], including the work described here.

In this paper, we carefully examine the distribution of two types of alerts (“Out of Memory” and “SymbolError”), and answer a relevant question related to each type of alert. For the “Out of Memory” alert, we attempt to determine whether these alerts should be attributed

^{*}University of Michigan (while working at Sandia National Laboratories), rsnation@umich.edu

[†]Sandia National Laboratories, jrstea@sandia.gov

[‡]Sandia National Laboratories, samitch@sandia.gov

[§]University of Michigan, gmichail@umich.edu

to the job currently running when an alert occurs, or as a residual effect of a previous job. We investigate to what degree the “SymbolError” alerts vary with respect to the physical location of the component recording the alert. We then discuss four ongoing research branches related to the root cause problem and describe our current progress and future direction in each case.

2. Distribution of Host and Cable Alerts:. In this section, we present an analysis of two types of alerts from Red Sky. The two alerts were chosen because they are both currently of interest to the system administrators. The first, “Out of Memory,” is reported by the compute nodes themselves, while the second, “SymbolError,” is reported by switches on the Infiniband network that connects the compute nodes with each other and with the rest of the system. For each alert, we examine, to the degree currently possible, the distribution of alerts across both time and network space. In addition, for each alert type, we attempt to shed light on a current and relevant question of interest related to that alert. With the “Out of Memory” alert, we investigate whether the alert is caused by code running at the time the alert is observed or if the alert is the result of some previous machine state. The “SymbolError” alerts are examined as a function of their spatial location, do determine whether certain locations are significantly more likely to experience such alerts.

2.1. “Out of Memory” Alerts. Because the compute nodes of Red Sky are diskless, users cannot use more memory than is available on a node. “Out of Memory” (OOM) events occur when the Linux kernel kills a process in order to obtain memory. Intuitively, it seems as though these alerts are directly caused by users writing code less carefully than is necessary. However, it has been suggested that the linux kernels used on the compute nodes can leak memory. More specifically, Brandt et al claim that these alerts are the result of memory leaks. [3] For example, one user could use a certain quantity of memory, but not all this memory would be available to the next user. This user, expecting 16 GB of memory might, because of the leak, only have access to 10 GB. This would result in an OOM and potentially the failure of a job that should have succeeded. In this case, it would look as though the second user was the cause of the fault whereas the fault actually lay in a different factor (the OS).

Using Splunk, we are able to obtain a list of OOM alerts, indicating a time stamp of the alert and the host that reported the alert. Here, we can view the data as a realization of a point process, meaning we have available the exact time and location of each alert. Using Splunk “lookups”, we associate these alerts with a particular job and consequently with a particular user. Finally, a database called genders (also implemented as a lookup table within Splunk) gives both the physical and network location of the computers. We have examined the distribution of the alerts over many factors but only present those that seem the most interesting, or those that are relevant to the question of interest. This analysis concerns OOM alerts on RedSky from the period of June 20, 2010 to June 26, 2010.

Distribution of Alerts. The distribution of OOM alerts were studied over several factors, as well as time and space. Here we present a small selection of the analyses carried out.

The authors and others have used a Poisson process as a generating model for alerts and the underlying faults in HPC systems. (See [1] or [15].) In order to determine how reasonable this basic model describes OOM alerts in practice, we examined the inter-arrival times of the alerts. Specifically, we examine the unique inter-arrival times. That is, we consider the system-wide inter-arrival times. If OOM alerts occur simultaneously on multiple hosts, this is considered a single event. This simplification mitigates the non-random effect of multiple-node jobs, where a single job can experience OOM alerts simultaneously on several nodes. If the Poisson process is indeed a good model for the arrival process of the alerts, then the inter-arrival times between these OOM alerts will be exponentially distributed. Figure 2.1 shows an exponential QQ plot of the inter-arrival times. If the observations match the exponential

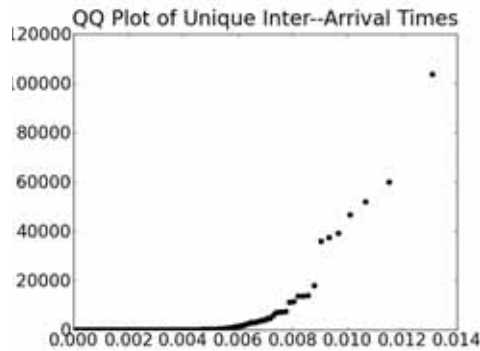


FIG. 2.1. Analysis of OOM inter-arrival times. We see that a Poisson model does not appear appropriate for this process, as the QQ plot does not follow a 45 degree line.

distribution, this plot would show the points along a 45 degree line with a positive slope. However, the data deviate noticeably from exponential distribution, indicating that this is not a good model for OOM alerts. However, slightly more complicated models, such as an inhomogeneous Poisson process, might be appropriate. Such a model would take into account that the system's utilization varies with time. Allowing the intensity of the Poisson process to vary with time would better reflect the behavior of the system.

We next examine when OOM alerts occur within a particular job. In this paragraph and the next, we consider all OOMs from our period of study, and their temporal location within their job (the job to which the processor experiencing the OOM is currently assigned). In particular, we can examine the time between the start of a job and the OOM alerts or the time from the OOM to the end of the job. Both distributions are shown in Figure 2.2. We see that the vast majority of times between Job Start and OOMs are between one and ten minutes, with a very long tail to the right. (The figure is truncated.) That is, most OOMs occur within the first ten minutes of the beginning of the job, while a few occur much later.

On the other hand, the second figure shows a bimodal distribution of the time from the OOM alert to the end of the job. This shows that jobs tend to end either soon after (or slightly before) the OOM or continue for a long time before ending. This would suggest that the state that causes the OOM alert is either immediately fatal (the first case) or allows the job to continue (the second case.) Further analysis using the logs reveal that these two modes represent very different distributions of the several possible "end states" of the job. Additional analysis of these state labels are necessary, but initial inspection shows that many of the jobs that continue running long after they experience OOMs end in a "failed" state. If this label accurately indicates that the job is unsuccessful, this result suggests a policy of killing such jobs might be useful in both saving machine resources and allowing users to know of problems in their code hours sooner.

Question of Interest. One relevant question is whether OOMs are more likely caused by the *current* user (the user who experienced the OOM during his or her job) or the memory leak phenomenon. Generally, the OOM alerts are related to code using more memory than is available. Since it is not possible to directly examine the code for each job, we instead examine the users. Some users, and the code they run, are more likely to stress the memory of the machine. Thus, the question of memory leaks vs. current user can be asked as the question of whether the users running the jobs are more strongly associated with the alerts, or the prior users of nodes are more strongly associated. We examined the OOM alerts as a function of current user, and as a function of *previous* user. That is, rather than attributing

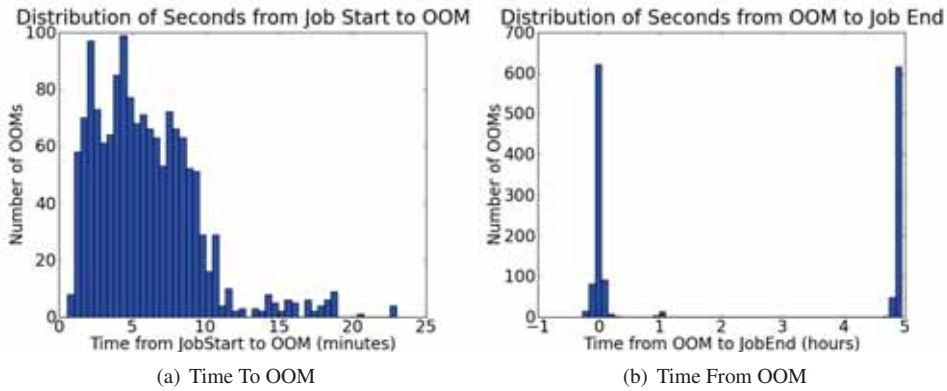


FIG. 2.2. Distribution of time from Job Start until OOM alert (left) and distribution of time from OOM alert to Job End (right). Note that the negative values in the right-hand graph indicate the job was ended by the scheduler before the OOM alert on that node.

the alert to the individual running a job, they were attributed to the individual who had last run a job on the host reporting the OOM. If the alerts are caused by the *current* user, we would expect a stronger association between alerts and current user, while if they were in fact caused by the memory leak problem, we would expect a stronger association between the *previous* users and the OOM alerts, because the previous user would be the one responsible for stressing the memory and putting the node in the vulnerable state.

Figure 2.3 shows the association between OOM alerts and users. In these figures, the association of OOM alert with host is maintained, while the alerts are attributed to users in different ways. As a baseline (on the left) we show the alerts randomly assigned to users, weighted by overall usage. If there were truly no association between the OOM alerts and users, we would expect to see a graph such as this. Next, in the middle, we show the results of attributing the OOM alerts to the users running the jobs that experienced them. In the third figure, we show the results of attributing the alerts to the previous user. We constructed other variations of the previous user graph, but all were somewhat similar to the one displayed. We see that neither the current nor previous user allocation resembles the random assignment graph, and thus there is a signal present in both of these assignments, indicating that both factors might indeed be relevant. The signal appears stronger in the current user setting, indicating that it is a more significant factor than the previous user; however, the signal associated with previous user should not be ignored.

To further investigate, we carried out our FDR analysis [15] on both the current user and previous user factors. The results are shown in Table 2.1. In each case, the distribution of p -values was also examined via histograms (not shown). They are more uniformly distributed in the previous user case, indicating that this case has a weaker signal. We use the job table to conduct a closer examination of the “significant” users in each case. The most significant previous user, *user117*, ran only one job over the time period studied and experienced no OOMs directly. The nodes in the job were next taken by four different jobs, only one experienced OOM alerts. This job was run by *user12*, who was identified as significant in both the current and previous user case. It is more reasonable to assume that *user12* is responsible for these alerts, given the other evidence against him, than to assume they were caused by a memory condition established by *user117*. Furthermore, *user12* often ran successive jobs on the same node, meaning many OOMs attributed to this person as the previous user are also attributed as the current user, possibly explaining the user’s significance. Thus,

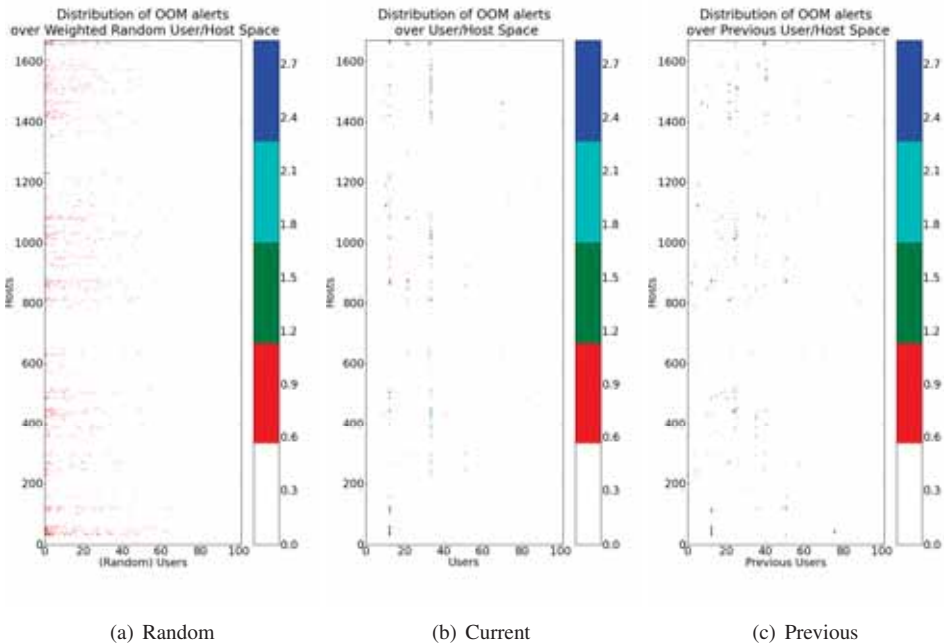


FIG. 2.3. Distribution of OOM alerts across users and hosts, via different methods of attributing alerts to users. In all figures, users are arranged from left to right in order of decreasing machine usage. The left figure shows the distribution if alerts are randomly assigned to users, based on each user’s total node–hours. The middle figure shows the distribution if alerts are attributed to the current user. The right figure shows the distribution if the alerts are attributed to the previous user. In comparison the randomly assigned alerts, both the current and previous user distributions show a signal, indicating that both are relevant factors, with the stronger signal in the current user graph indicating that this factor is more important with regards to OOM alerts.

User	p-value	Comp.	Reject?	Previous User	p-value	Comp.	Reject?
user12	4.2e-05	0.0023	True	user117	3.8e-10	0.0023	True
user88	0.0034	0.0045	True	user12	0.0027	0.0045	True
user21	0.023	0.0068	False	user113	0.012	0.0068	False
user50	0.033	0.0091	False	user28	0.044	0.0091	False
user16	0.13	0.011	False	user23	0.051	0.011	False
user3	0.16	0.014	False	user14	0.063	0.014	False
user25	0.16	0.016	False	user19	0.065	0.016	False
user73	0.16	0.018	False	user50	0.068	0.018	False
user14	0.17	0.02	False	user88	0.083	0.02	False

TABLE 2.1

Results of FDR Root Cause Analysis of OOM alerts, comparing results of associating alerts with current user (left) and previous user (right). In each case, the ten users with the smallest p–values are displayed, and sorted by p–value. The rejection criteria is a varying threshold based on the false discovery rate controlling procedure of Benjamini and Hochberg [2], which is discussed in relation to the Fault–Cause Problem in [16]. Note that in both cases, two users are chosen as significant. (Although the data is real, the user names have been anonymized. The numbers correspond to the columns in Figure 2.3.)

the two significant previous users are easily explained in terms of current users. In contrast, the two significant current users are not easily explained in terms of previous users. Each ran many jobs that experienced OOMs, and these jobs had a large number of different previous users. After this is taken into account, there is much stronger evidence associating the OOM alerts with the current user, as opposed to the previous user as the result of a memory leak.

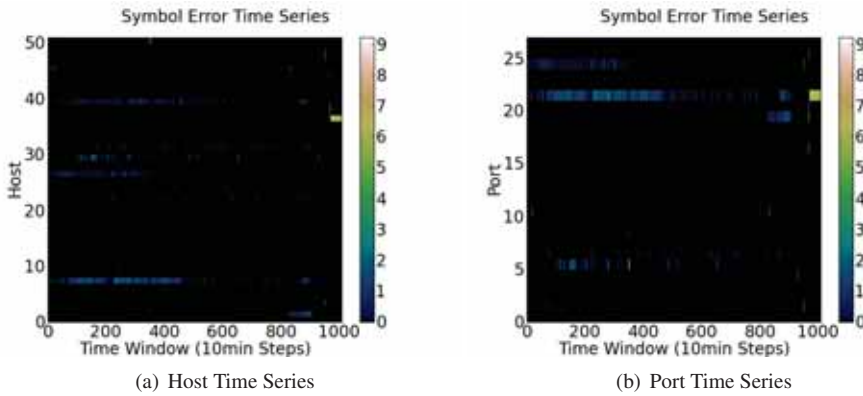


FIG. 2.4. Time series of SymbolError alerts, by host (left) and port (right). The number of alerts is shown on a log scale to improve visibility. The yellow block in the upper right area is a host that experienced a high number of alerts during a few consecutive time periods. The system administrators suggested that this is the result of a change made to the system.

2.2. “SymbolError” Alerts. The Infiniband network that connects the various components is crucial for the successful execution of all jobs, whether to communicate across multiple nodes, access the network disk arrays, or even launch. SymbolErrors occur when a switch recognizes a portion of a received packet as being corrupted during transmission. Unlike the OOM alerts, the SymbolError alerts are not reported with individual time stamps. Every ten minutes, each switch reports the number of SymbolErrors it experienced on each of its ports. Thus, this data is a set of time series, one for each switch/port pair. In order to associate these with jobs as we did with OOMs, we must join the alerts, jobs, and network routing tables. The routing tables consist of the ordered list of network ports a transmission passes through for host A to communicate with host B (and B to A takes a different path). There are five to ten million such route changes per day, and we are still working on sufficiently joining all these data. However, via the afore-mentioned genders table and a table of physical links (which do not change with time), we are able to associate the alerts with their physical locations. Below, we examine the SymbolErrors that occurred on Red Sky between July 1, 2010 and July 7, 2010.

Distribution of Alerts. We next present a subset of the analysis of SymbolError alerts. Figure 2.4 shows the time series of the alerts by host and port. We notice different patterns across hosts, with a high-alert event near the end of the period studied. Some hosts and ports that rarely see SymbolErrors, while others see them somewhat regularly. A third set saw them infrequently except during a small time window, during which they saw many. As with the OOM alerts, it is of interest to determine if a Poisson model is appropriate to describe the behavior of SymbolErrors. Since event inter-arrival times are not available, we instead evaluate whether the counts of SymbolErrors among time periods follow a Poisson distribution. Therefore, we construct QQ plots comparing the observed distribution of counts to a Poisson distribution. The results, in Figure 2.5, indicate that the Poisson model is not appropriate to model the aggregate SymbolErrors. We followed up by examining each host separately. Of the 52 hosts experiencing alerts during the time studied, 9 exhibit Poisson behavior. One such host, (b4-nem1-b) is shown on the right-hand side of Figure 2.5. Of the remaining hosts, 27 showed rare positive alert counts of small magnitude, while 16 showed equally rare alert occurrences, but where the counts were much higher. While the 27 hosts with rare small positive counts could be modeled as Poisson with a very low rate, the model is

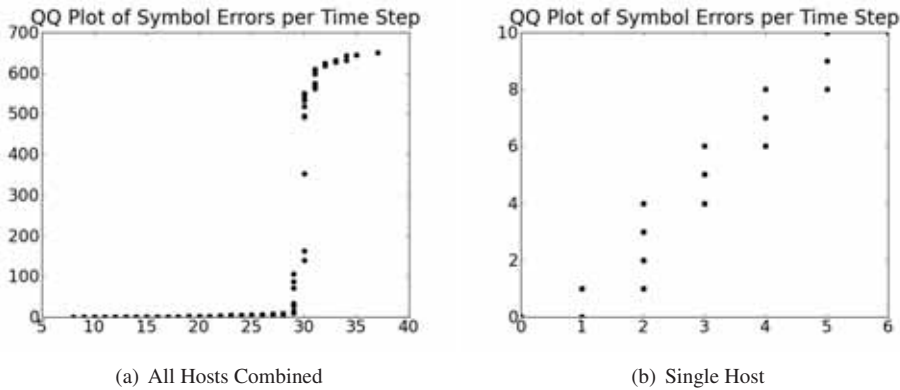


FIG. 2.5. Poisson *QQ* plots of *SymbolError* time series. The left plot, showing the aggregate *SymbolErrors* from all hosts, shows a deviation from the Poisson distribution. The right plot, which considers *SymbolErrors* on a single host, shows that the Poisson model is reasonable for this host. Eight other hosts show similar behavior, while 27 hosts could be modeled by a Poisson process with a low rate. However, 16 hosts have rare but high counts, a phenomenon which is not captured well by the Poisson model that spoils the aggregate model.

not appropriate for the hosts that have rare but large counts. These results suggest a modeling approach where each host independently exists in one of three alert-generating states: off, when the host experiences no alerts; on, where the host experiences alerts according to a Poisson mechanism; and high, where the host experiences a high number of alerts.

Question of Interest. The nodes making up Red Sky are placed in racks, arranged in four rows and five columns. Each rack consists of four shelves, stacked one on top of the other. Each shelf consists of 12 slots that contain the individual compute nodes. It has been hypothesized that there is a manufacturing defect in the hardware, making *SymbolErrors* more likely in the backplane connections to certain slots. We investigated the distribution of *SymbolErrors* across slots, shown in Figure 2.6. In the left hand figure, it is clear that *SymbolErrors* occur more frequently in certain slots (e.g. slot 8) than others. The right hand graph shows the numbers of populated slots. The near-uniform distribution of this figure indicates that the higher number of alerts in certain slots is not simply because these slots are occupied more than other slots. To be more rigorous, we perform a Chi-squared test against the uniform distribution where each slot is equally likely to receive alerts, and get a test statistic of $X^2 = 5807.4$ and a p -value ≈ 0 . Normalizing by usage results in a test statistic of $X^2 = 7386.3$ and a p -value ≈ 0 . Both tests indicate the distribution of *SymbolErrors* across slots is significantly different from what would be expected if the alerts were randomly assigned slots, indicating that there is some systematic difference across the machine. One explanation could be a manufacturing defect, although other possibilities, such as the way cables rest, could also be responsible.

We also examined the distribution across the other spatial dimensions: shelf, row, and column. These distributions can be seen in Figure 2.7. It is surprising to note that in each case, there is one or two levels that experience more alerts than others. The patterns in Figures 2.6 and 2.7 are not the result of a single faulty host, suggesting a systematic cause across these dimensions. This phenomenon had not been previously noticed by the system administrators, and there is currently no explanation for the systematic differences noticed, although it is being investigated.

3. Ongoing Developments in Root Cause Analysis. In the previous sections, we have shown the usefulness of statistical analysis of log data in answering relevant questions about

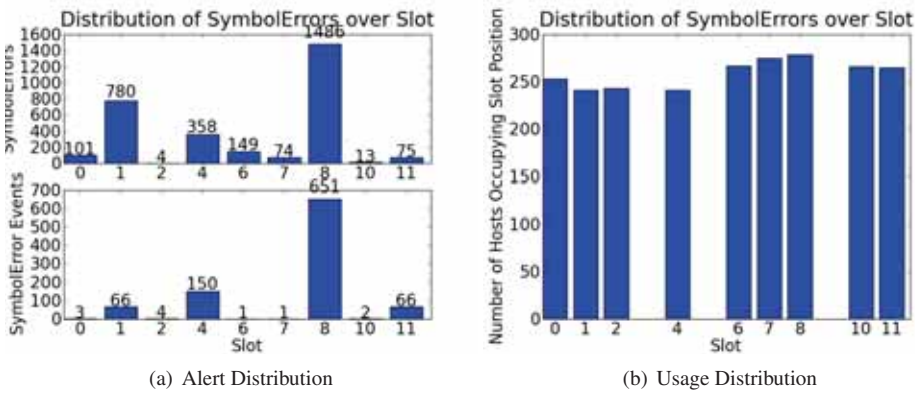


FIG. 2.6. Distribution of SymbolErrors across Slot, showing noticeably more SymbolErrors in certain slots. The top left plot shows the total number of SymbolErrors by slot, while the lower left plot shows the total number of time windows with a positive SymbolError count for each slot. The slot usage (right) is much more uniform than either graph.

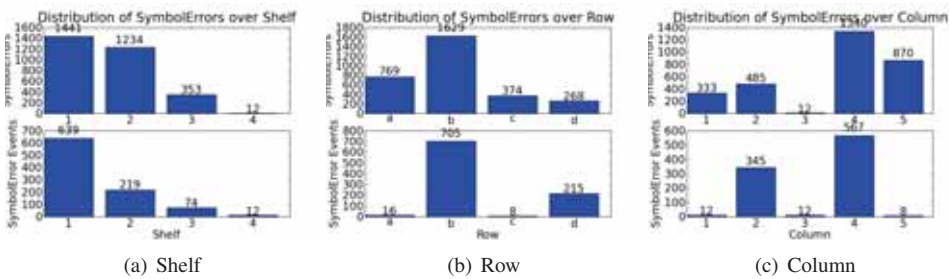


FIG. 2.7. Distribution of SymbolErrors across shelf, row, and column. The top figure shows the number of SymbolErrors, while the lower figure shows the number of time windows with a positive SymbolError count for each dimension. Note that all three of these dimensions, some entries see noticeably more SymbolErrors than others.

Red Sky. In the remaining sections, we discuss our ongoing work in Root Cause Analysis, showing how statistical analysis of logs is being developed to be both more effective at determining the root cause and more accessible to network administrators.

3.1. Custom Splunk Search and Analysis. Here we report on the start of turning our research methodology into a deployed capability. Root cause analysis like the work presented above and the FDR-based analysis we present in [16] and [15] is useful, but it requires multiple steps: aggregating and processing multiple system logs, searching these logs for relevant information, performing relevant statistical analyses, and reporting on the results. If these steps need to be performed by multiple individuals, even the useful methodologies we have developed have little chance of being employed in practice. Splunk's capabilities to efficiently combine and present data from across multiple logs have already been employed for the first steps in this analysis, with a considerable savings in the work necessary to examine the logs, as described in [12].

Splunk provides several ways to create custom search commands within the program [11]. By taking advantage of Splunk's Python interface, we have prepared some of the analyses from [15] as custom searches, shown in Figure 3.1. Thus, Splunk can be used by system administrators or others to search, aggregate, and analyze the data to perform FDR based root cause analysis of certain types of alerts using a single command from the Splunk web

interface.

pval ↕	host ↕	Reject? ↕	compval ↕
0.00730721338059	rs340	False	0.000122025625381
0.0956510928214	rs545	False	0.000244051250763
0.10256893328	rs343	False	0.000366076876144
0.102576603661	rs348	False	0.000488102501525
0.157716463669	rs286	False	0.000610128126907
0.157733942834	rs288	False	0.000732153752288

FIG. 3.1. Screenshot of results of FDR custom search command. With a single command, the splunk user dispatches the relevant searches and performs our FDR root cause analysis, generating the output shown.

3.2. Cables as Components. It has been determined by extensive testing that the number of SymbolErrors is dependent on the traffic patterns that the cables are carrying. Thus, based on our previous work [15], a job-centric analysis of the SymbolError alerts should therefore help determine which jobs “cause” SymbolErrors (or at least, how to reliably reproduce their occurrence, which is often the first step to finding a solution). There is, however, a key difference. The alerts studied in our previous work appear on hosts. A job-centric analysis of these alerts assigns each component to a single job at a given time. However, cables may be shared by multiple jobs at any given moment. Our methodology does handle the situation of shared components, but as previously described we are currently unable to perform the necessary join of alert, job, and routing information.

3.3. Component Types by Significance. We now briefly discuss an ongoing and important research question. Consider again the OOM alerts described in Section 2.1. With our existing methodology, it is possible to independently search for statistically significant hosts or users, as well as other factors. However, these factors are closely related. In fact, the very same alerts must be used for both user and host, since the alerts occur on a particular host while being employed by a particular user. Thus, if an analysis turns up significant components of multiple types, which type of component is most likely to be the root cause? This problem is of primary importance, as a successful answer will result in effort being quickly focused on solving the problem at the root cause, rather than trying to determine the cause or trying to fix a component other than the cause of the problem. Often, further analysis can show, at least intuitively, that one factor is more relevant than another. Consider again Figure 2.3(b). This figure shows that although both hosts and users might be identified as significant, the spread of these alerts across users is far more organized than the distribution across hosts, indicating that the user factor is the most relevant to this type of alert. Although this follow-up analysis is useful, it relies partially on intuition. As we continue working on this problem, we hope to both formalize this technique and automate the procedure, making it available to system administrators as a useful and relevant tool. One possible solution is the group hypothesis testing approach discussed in the following section.

3.4. Group Hypothesis Testing. Our original approach to the root cause problem is based on hypothesis testing with FDR control. This method was also used extensively in communities analyzing gene microarray experiments. In both contexts, the large number of simultaneous tests gained statistical power from the FDR based approach as compared to more traditional multiple testing paradigms. However, as technology advances, the number of tests is continuing to grow, and pressing the limits of even the FDR based method. This is true

both in the microarray setting, where improved technology allows for an ever increasing number of experiments to analyze, and the HPC setting, where computers are being constructed with more and more components. As the number of components increases, it becomes nearly impossible to obtain a statistically significant result, even in the presence of strong evidence.

In order to handle the increasing number of hypotheses, those working with microarrays have suggested grouping meaningful sets of individual genes together, and conducting a series of hypotheses tests across the groups rather than the individual genes. If relevant to the research question at hand, a follow-up analysis can examine only the significant groups and test the individual genes within the group. (See [13] and [4].) This procedure makes large gains in power, as there are fewer hypotheses both when testing for significant groups and when testing within the group. The complexity of HPC systems and the nature of our data available make it difficult to apply these methods directly; however, we have made appropriate modifications to the methods advanced by Efron and Tibshirani [4] and carried out some preliminary simulation studies. These studies showed the potential of this method to greatly improve the overall power of the procedure in situations where our previous method would suffer.

As noted by Efron and Tibshirani, the groups need to be constructed in a meaningful way. Our simulations have suggested that poorly chosen groups do not provide sufficient statistical power, while well-chosen groups perform very well. Table 3.1 shows the power obtained in our simulations. Note that the power depends strongly on the grouping structure used and in some cases on the number of groups considered. Thus, it is important to use a carefully chosen group structure.

The importance of the chosen group structure in this procedure has the potential to be useful in solving the root cause problem, particularly the issue of determining the most significant factors, as discussed in Section 3.3. Rather than trying to determine a single grouping structure, we propose developing several grouping structure, each based around a specific factor type. We then will run the testing procedure on each one of the groups. If the grouping structure is not meaningful, no group should be chosen as significant, and this will suggest that the factor associated with the grouping structure is not significant as a root cause. However, if a grouping structure is meaningful, one or more groups will be identified as significant. This will identify the associated factor as significant, and an analysis of the significant groups will identify which individual components are the root cause. For example, consider two grouping strategies, one that puts together users running similar code, while the other puts together hosts that are in similar physical locations. If the root cause of a problem is user-related, one or more of the user-based groups will be significant, while the host-based groups would not be.

Although the above plan is theoretically promising, one difficulty that must be overcome is creating group structures that associate with the factors being considered. We have begun exploring the creation of these groups by partitioning a series of association graphs. For a given factor, we hope to construct a graph describing the associations between components, and then form groups by partitioning that graph. For partitioning, we plan to use software called Metis [7].

4. Conclusion. By using the capabilities of Splunk, we have been able to gather and aggregate data on system alerts from a diverse set of relevant logs. This information has then been used to perform statistical analysis to answer relevant questions about Red Sky, and raised some new questions. In addition, this work has demonstrated the necessity of continuing work on the root cause problem, formally addressing both the problem of determining the most significant factor and that of increasing numbers of components. Finally, in order for these methods to be useful, they must be readily available to those who maintain such sys-

Number of Groups:	Direct Alerts			Indirect Alerts		
	Structure 1	Structure 2	Structure 3	Structure 1	Structure 2	Structure 3
5	1.00	0.06	0.00	1.00	0.05	0.08
10	1.00	0.00	0.66	1.00	0.05	0.40
15	1.00	0.03	0.77	1.00	0.08	0.63
25	1.00	0.75	0.81	1.00	0.29	0.79

TABLE 3.1

Statistical power of group hypothesis testing method. Three different grouping structures were tested for each of 200 simulated iterations. The structures were chosen so that in the simulation, different numbers of faulty hosts were put into groups in the three different structures. Notice that the power of the group testing depends greatly on both the grouping structure chosen for the analysis and the number of groups chosen. Structure 1, which collects all faulty hosts in the same group, performs well (power ≈ 1) regardless of the number of groups. Structures 2 and 3, which spread the faulty hosts over multiple groups, perform better with a larger number of groups.

tems. By providing our methods via custom Splunk searches, it is more likely that progress made on the root cause problem will translate into less time spent finding problems and more time using HPC systems for their intended purpose.

Acknowledgments. The authors would like to thank the Red Sky system administrators for their help in obtaining and interpreting the data. In particular, we thank Sophia Corwell, Matthew Bohnsack, Ken Lord, and Marcus Epperson. We thank David Day for his helpful introduction to Metis. Software used in this work include: R [9], Python [14], Matplotlib [5], Scipy [6], Numpy [8], Metis [7].

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

REFERENCES

- [1] R. A. BALLANCE, J. T. DALY, AND S. E. MICHALAK, *Application monitoring*, in Proceeding of 2008 Cray User Group Meeting, 2008.
- [2] Y. BENJAMINI AND Y. HOCHBERG, *Controlling the false discovery rate: a practical and powerful approach to multiple testing*, Journal of the Royal Statistical Society, Series B., 57 (1995), pp. 289–300.
- [3] J. BRANDT, A. GENTILE, J. MAYO, P. PÉBAY, D. ROE, D. THOMPSON, AND M. WONG, *Methodologies for advance warning of compute cluster problems via statistical analysis: A case study*, in Workshop on Resiliency in High Performance Computing (Resilience) at the 18th ACM International Symposium on High Performance Distributed Computing (HPDC), 2009.
- [4] B. EFRON AND R. TIBSHIRANI, *On testing the significance of sets of genes*, The Annals of Applied Statistics, 1 (2007).
- [5] J. D. HUNTER, *Matplotlib: A 2d graphics environment*, Computing in Science and Engg., 9 (2007), pp. 90–95.
- [6] E. JONES, T. OLIPHANT, P. PETERSON, ET AL., *SciPy: Open source scientific tools for Python*, 2001–. Retrieved August 2010.
- [7] G. KARYPIS AND V. KUMAR, *A fast and highly quality multilevel scheme for partitioning irregular graphs*, SIAM Journal on Scientific Computing, 20 (1999).
- [8] T. E. OLIPHANT, *Python for scientific computing*, Computing in Science & Engineering, (2007).
- [9] R DEVELOPMENT CORE TEAM, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2010. ISBN 3-900051-07-0.
- [10] SPLUNK, <http://www.splunk.com/product>. World Wide Web, 2009. Retrieved August 2009.
- [11] ———, *Splunk developer manual: custom search commands*. <http://www.splunk.com/base/Documentation/latest/Developer/SearchScripts>, 2010. Retrieved July 2010.
- [12] J. STEARLEY, S. CORWELL, AND K. LORD, *Bridging the gaps: Joining information sources with splunk*, Submitted to Workshop on Managing Systems via Log Analysis and Machine Learning Techniques, (2010).
- [13] A. SUBRAMANIAN, P. TAMAYO, V. K. MOOTHA, S. MUKHERJEE, B. L. EBERT, M. A. GILLETTE, A. PAULOVIKH, S. L. POMEROY, T. R. GOLUB, E. S. LANDER, AND J. P. MESIROV, *Gene set enrichment analysis: A knowledge-*

- based approach for interpreting genome-wide expression profiles*, Proceedings of the National Academy of Science, 102 (2005), pp. 15545–15550.
- [14] G. VAN ROSSUM ET AL., *Python language website*. <http://www.python.org>. Retrieved August 2010.
 - [15] J. VAUGHAN, J. STEARLEY, G. MICHAILIDIS, S. MITCHELL, AND R. LAVIOLETTE, *Root cause analysis of networked computer alerts*. Submitted to Technometrics, SAND Number: 2010-3917J, 2010.
 - [16] J. M. VAUGHAN, J. R. STEARLEY, S. A. MITCHELL, AND G. MICHAILIDIS, *Root cause analysis of errors for high performance computing*, in CSRI Summer Proceedings 2009, Z. Wen and S. S. Collis, eds., Sandia National Laboratories, 2009.

TECHNIQUES FOR MANAGING DATA DISTRIBUTION IN NUMA SYSTEMS

ALEXANDER M. MERRITT^{*} AND KEVIN T. PEDRETTI[†]

Abstract. MPI is the dominant programming model for distributed memory parallel computers, and is often used as the intra-node programming model on multi-core compute nodes. However, application developers are increasingly turning to hybrid models that use threading within a node and MPI between nodes. In contrast to MPI, most current threaded models do not require application developers to deal explicitly with data locality. With increasing core counts and deeper NUMA hierarchies seen in the upcoming LANL/SNL “Cielo” capability supercomputer, data distribution poses an upper boundary on intra-node scalability within threaded applications. Data locality therefore has to be identified at runtime using static memory allocation policies such as first-touch or next-touch, or specified by the application user at launch time. We evaluate several existing techniques for managing data distribution using micro-benchmarks on an AMD “Magny-Cours” system with 24 cores among 4 NUMA domains and argue for the adoption of a dynamic runtime system implemented at the kernel level, employing a novel page table replication scheme to gather per-NUMA domain memory access traces.

1. Introduction. Scalability plays an important role in high-performance computing. Large-scale simulations of nuclear reactions, nuclear decay, climate modeling, fluid dynamics and combustion all require greater precision and reduced time-to-solution to achieve accurate and timely predictions. In order to achieve this goal hardware and software must scale well together. With the onset of exascale computing current parallel programming models approach their limits in scalability as supercomputing hardware transitions from distributed-memory single-core processor architectures to hybrids of distributed-memory and shared-memory, heterogeneous and accelerator-supported systems.

Our work is focused on the architecture of the upcoming LANL/SNL “Cielo” capability supercomputer, which consists of AMD Opteron “Magny-Cours” non-uniform memory access (NUMA) processors. In contrast to previous SNL systems such as ASCI Red and Red Storm, Cielo’s hardware exhibits greater complexity within the node and processor itself, adding more cores and greater variation in memory access latencies. Non-local memory accesses comprise multiple levels of non-uniformity, incurring different penalties depending on the path traveled. Figure 1.1 illustrates our 24-core 4 NUMA domain dual-socket Magny-Cours experimental system. Off-chip diagonal HyperTransport links exhibit the largest latencies and are 8-bit wide, which is half as wide as all other processor interconnects in the system. Each Cielo compute node is similar to our test system except that Cielo uses 8-core processors instead of 12-core, and that the Cielo cores operate at 2.4 GHz instead of 2.2 GHz on our test system. As parallel applications become increasingly hybrid—utilizing threaded programming models combined with message passing, for example—performance degradation from inadequate intra-node data distribution on this architecture becomes more pronounced.

In this paper we discuss the hybridization of parallel programming models and how they perform in light of this new architecture, focusing on the evaluation of current intra-node approaches to data distribution that attempt to minimize the influence of NUMA latencies in multithreaded applications. We demonstrate that these approaches are not adequate to address losses in performance and additionally require developers to modify their application code. We argue for the use of a dynamic runtime system within the operating system to identify data access patterns of an application and use this information to redistribute data, avoiding code changes and user intervention current approaches require.

^{*}Georgia Institute of Technology, merriitt.alex@gatech.edu

[†]Sandia National Laboratories, ktpedre@sandia.gov

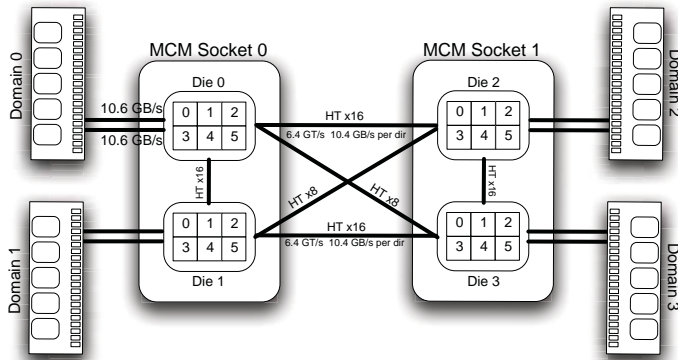


FIG. 1.1. Two AMD Opteron 6174 processors.

1.1. Parallel Programming Models. Programming languages have evolved along with the hardware on which they run. MPI [1] is a library-based message-passing extension to existing languages, such as C and Fortran. This model is a good fit for the distributed-memory architectures of supercomputers. Parallel units of execution called *ranks* exist in their own address space, each occupying one CPU core per compute node. In this model, communication and data sharing are explicit and must be known to the application developer¹ enabling him or her to finely tune algorithms for minimizing communication overheads. This advantage of MPI is, however, also its drawback: scaling applications to the extreme of exascale computing with hundreds of millions of processors [12] detracts from programmer productivity in addition to making debugging difficult. More limitations on scalability arise with the introduction of modern supercomputing hardware: higher core counts—more ranks—within a node increase memory consumed by global state replication and increase communication, causing contention on processor interconnects. Furthermore, a recent research study [14] compared the performances of message-passing and threaded programming implementations of sparse matrix-vector multiplication code, demonstrating that threaded models perform better on SMP hardware. These are all motivations for relaxing the trend of running “MPI everywhere”.

Thread-based parallel programming models such as OpenMP [2] and Pthreads operate in a single address space, avoiding the overheads of MPI. One address space allows global access to shared state and communication to be achieved through shared variables. Compared to other models, OpenMP is an automated parallelization tool designed to move the burden of explicit parallelization from the programmer to the compiler using simple `#pragma` directives in code. While combining threaded and message-passing programming models for intra-node and inter-node parallelization may show improvements [11], support for threading models on SMP NUMA systems is still immature. OpenMP was designed assuming homogeneous processing and uniform memory access architectures (UMA), but this is no longer true as commodity processor technologies are becoming increasingly NUMA. Without data replication, distribution is proving to be an inhibitor on software scalability with the advancement of parallel hardware present in the Cielo supercomputer.

¹Programmer and developer are used interchangeably in this paper.

1.2. Related Work. Current approaches take advantage of the virtual memory subsystem that the operating system kernel provides [5], use hardware counters provided by the CPU [13] as well as combine techniques at the user-level [3, 4].

User-level support for data distribution [3] combines knowledge from both a NUMA-aware memory manager and custom thread scheduler, provided as an extension to OpenMP to best minimize remote memory references. As with our work, this research advocates a *dynamic* runtime system that can adapt to changes in thread-data affinities throughout an application's execution. In contrast to our proposed technique for managing data distribution, this work still requires applications to use a supplementary API where ours aims for an application-independent implementation.

2. Evaluation. In this section we discuss recent research in literature and their effects on relevant micro-benchmarks at Sandia as our motivation for runtime analysis.

2.1. Background. We begin by giving some background on virtual memory support and use that as discussion for the various techniques in the literature.

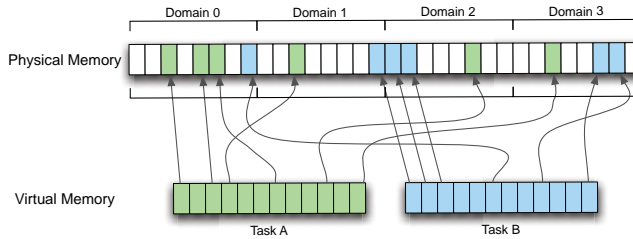


FIG. 2.1. Virtual memory mapping.

Hardware and operating system support for virtual memory allow for flexibility when designing models for data distribution. Each process is allowed to believe it has complete control over the entire system—full access to all of memory and the processor. In figure 2.1 we see the state of two processes at a given point in time. Memory (both virtual and physical) are divided into equal-sized regions called *pages*. When a process performs a memory operation such as a load or a store on data not present in memory, the hardware traps the faulting process and automatically transfers control to the operating system. The operating system then loads an entire page of data from an external source into an empty page in memory and establishes a translation. Each translation's state is represented by a structure in memory maintained by the kernel—called a *page table*—containing among other information, protection and access bits. Applications run as a combination of processes and threads, unaware of this mechanism that controls the distribution of its data in real hardware.

Within a NUMA system, all processors and memory regions are divided into *domains*. A domain is defined as a set of processors or cores and a region of memory between which is the lowest latency. Figure 2.2 depicts normalized latency costs of accessing all regions of memory for domain 0 in both an UMA and four-domain NUMA environment. Should a process be scheduled to execute in domain 0, any data accesses to domain 3 would incur the highest latency.

In the following subsections we review micro-benchmarks and their memory access patterns, current techniques examined, finally concluding with a discussion on the impact of these techniques as motivation for a dynamic runtime memory migration system.

2.2. Benchmarks. In this subsection we discuss two micro-benchmarks used to evaluate current approaches to data distribution, describe them in terms of *phases* and how current

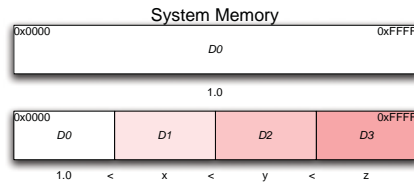


FIG. 2.2. UMA vs NUMA: memory latencies relative to domain 0.

approaches affect these phases. Both micro-benchmarks model common characteristics seen in scientific code at Sandia and are used as the foundation for further study. Benchmarks designed around MPI rewritten to use OpenMP have all explicit data movement removed.

We define a phase in a multithreaded application to mean an interval of time within which data access patterns remain deterministic or constant for each thread, within some threshold. A change in the number of threads also constitutes a phase change as data access characteristics must either be established for newly-spawned threads, or forgotten with fewer threads.

The use of a dynamic binary instrumentation tool called Pin [9] allows us to capture all instructions of an application that access memory. With this information we are able to visualize the data access patterns for both micro-benchmarks.

2.2.1. Evaluation System. Our evaluation system is a single shared-memory system with two AMD Opteron 6174 “Magny-Cours” processors. Each is a multichip module (MCM) containing two processor dies each with six symmetric cores running at 2.2 GHz, two memory controllers rated at 10.6 GiBps and a local subset of system memory. All four dies are connected with 10.4 GiBps HyperTransport (HT) links in each direction, forming a complete graph. All HT links are 16-bit wide except for two 8-bit wide diagonal crosslinks connecting domains 0-3 and 1-2. Figure 1.1 illustrates this design. Four memory and processor domains are available, each with 8 GiB of memory. Our study of this processor is significant as it forms the basis of the upcoming Los Alamos/Sandia National Labs “Cielo” capability supercomputer.

Results from current approaches were obtained on this system running RHEL 5.4 with a vanilla 2.6.27 kernel patched with support for the next-touch [5] memory policy.

2.2.2. STREAM. STREAM [10] is a memory bandwidth micro-benchmark parallelized with OpenMP. Each thread executes multiple kernels over its subset of data within three global arrays. Two kernels are represented in our data, “copy” and “triad” illustrated in figure 2.3. STREAM was configured to use 333 MiB-sized arrays to minimize caching effects (our test system has 20 MiB of effective last-level cache ²). Additionally, array elements are only accessed within parallel regions to create high affinities between threads and their data.

Kernel	Code
<i>Copy</i>	<code>c[j] = a[j]</code>
<i>Triad</i>	<code>a[j] = b[j] + scalar * c[j]</code>

FIG. 2.3. STREAM micro-kernels.

²While the system has 24 MiB of L3 cache, the HT Assist optimization uses 4 MiB of L3 to store directory information.

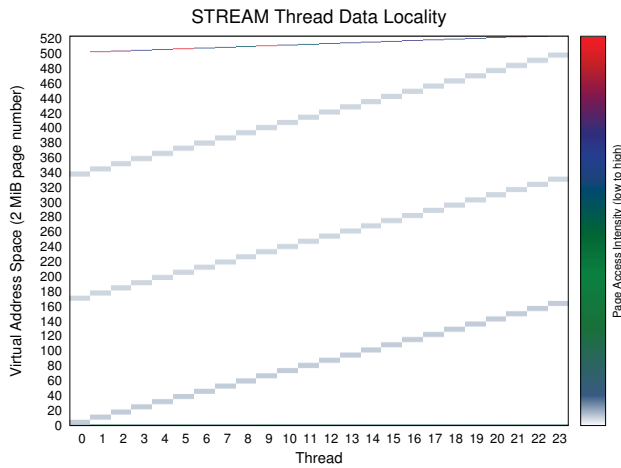


FIG. 2.4. *STREAM per-thread data locality (entire execution).*

Each of the three arrays are visible in figure 2.4 clearly indicating the regular access patterns of all threads. A fourth line towards the end of its virtual memory space represents the result vector storing timing measurements. STREAM exhibits this behavior throughout its entire execution.

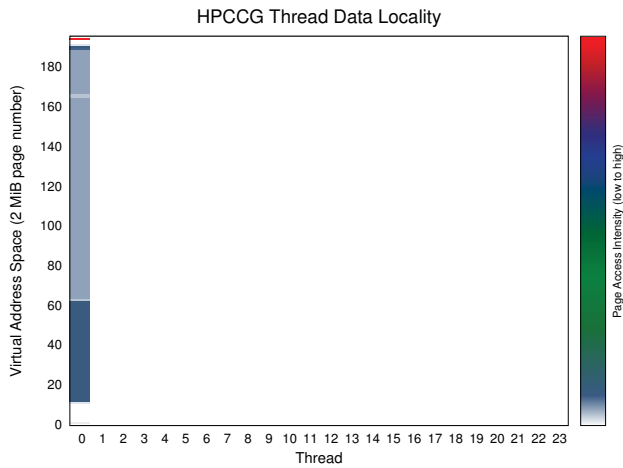
2.2.3. HPCCG. HPCCG [6] is a sparse-matrix vector multiply application with parallel implementations in both MPI and OpenMP. HPCCG allocates multiple arrays of data before entering repeated iterations of its parallel sections where, like STREAM, each thread performs work on a subset of data. Two phases have been identified in this micro-benchmark, both visualized in figure 2.5 for the OpenMP implementation.

Figure 2.5(a) depicts the master thread allocating and initializing all data. Figure 2.5(b) depicts the second phase of HPCCG's execution, wherein multiple parallel sections perform work on subsets of the overall data. The majority of work performed by this micro-benchmark is within this phase. Affinities between threads and data are strong but different in the two phases.

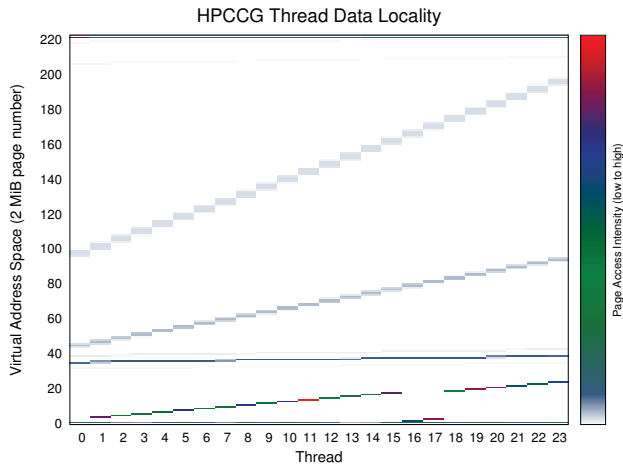
2.3. Techniques. In this subsection we discuss current approaches to data distribution at or below the operating system level and investigate their advantages and disadvantages as they apply to application phases seen in STREAM and HPCCG.

2.3.1. First-Touch. This is the default policy in the Linux kernel. It is not a solution to the NUMA problem per se, but rather a means to enable memory allocated by applications to be local to the domain in which they are executing. The Linux kernel memory allocator maintains a pool of memory for each domain in the system. This policy has no means to assist applications exhibiting strong phase changes, as data is never moved once allocated. Applications must be designed accordingly to minimize remote memory accesses, requiring that all data be touched first by the thread using it the most.

2.3.2. Memory Interleaving. Memory interleaving allows for an even distribution of memory over NUMA domains at the page and cache line granularity. Two configurations were available to us for experimentation and were simple to configure; these are presented in figure 2.6. Figure 2.6(a) illustrates operating system control over an application's virtual memory pages. The kernel memory allocator maintains pools of memory for each domain,



(a) Phase 1.



(b) Phase 2.

FIG. 2.5. HPCCG per-thread data locality.

cycling among them when creating translations from the virtual memory space of an application. Accessing virtual memory linearly physically iterates over all available NUMA domains at a page granularity. Figure 2.6(b) illustrates a second method of interleaving. Here the memory controllers are configured by the BIOS to modify its mapping of physical addresses to machine memory, interleaving them among the NUMA domains. In contrast, this method operates at the granularity of a cache line, is transparent to the operating system and allows for a finer distribution of memory among the domains (interleaving the kernel address space as well as all application address spaces). Both methods distribute memory such that the chance of accessing either a page or cache line locally is $\frac{1}{D}$ for D domains.

Page interleaving support is provided by the `numactl` command line tool in Linux. It allows for various NUMA-related operations on processes, such as memory and domain execution pinning, CPU core pinning and memory interleaving.

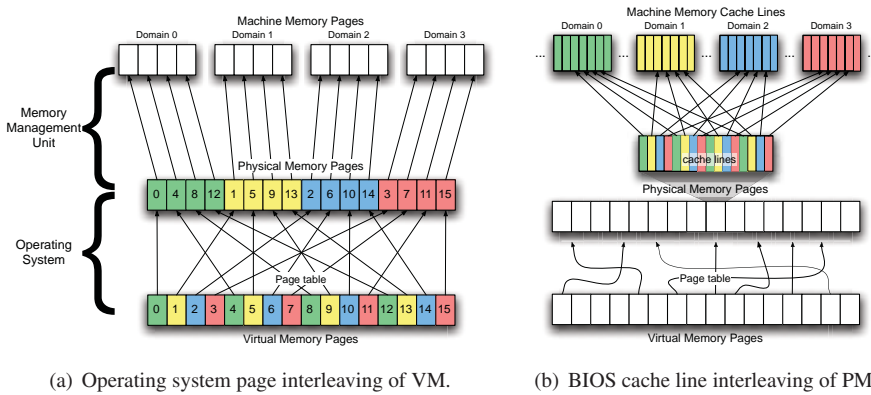


FIG. 2.6. Two levels and granularities of memory interleaving.

2.3.3. Next-Touch. Next-touch is a memory policy implemented in kernel space. In this scheme, an application flags regions of memory that should be migrated to the NUMA domain of the next thread that accesses the region. By manipulating protection bits in the page table we force the hardware to intercept memory accesses, migrating their pages before resuming process execution. Recent work [5] provides this implementation as a patch for the 2.6.27 Linux kernel. The patch adds additional flags to the `madvise()` system call enabling user-space applications to inform the kernel's memory subsystem to modify the appropriate protection bits on a given range of pages. On subsequent touches the hardware traps the executing process and migrates pages to whichever domain the process is executing on.

Static code analysis enables the programmer to identify when page migrations are needed. One difficulty with this approach is ensuring that no thread other than the one intended touches the pages it will most heavily use. This assumes the appropriate place to call `madvise()` can be determined through static analysis. Furthermore, the data access pattern among threads may change throughout an application's lifetime. For this approach to be effective each application phase must be identified and a call to `madvise()` inserted.

Support for this feature exists in the Oracle Solaris 9 operating system [13], but currently has not been accepted into the mainstream Linux kernel codebase.

2.4. Results. In this subsection we present and compare performance results for the next-touch, first-touch and memory interleaving strategies with respect to execution phases seen in STREAM and HPCCG. Results are shown in figures 2.8 and 2.7. Points plotted in each of the three graphs comprise an average of 20 executions with error bars illustrating one standard deviation. We used two strategies for pinning threads, round-robin ("Pin RR") and ascending ("Pin Asc"), both tied to the Linux-logical core IDs.

STREAM performance data was collected from a combination of operating system thread scheduling and manual thread pinning in addition to the first-touch and page interleave strategies. Red curves in figures 2.7(a) and 2.7(b) show the performance of STREAM with no modifications and no runtime policies. Variability is high due to the non-deterministic behavior of the 2.6.27 Linux scheduler and its inability to identify affinities between data and threads. It may repeatedly schedule multiple threads for execution on the same domain before attempting to reduce oversubscription. This behavior causes threads to access their data from various domains, causing scattered first-touch allocations to occur. Should the scheduler know to not relocate threads, first-touch would show the best performance. Forcing thread pinnings reproduces this behavior and indeed shows the best performance, represented

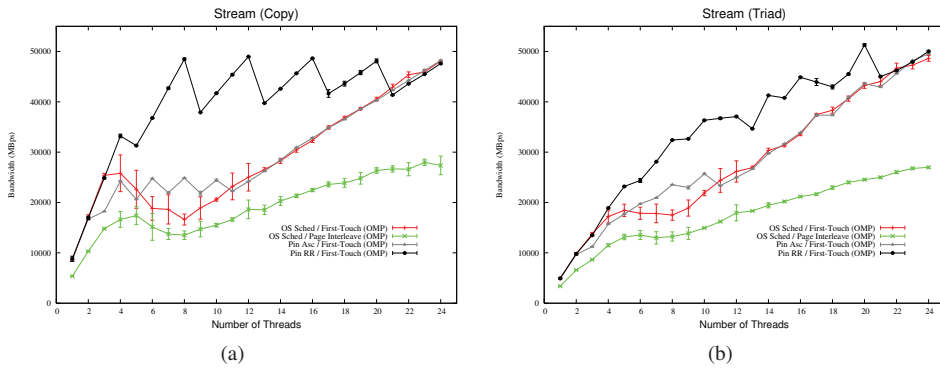


FIG. 2.7. Performance of STREAM applying current data distribution techniques.

by black curves in both plots.³ Interleaving pages reduces performance considerably, as the probability of accessing local data is reduced to 25% from near 100% when compared with thread pinning and first-touch.⁴ Pinning threads in ascending order by Linux-logical core IDs shows a plateau (gray curve) as the first twelve core assignments alternate between domains 0 and 3, the latter twelve between domains 1 and 2. This demonstrates that assumptions cannot be made regarding the logical-to-physical mapping of cores by the operating system. Next-touch results were not included as first-touch and thread pinning give the same result. Curves in figure 2.7(b) are more linear due to a larger percentage of work coming from computational overhead. Overall trends are however still visible.

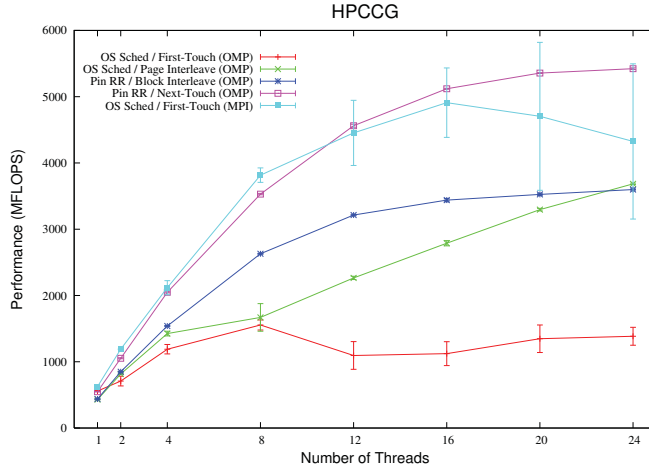


FIG. 2.8. Performance of HPCCG applying current data distribution techniques.

HPCCG performance data was collected from the same policy combinations used for STREAM, with the additional cache line interleaving technique and results from the MPI implementation. The red curve in figure 2.8 is the same policy as the red curve in figure 2.7,

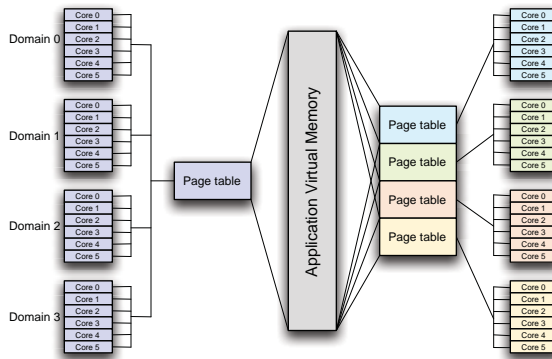
³We do not yet know the cause of the saw-tooth shape in the curve.

⁴A bug was discovered in the PGI OpenMP runtime library, preventing thread pinning while interleaving pages. Unfortunately this prevents accurate comparisons as we cannot eliminate the scheduler's nondeterministic behavior. This bug was reported to PGI and promptly fixed; it will be available in the next release of the PGI compiler suite.

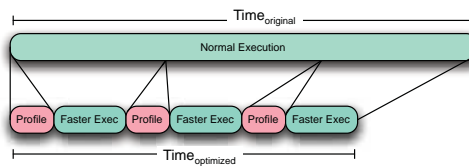
with no code modifications or runtime policies. This behavior is attributed to the changes in the data access patterns mentioned in section 2.2.3: the existence of phase 1 forces the first-touch policy to allocate all memory pages on single memory domain. Transitioning to phase 2 shows an increase in threads, all of which are relocated by the scheduler to CPU cores in other domains, to minimize oversubscription. In doing so memory references for 75% of all threads become almost entirely non-local. In contrast, an MPI implementation scales much better due to data duplication, yet performance tapers off with many threads showing large variations most likely attributed again to the nondeterministic behavior of the Linux scheduler. Without data distribution in a threaded environment first-touch prevents applications with phases similar to HPCCG from scaling on NUMA hardware.

Having identified both phases of HPCCG, we were able to effectively use next-touch by placing appropriate calls to `madvise()` at the end of phase 1. Combined with thread pinning this method showed the best overall performance as memory pages were migrated to domains in which the accessing threads were executing, enabling nearly all memory references to be local. Page and cache line interleaving also improve performance as expected, with cache line interleaving showing slightly higher performance due to the smaller granularity.

3. Future Work. An application's data access patterns change throughout execution. Approaches discussed in prior sections function in a one-shot manner, continuously apply the same rule or require user interaction. We therefore propose a dynamic runtime system for monitoring an application's memory access patterns from within the kernel. Active monitoring allows the operating system to observe affinities between an application's threads and pages in memory, migrating pages to reduce remote memory references.



(a) Multiple page tables for monitoring page access rates.



(b) Effect of runtime profiling on an application.

Fig. 3.1. Proposed dynamic runtime system for page migration within the kernel.

An operating system typically allocates one page table per process. By using one page table per domain per process we will be able to capture where accesses originate and what

regions of memory they reference, as seen in figure 3.1(a). The appropriate page table is installed on a context switch, and access bits within page table entries updated by the hardware will be read to monitor an application's data access patterns. A kernel thread will periodically scan these entries to observe page access patterns, identify frequent accesses of non-local memory and migrate pages accordingly. Frequent scanning will be required as only one access bit is available per page, potentially increasing profiling overhead. To prove itself advantageous, our approach must ensure the savings in execution time of the application are greater than the combined cost of data migration and active profiling—figure 3.1(b). We argue that this approach will allow for a more flexible and customizable solution to the problem of data distribution on NUMA hardware.

Overheads introduced by this mechanism can be reduced if implemented in a light-weight kernel such as Kitten [7]. Kitten creates a complete linear mapping of all virtual memory addresses on process creation, in other words, pre-populating all page tables. Knowing the locations of all last-level page table entries can reduce the execution overhead introduced from frequent access bit scanning by avoiding full page table traversals. Kitten furthermore supports the use of large pages to map virtual memory. Enabling this will reduce the height of each page table and lower the number of page table entries needed. Using our proposed approach, each process will consume four times the amount of memory needed to store their address translations; large pages will reduce this footprint.

Our approach is a mechanism that will require various policies to show any benefit. Varying the profiling frequency and defining intervals for page migration and the meaning of “heavy” page access will have to be adjusted to determine which combinations give the best results across a range of application phases. Research in the early nineties examined this [8], but on different hardware and with a different application domain. It was determined that no single policy proved beneficial across all kernels in their benchmark suite. Our goal is to re-examine this on modern hardware across a more appropriate set of kernels.

Future support in hardware may include widening the access bit field in each page table entry and modifying the processor to treat this field as a saturating counter instead of a bit flip. While beyond the scope of this research, it would reduce the execution overhead from profiling, by lowering the frequency at which page table entries need to be scanned.

4. Conclusion. In this paper we examined several existing techniques for managing data distribution in a multicore NUMA environment, the basis for the upcoming “Cielo” capability supercomputer. As scientific applications are becoming increasingly hybrid, incorporating threaded programming models within nodes, support for data distribution becomes a limit on intra-node scalability. We demonstrated the static nature of current techniques, requiring time-consuming code modifications or user intervention. With tools developed to visualize application data access patterns we are better able to identify phase changes in thread-data affinities, enabling a more complete understanding of our application domain. Combined with an evaluation of our proposed kernel-level dynamic runtime technique we demonstrate the need for an invisible and adaptive data distribution model, empowering systems software to continue to scale as we approach exascale computing.

REFERENCES

- [1] *Mpi*. <http://www.mpi-forum.org>.
- [2] *Openmp*. <http://www.openmp.org>.
- [3] F. BROQUEDIS, O. AUMAGE, B. GOGLIN, S. THIBAUT, P.-A. WACRENIER, AND R. NAMYST, *Structuring the execution of openmp applications for multicore architectures*, in Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on, 19-23 2010, pp. 1–10.

- [4] F. BROQUEDIS, N. FURMENTO, B. GOGLIN, R. NAMYST, AND P.-A. WACRENIER, *Dynamic Task and Data Placement over NUMA Architectures: an OpenMP Runtime Perspective*, in International Workshop on OpenMP (IWOMP), Dresden Allemagne, 2009.
- [5] B. GOGLIN AND N. FURMENTO, *Enabling high-performance memory migration for multithreaded applications on linux*, in Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on, 23-29 2009, pp. 1–9.
- [6] M. HEROUX, *Hpccg*. <http://bec.syr.edu/hpccg.html>.
- [7] J. LANGE, K. PEDRETTI, T. HUDSON, P. DINDA, Z. CUI, L. XIA, P. BRIDGES, A. GOCKE, S. JACONETTE, M. LEVENHAGEN, AND R. BRIGHTWELL, *Palacios and kitten: New high performance operating systems for scalable virtualized and native supercomputing*, apr. 2010, pp. 1–12.
- [8] R. P. LAROWE, JR. AND C. SCHLATTER ELLIS, *Experimental comparison of memory management policies for numa multiprocessors*, ACM Trans. Comput. Syst., 9 (1991), pp. 319–363.
- [9] C.-K. LUK, R. COHN, R. MUTH, H. PATIL, A. KLAUSER, G. LOWNEY, S. WALLACE, V. J. REDDI, AND K. HAZELWOOD, *Pin: building customized program analysis tools with dynamic instrumentation*, in PLDI '05: Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation, New York, NY, USA, 2005, ACM, pp. 190–200.
- [10] J. D. MCCALPIN, *Memory bandwidth and machine balance in current high performance computers*, IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter, (1995), pp. 19–25.
- [11] L. SMITH AND M. BULL, *Development of mixed mode mpi / openmp applications*, Sci. Program., 9 (2001), pp. 83–98.
- [12] P. THIBODEAU, *Scientists, IT community await exascale computers*, December 2009.
- [13] M. M. TIKIR AND J. K. HOLLINGSWORTH, *Using hardware counters to automatically improve memory performance*, in SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing, Washington, DC, USA, 2004, IEEE Computer Society, p. 46.
- [14] S. WILLIAMS, L. OLIKER, R. VUDUC, J. SHALF, K. YELICK, AND J. DEMMEL, *Optimization of sparse matrix-vector multiplication on emerging multicore platforms*, in SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing, New York, NY, USA, 2007, ACM, pp. 1–12.

INTEGRATING ROUTER POWER MODELS INTO THE STRUCTURAL SIMULATION TOOLKIT

KEVIN D. THOMPSON[¶], ARUN F. RODRIGUES^{||}, AND MING-YU HSIEH^{**}

Abstract. As the Structural Simulation Toolkit (SST) grows to encompass a large assortment of hardware components, the need to model power consumption becomes very clear. Networking hardware, specifically routers, consumes a very measurable amount of power. SST needs an interface to capture this data and report it to the user in a simple and friendly manner. We intend to utilize existing power modeling toolkits, specifically McPAT and ORION 2.0, into SST's router models in order to get a reasonably accurate idea of the power being consumed.

1. Introduction. The Structural Simulation Toolkit (SST)[4], is used to explore architectural options when designing the next generation of supercomputer. Its highly modular framework allows it to support a wide range of customized models and system parameters. SST allows hardware and software to be developed and modified separately, neither limiting the other. It then links the programming model to the hardware organization and relays valuable performance data to the user who needs to design a very powerful, yet efficient and cost effective, system.

Given that power is now the major limitation of computers[1], SST needs to have a way to monitor the approximate power usage of a certain application being run on the proposed architecture. The team here at Sandia is working on integrating accurate power modeling to each main component of the toolkit. Interconnection networks have been estimated to consume roughly 20-40% of a system's total power budget[5]. We need a way to monitor and limit that power consumption. When considering cost/performance trade-offs, power models integrated into the networking components of SST will allow the developer to consider how varying the topology, port count, clock rate, etc. would affect overall consumption of power.

We have identified two well-respected power modeling tools to use with SST: McPAT (Multicore Power, Area, and Timing), a general purpose modeling framework from HP Labs[3], and ORION 2.0 (Open Research Infrastructure for Optimizing Networks), a networks-on-chip (NoC) specific power model suite from Princeton University[2]. In this paper, we will discuss a little background on these power models and why they fit our needs (Section 2). We'll then go over the design and integration of the power models into SST and the network components (Section 3) and discuss the initial results obtained through simulations thus far (Section 4). Finally, we will address future work that still needs to be completed before final production (Section 5).

2. Power Models. The two power models we integrated to measure network power were McPAT[3] and ORION 2.0[2]. Both of these power modeling tools have been written to account for networks-on-chip (NoC) power. McPAT is a general power model that computes power, area, and timing for multicore processors; NoC power modeling is only one fraction of McPAT's modeling capabilities. For this reason, McPAT is a very attractive power modeling tool; not only can we use it for this research, but it will be (and has been) used to model power in many more of SST's components. ORION, on the other hand, is designed specifically for power analysis of NoC. It utilizes many low-level customizable parameters and we anticipate it providing very specific and accurate results. It should be noted that McPAT does account for short-circuit power, which is ignored by ORION. We may try to add some short-circuit power modeling capabilities to ORION in the future.

[¶]Electrical & Computer Engineering, New Mexico State University, kevin@nmsu.edu

^{||}Scalable Computer Architecture, Sandia National Laboratories, afrodr@sandia.gov

^{**}Scalable Computer Architecture, Sandia National Laboratories, myhsieh@sandia.gov

2.1. McPAT. McPAT has been used as the primary power modeling toolkit for SST. McPAT offers accurate multicore and manycore modeling and can “accurately scale circuit models into deep-submicron technologies.”[3] McPAT models dynamic, static, and short-circuit power to fully encompass all sources of power dissipation. It has an inherent XML-based interface which makes it very easy to parse the high- and low-level parameters specified in each component’s XML file. McPAT’s framework is presented as a block diagram shown in Figure 2.1. The data is input through an XML interface which McPAT’s core reads and analyzes. Timing, area, and power data are then output to the user. This data can even be output in real-time so that a sophisticated simulator can make real-time adjustments as necessary. In this paper, we are only concerned with the power data output. As seen in the diagram, power dissipation is broken into its three main sources: dynamic, leakage, and short-circuit power. This separation helps us to determine what parameters we should change to optimize power consumption. In Section 4, we will present some sample output data.

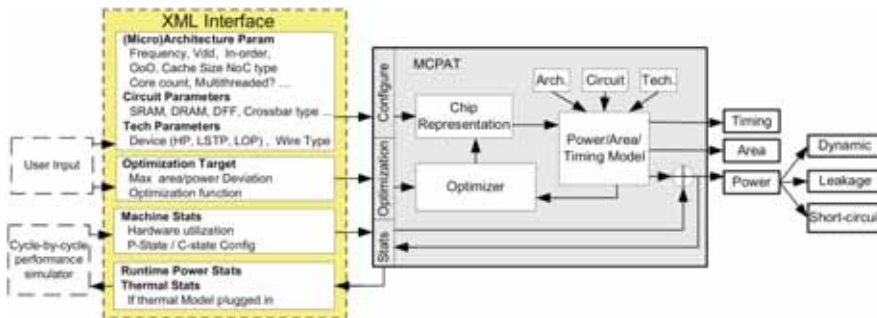


FIG. 2.1. Block diagram of the McPAT framework. [3]

McPAT breaks down the router model into its main components: flit buffers, arbiters, and crossbars are the big ones. As seen in Figure 2.2, McPAT uses a traditional router with four stages to its pipeline: routing computation, virtual channel allocation, switch allocation, and switch transversal. The parameters defined by the user determine the size and number of the router’s main components. SST sends McPAT a signal letting it know every time the router is accessed. Using the predefined parameters, it then calculates the power used by each hardware component. A more detailed discussion of McPAT’s power analysis, including formulas and figures can be found in its reference manual[3]. McPAT was validated against Niagara, Niagara2, Alpha 21364, and the Xeon Tulsa processors. The difference between its models and the reported data was between 10-20%.

2.2. ORION. ORION had not previously been incorporated into SST as McPAT had. We identified it as a good “second opinion” power modeling tool to use with SST’s networking components. ORION has been proven to work with many-core chips, such as Intel’s 80-core Teraflops chip as well as their Scalable Communications Core (SCC) chip. With so many cores on a single chip, it is obvious that interconnect networks are playing a major role in future systems. The other interesting aspect of ORION is that its developers have included “a semi-automated flow that automatically updates its models as new technology files become available.”[2] This feature allows ORION to stay up to date with minimal labor for the user. Since SST’s main role will be helping to facilitate the design of future supercomputers, we need power modeling tools that are always current and up to the task at hand. Figure 2.3 shows ORION’s modeling framework. Like McPAT’s framework in Figure 2.1, it outputs area and dynamic and leakage power (note that it doesn’t output timing or short-circuit power). ORION was validated to within 7% and 11% error of the Intel 80-core and SCC

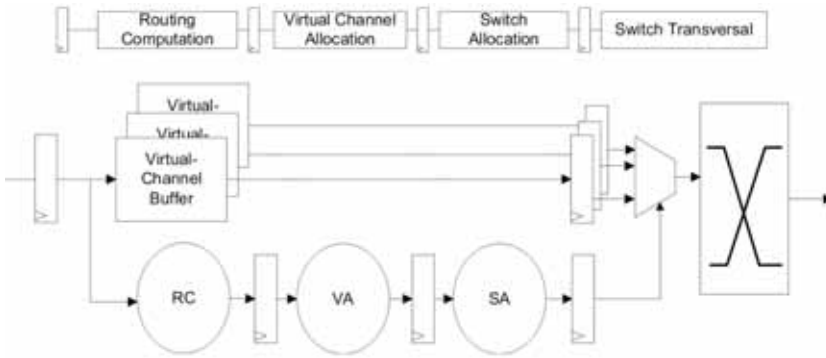


FIG. 2.2. McPAT's router model. [3]

chips, respectively.

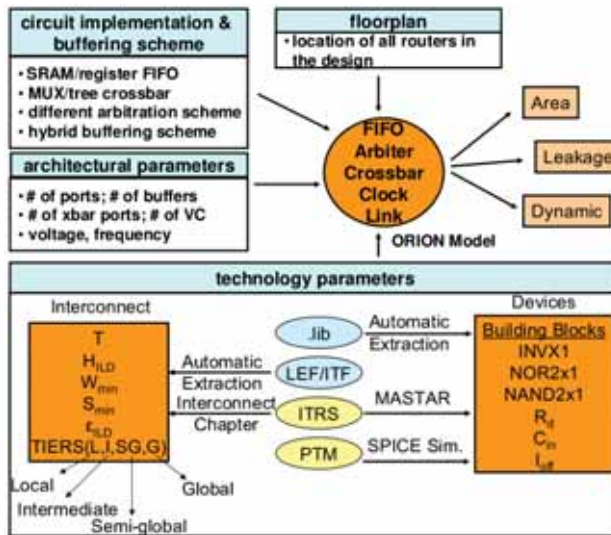


FIG. 2.3. ORION 2.0 modeling methodology. [2]

3. Design and Integration. Since McPAT had been previously integrated for other purposes, our main design tasks were focused on ORION and the router model within SST. We'll discuss briefly how McPAT was customized, as well as the process required to get ORION working with SST. We'll then go over the additions to get the router model to call and utilize these libraries.

3.1. Integrating McPAT. McPAT's source files were modified to make it possible for it to communicate effectively with SST. Functions were added in the format `SSTreturnXXX()`, where `XXX` is the name of the subcomponent in McPAT's core. These are the functions responsible for returning McPAT component objects with unit energy to SST. It was also modified to get its data from the XML file of the component being monitored as well as the original McPAT XML file that keeps default parameter values and instructions. There is a power API within SST that manages all of the interactions between the component being monitored and the power library being used. It is the middle-man controlling the transaction.

3.2. Integrating ORION. ORION had not been previously implemented, so we needed to modify it from scratch. It is written in C, while McPAT and SST are mainly coded in C++. There are several functions used to keep track of the power being used by the router model. The first one to be called is `SST_SIM_router_init()` which sets up the parameters in ORION based on a combination of user input and default values. Next, there are two avenues to take. The first is to simply run `SST_SIM_router_stat_power()` which will, based on the previously set parameters, compute the average power estimated to be used by the model in question. The second one uses four functions: `SIM_buf_power_data_read()`, `SIM_buf_power_data_write()`, `SIM_crossbar_record()`, and `SIM_arbiter_record()`. Every time a buffer read/write, crossbar activity, or arbiter activity occurs in the model, the corresponding function is called and power data is accumulated.

3.3. Implementation in the Router Model. Now that the power libraries have been set up, we need to link the router model with SST's power API to drive the calculations. SST uses an XML file generated by the router code to run the simulation. We modified this generation of the XML file to include the proper parameters required by McPAT and ORION to run a simulation. We set up a test router that was configured with the initial parameters outlined in Table 3.1. The parameters shown are only the most critical. There are a huge number that can be customized in addition to these. We will vary these parameters to obtain a graph that will be presented in Section 4. Finally, the router model needs to communicate its settings at run-time to the power libraries. This is accomplished by sending the appropriate data to the power API which forwards it to McPAT or ORION. We will briefly discuss the critical lines of code and their operation.

TABLE 3.1
Test-router parameters used.

clock rate	1.0 GHz
supply voltage	1.2 V
topology	ring
input ports	8
output ports	8
flit bits	128
virtual channels per port	2

The code shown in Figure 3.1 is called during every router event. The function `resetCounts()` is called to reset all of the counts to zero to assure accurate accumulation. The integer `mycounts.router_access` defines how many times the router is accessed. In our model, it is accessed twice: once to receive a flit, and once to send it on to its destination. `getPower()` sends the data to either McPAT or ORION (whichever is specified in the XML file) where it is analyzed, power is computed, and data is returned to SST. Finally, `regPowerStats()` records the data returned. The total power, leakage power, and dynamic power, are accumulated at every call, until they are reported to the user at the termination of the simulation.

```
power->resetCounts(&mycounts);
mycounts.router_access=2;
pdata = power->getPower(this, ROUTER, mycounts);
regPowerStats(pdata);
```

FIG. 3.1. *Sample calls from the router model.*

4. Results. Since our model integration has not yet been validated, we will simply present some sample output as it would be seen and applied by the user. We'll run each simulation varying the number of ports and the flit width to demonstrate how power consumption changes by varying critical parameters. Except for the parameter being varied, the rest of the parameters will be constant as listed in Table 3.1.

4.1. Sample output. We ran SST's router model using the parameters defined in Table 3.1, with McPAT's power libraries. The results are shown in Figure 4.1. Our McPAT simulation breaks the total power down into leakage and runtime power.

```
current total power = 1.19377 ± 5.96883e-06 W
leakage power = 0.479309030943142 W
runtime power = 0.714458 3.57229e-06 W
```

FIG. 4.1. Router power dissipation output by McPAT.

We also ran the router model with the same parameters using ORION's power libraries. The results are shown in Figure 4.2. ORION breaks its power dissipation down into five subcomponents so that the user has a better idea of where the most power is being consumed and of what changes to the design would be best to make.

```
Buffer:0.195698 W
Crossbar:0.730277 W
VC_allocator:0.00366383 W
SW_allocator:0.0119834 W
Clock:0.0932492 W
Total router power:1.03487 W
```

FIG. 4.2. Router power dissipation output by ORION.

These results are each one of several data points used to produce the comparison graphs in Figure 4.3. All parameters were left constant except for the number of ports and the flit width. In graph 4.3(a), the flit width was left at 128 bits for all iterations while the number of input/output ports was varied from two to twelve. By increasing the number of ports, power dissipation grows at a pretty steady rate. Designers will want to keep the number of ports and routers under control during the design process. SST allows a designer to make trade-offs during development. For instance, increasing the number of ports while decreasing the number of routers may ultimately result in less power consumption. In graph 4.3(b), the input/output ports were left at eight a piece, while the flit width was varied from 8 bits to 256 bits. This shows quite modest growth from 8 to 128 bits, but 256 bits shows a substantial leap in power dissipation. Obviously if designers are thinking of using a flit width over 64 bits, they need to really consider whether the performance boost would be worth the huge power increases. Valuable information can be gained by running these simple simulations.

The discrepancy between McPAT's and ORION's power estimations is quite small, and they both grow at about the same rate. This is a good sign. By using two totally different power models, and getting similar results, we are on our way to obtaining a good estimation of power dissipation (this is, of course, dependent upon the accuracy of the parameters fed to the models). Each has its advantage over the other. McPAT makes it easy to model more than just network components. By using CPUs with NICs and routers, we can simply call one power library and get an overall power dissipation report. On the other hand, ORION does a good job of breaking down each of the main router components and its parameters are much

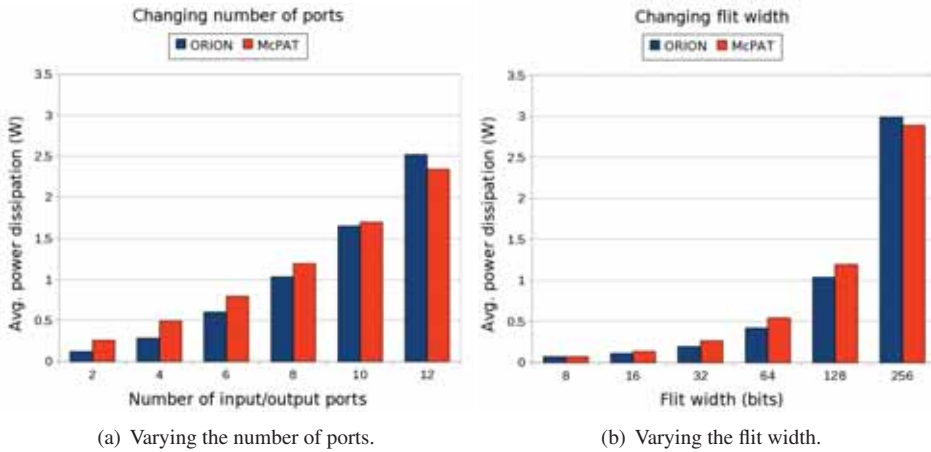


FIG. 4.3. ORION and McPAT power dissipation.

more customizable than McPAT's for potentially greater accuracy. When a detailed report of power dissipation in the router is needed, ORION will be the way to go. It can always be used in parallel with McPAT as an accuracy check.

5. Future Work. The power modeling calls within the router model will likely need to be further optimized. We can achieve this through validation steps. Our results have not yet been validated with a real model. Validation needs to be completed before the models can be officially released in a public version of SST. The plan is to obtain router parameters from real routers with published data. We can then run these parameters through our power API and find out how the accuracy of our integrated power models relates to what it should be. The results should be similar to those previously published for McPAT and ORION in [3] and [2] if everything was incorporated correctly. Another component closely related to the router is the network interface controller (NIC) model. It is also responsible for a large portion of the power consumed by the network and should be modeled accordingly. This area may also include the modeling of inter-router links which can consume up to 30% of total NoC power [3]. Both McPAT and ORION have the capabilities to model inter-router links. Link length should be known, so we'll need a good algorithm to properly calculate these lengths before we can model it accurately.

6. Conclusions. This work has paved the way for more detailed power estimation to be accomplished. Now cost/performance graphs can be more easily attained to determine and identify necessary trade-offs during initial design. Soon, SST will be publicly available to both academia and industry. It is highly anticipated to be a very useful and widely used tool in supercomputer design. Since future many-core systems will rely heavily on interconnection networks, designers will need a quick and easy way to determine their cost. This work has laid a foundation for those goals to be obtained. Using McPAT and ORION 2.0, we can get a quick and accurate idea of the power being consumed.

REFERENCES

- [1] J. L. HENNESSY AND D. A. PATTERSON, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, San Francisco, CA, 2007.

- [2] A. KAHNG, B. LI, L. PEH, AND K. SAMADI, *Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration*, in Proceedings of Design Automation and Test in Europe (DATE), Nice, France, April 2009.
- [3] S. LI, J. AHN, J. B. BROCKMAN, AND N. P. JOUPPI, *Mcpat 1.0: An integrated power, area, and timing modeling framework for multicore architectures*, tech. rep., HP Labs, 2009.
- [4] A. F. RODRIGUES, *The structural simulation toolkit*, September 2007. www.cs.sandia.gov/sst.
- [5] L. SHANG, L.-S. PEH, AND N. K. JHA, *Dynamic voltage scaling with links for power optimization of interconnection networks*, in Proceedings of the 9th International Symposium on High-Performance Computer Architecture (HPCA), 2003, pp. 91–102.

PROCESS LAYERS FOR DISCRETE EVENT SIMULATION OF COMPUTER SYSTEMS

CHAD D. KERSEY* AND ARUN F. RODRIGUES†

Abstract. Programming environments for discrete event simulation often provide a process-like abstraction for their users. A new class of parallel discrete event simulation kernels created for the large-scale simulation of computer hardware provide only a minimal set of primitives for interaction with the simulator. For certain problems, the ability to use processes greatly improves programmer productivity and code readability, at the cost of the lower performance and increased memory use imposed by the process implementation. In this paper, we introduce two implementations of processes for these frameworks, provide a case study to demonstrate the advantages and costs of using processes, and propose methods for mitigating their costs. Similar software layers have been in use for years to ease the programming of event driven real time systems and network servers, and it is our goal to provide similar capabilities for the high-performance simulation of computer systems.

1. Introduction. A multitude of purpose-built programming environments for discrete event simulations have emerged over the years, typically featuring processes as a fundamental abstraction. CSIM [12] and SimPy [13] exemplify pure process-oriented simulation environments. In both of these systems, the basic abstraction presented to the user is a collection of sequentially-executing communicating processes. Simulation time passes only while these processes are blocked and stands still while they are running. Common blocking primitives are waiting for a given amount of simulation time to elapse or waiting for an event to be triggered by another process.

A new class of parallel simulation kernels created for the large-scale simulation of computer systems and exemplified by the Manifold simulation kernel from Georgia Tech and the Structural Simulation Toolkit (SST) simulation kernel from Sandia National Laboratories [11], provide only a minimal set of primitives for interaction with the simulator, leaving implementations of more expressive constructs to other software layers. By leaving the implementation of processes out of the kernel, not only is the design of the simulation kernel simplified, but the programmer is given the freedom to choose the appropriate tool for their task. It is expected that multiple implementations of processes and other high-level abstractions will be used simultaneously to build large-scale simulations.

2. Background.

2.1. Discrete Event Simulation. The design of computer systems has been recognized as an appropriate problem domain for discrete event simulation since 1974 at the latest [10]. Through the use of accurate simulations, it is possible to demonstrate effects that only become apparent at the system level, when all of the components are combined, and evaluate solutions to performance limiting behavior.

A major challenge to the performance of the simulators themselves is implementing correct and efficient parallel discrete event simulation. As discussed by Fujimoto in [7], two classes of simulation methods exist: *conservative* approaches, limited by the look-ahead between parallel *logical processes* (which are distinct from the simulator processes we implement here!), and *optimistic* approaches, which detect causality violations after they occur and roll back.

Broadly, discrete event simulators can be categorized as *process*-based or *event*-based. A process in this sense is a component-level construct that is able to persist over a span of simulation time. Event handlers, on the other hand, are executed entirely within a single

*Georgia Institute of Technology, cdkersey@gatech.edu

†Sandia National Laboratories, afrodri@sandia.gov

time step. While processes are running, no simulation time passes, but there are a variety of blocking calls that they can make, during which simulation time can advance. The two most common of these are waiting on a condition variable to be triggered by an asynchronous event and delaying for a given period of simulation time.

These two blocking operations are available in discrete event simulation frameworks like SimPy and CSIM [13, 12], and also in the widely used VHDL [1] hardware description language. VHDL processes work the same way as SimPy or CSIM processes, but are not the only abstraction provided by the simulation environment; it is possible to build detailed models of hardware in VHDL without using processes. Similarly, the software layers for implementing processes presented here are available in addition to, not instead of, the ability to define individual event handlers.

2.2. SST and Manifold. SST and Manifold both approach parallel discrete event simulation conservatively, dividing the simulation into a set of composable *components* connected by *links*. In SST, these components are automatically partitioned between parallel threads of execution (MPI ranks), while in Manifold the partitioning must be manually performed by the user. Links are associated with a latency, which guarantees that the *lookahead* value, the amount of simulation time that one rank can run ahead of another rank, with no possibility of a causality violation, between any two ranks is the minimum of the link latencies between them.

The API for writing a component for either of these simulators is simply the implementation of a set of event handlers and a constructor. Though both simulators also provide a concept of a periodic *clock* in addition to events, this can be viewed as an optimization of the event scheduling system for components written with a time stepped paradigm. While these simulation kernels do not use processes in the sense of a single logical process per MPI rank, there is no notion of a process from the point of view of the component implementer.

SST, in development at Sandia National Laboratories [11], provides a parallel simulation kernel along with a set of *components*, which can be combined through the use of an XML system description language into a full simulation. The component graph is partitioned automatically across MPI ranks, which correspond to what Fujimoto referred to as *logical processes* [7], in order to maximize lookahead.

In order to provide edges representing lookahead in the component graph so that it can be partitioned to maximize lookahead, components in SST cannot arbitrarily schedule events in the simulator for any other component. A link must be defined in the SDL file between these components with a corresponding latency. This latency represents the amount of elapsed simulation time between an event being created and an event arriving at its destination component.

The Manifold project at Georgia Tech is similar in its scope and goals to SST, but uses slightly different semantics to simplify design. In Manifold, processes within the same rank may communicate by scheduling function calls on one-another directly, without the use of links. Only for communication between ranks are links necessary, with partitioning of components among ranks performed manually.

Both Manifold and SST share an event-based programming model. Event handlers (which much be configured as such in SST but can be any void function in Manifold) execute entirely within a single simulation time step. If an event handler cannot proceed in its task until a condition is satisfied, it must save its state in some known location from which the task will be resumed when the condition is satisfied. If its task requires a delay for a given period of simulation time, it must schedule a future event. Both of these tasks require splitting a logically continuous process among multiple events.

As a simple example, a Poisson process traffic generator written in an event oriented

manner might be implemented in pseudocode as:

```
void event_a_handler() {
    if (running) {
        generate_packet();
        schedule(exp_rn(MTBP), new event_a());
    }
}

constructor() {
    schedule(exp_rn(MTBP), new event_a());
}
```

An identical traffic generator implemented with processes might look like:

```
process_main() {
    pause(exp_rn(MTBP));
    while (running) {
        generate_packet();
        pause(exp_rn(MTBP));
    }
}

constructor() {
    spawn_process(process_main);
}
```

Notice that the control flow structure has been improved by allowing the use of a `while` loop for a repeated action, instead of requiring that the new events be manually scheduled as long as `running` is true.

SST imposes an additional requirement on component writers, that the components they create must be *serializable*; they must provide a way to save and restore their state to a byte stream. This is important for the largest-scale simulations to avoid the consequences of hardware failure during simulation. This requirement has far-reaching consequences in component implementation that impact the implementation of process layers for use with SST.

2.3. Event-Driven Programming. The limited component implementation API of SST and Manifold is similar to APIs seen in event-driven programming environments. Long used for embedded systems, UI-driven systems, and high performance servers, the event driven paradigm provides an efficient way for software to interact with asynchronous external events, advancing as long as there are events left to process. A common complaint among users of event-driven systems is that it is difficult to maintain state across a series of events, for instance to maintain a session when implementing a network protocol. Many solutions have been proposed to circumvent this while maintaining most of the efficiency of events [5, 6, 9, 14].

2.4. QSim and Related Components. QSim is a library developed as a layer on top of a modified QEMU CPU emulator [2], which adds to QEMU the ability to perform cycle-level simulations based on CPU timing models. QSim is designed to be used along with simulation

frameworks like SST and Manifold to create a simulation environment capable of executing Intel x86 and x86-64 benchmarks and operating systems.

The QSim implementation is independent of the timing model to the point that it does not require any timing information beyond a periodic timer interrupt for the operating system. The timing model sets execution callbacks within QSim and uses the information they provide about the instruction stream to determine the advancement of execution within QSim and the time within the simulator. QSim can be considered a full-system re-implementation of Shade [3] for the manycore era with the addition of timing feedback.

As a front-end with no concept of time, QSim relies entirely on a back-end simulation to control the advancement of execution and provide meaningful results. In the development of this back-end system it has become clear that the simulation frameworks on which they are developed could benefit from an implementation of processes.

The first complex component being developed as a part of the back end for QSim is a model of a non-blocking cache memory. Combined with the existing simple CPU models and an implementation of a coherence protocol, the first manycore simulations using QSim will be realized. During the initial stages of the implementation of this component, it has become apparent that implementation complexity and code readability would suffer greatly if processes remain unavailable. A read, for example, must be split over at least fourteen functions, or fourteen states within a single function, to handle various delays and corner cases. With processes, this same read can be represented as a single blocking function with ordinary control flow constructs.

3. Related Work.

3.1. Processes and Simulation. Processes have long been a standard feature of discrete event simulators. Frameworks for discrete event simulation frequently feature the process as a fundamental abstraction. Both SimPy and CSIM [13, 12] use an implementation of coroutines to provide a process abstraction to their users. They are limited by the fact that their users are forced to use their one-size-fits-all implementation of processes and cannot work with events directly, which may be appropriate for performance or implementation clarity.

Some frameworks have gone beyond the process abstraction and proposed semantics more appropriate for their field. YetiSim [8] makes its fundamental abstraction the UML execution graph instead of the process. Just like the earlier process-based simulation frameworks, however, YetiSim suffers from the limitation that simulations written with YetiSim cannot easily interact with simulations written with other paradigms.

What we have implemented is a lawyer that allows for hybrid simulations. Events can be used directly where appropriate and our process layers can be used within components to provide the familiar process paradigm in addition to events. Processes can be created and run until they block from event handlers or other processes, caused to advance by the setting of a condition variable from within an event handler or another process, and can communicate through messages sent on links with other components that may themselves use either of the process implementations described here, events, or something altogether different.

3.2. Cooperative Threads using Events. Quite a bit of work has been done on combining the performance of event-driven systems with the programmability of threads in domains where event-driven systems perform better but threads are more programmable, although this work has not yet spread to the realm of discrete event simulation, where processes have traditionally been provided by the simulation kernel.

On the smallest scale, in the embedded domain, are operating systems designed for resource-limited microcontrollers for use in applications like wireless sensor networks. The popular small-scale embedded OS Contiki [4] provides a pure event-driven paradigm to pro-

grammers. Protothreads [5] wraps Contiki with the most basic of stackless threadlike structures. Though they do not maintain per-thread state, which must be managed by the programmer with global variables, Protothreads are useful because they solve the problem of source code organization inherent in event-driven systems.

In the network server domain, the TaskJava [6], Tame [9], and Capriccio [14] programming language extensions provide the same functionality, with the additional feature of managing state without the use of a stack. Just like Protothreads they provide the programmability of threading with the lower memory and time overhead of events.

The approach common to all of these systems is factoring blocking functions into a form of *continuation passing style* (CPS) [6], where a value representing the current state of the function can be passed back to that function as an argument at a later time to continue the execution of that function. When it reaches the next blocking point, the function returns a continuation that can later be used to restart its execution from the point it left off. Combined with appropriate synchronization primitives, continuations can be used as a substitute for cooperative threads, potentially at a much lower cost. In our second process layer implementation, `cps-proc`, we apply this design style in the domain of computer architecture simulation.

4. Semantics. We will refer to the basic operation provided by the simulation kernel as `schedule`. `schedule` takes three arguments, the number of simulation steps in the future for its target function (event handler) to be called, the target function itself, and an optional argument to be passed to the target function. A variant of `schedule` is the fundamental operation in both SST and Manifold's simulation kernels.

To `schedule`, our process layers add four process-oriented operations: `spawn`, `run_ready`, `cond_wait`, and `cond_set`. Both implementations provide only these four operations, which are sufficiently expressive for a wide range of hardware modelling tasks.

The `spawn` primitive creates a new process. Its arguments are the main function of the process and an argument to be passed to it. `spawn` can be called from constructors, event handlers, or within another process. When a task is created it immediately becomes runnable and is added to the end of the ready queue.

The `run_ready` operation can be used from within event handlers and constructors but not other processes. It runs all ready processes until they are finished or blocking. In components, it is usually seen accompanying a `spawn` operation to ensure that the newly created process will be run.

The only synchronization provided by these process implementations is a Boolean condition variable. Processes only block by using the `cond_wait` operation on these variables, which takes as arguments a condition variable and the desired value. Processes that have called `cond_wait` will only resume when the condition variable has achieved the given value, by having `cond_set` called on it. In our C++ implementations, the assignment operator is conveniently overloaded to have the function of `cond_set`. This convention is followed by the pseudocode examples as well. `cond_wait` can only be used from within a process, but condition variables may be set from anywhere, allowing event handlers to trigger condition variables and allow processes to advance.

There is not, as there is in VHDL processes [1] and others, a primitive to wait for a given period of simulation time. However, an implementation is provided, built using the `schedule` and `cond_set` primitives. A pseudocode rendition of this is:

```
cond_var ready;

pause(delay) {
```



```

    ready = false;
    schedule(delay, become_ready);
    wait(ready, true);
}

become_ready() {
    ready = true;
}

```

5. Implementation.

5.1. ult-proc. Our first implementation of a process layer for discrete event simulation frameworks is ult-proc. In its original implementation, ult-proc provided a minimal wrapper for the `getcontext` and `swapcontext` calls. The default stack size is set to two kilobytes, sufficient for processes without many large local variables or deep call nests. An optional third parameter to `spawn` allows the stack size to be adjusted to meet the demands of the application.

The advantage to using user-level threads over other means is simplicity, both of the implementation and the API. While `cps-proc` requires all blocking functions to be wrapped inside of C++ classes, with local variables represented by class member variables, ult-proc can make any function that returns `void` and takes a single pointer argument into a process. The disadvantages are poorer performance than our other implementations and that the stack and process state represent binary regions which are not readily serialized. If local variables could possibly be pointers, then serialization becomes altogether impossible. This makes ult-proc a non-option for large-scale simulations using SST.

5.2. A Better Context Switch. the cost of using ult-proc is in the memory overhead of unused stack space, the extra time needed to allocate it, and the time overhead of the context switch. While the cost of a context switch in `cps-proc` is rather light, in ult-proc all of the callee-saved CPU registers must be saved to some memory region and later restored. When using `swapcontext` the situation becomes worse. Implementations of `swapcontext` are required to preserve the signal mask of each of their threads, which should never be touched by a simulator component. On x86 hardware running Linux, this requires a call to `sigprocmask`, which incurs hundreds of cycles of overhead as observed in Section 6.

5.3. cps-proc. The second implementation uses a technique similar to Protothreads [5], handling all of the complexities of resuming from continuations with the preprocessor. There is still some design overhead; the component writer must remember to adhere to conventions when creating processes. Other projects in the event-driven systems domain have dealt with this problem using source-to-source translation Tame or by adding new language features [6]. Instead, `cps-proc` imposes a set of constraints on the programmer in exchange for working with the existing toolchain. The Poisson process traffic generator example from Section 2.2 reimplemented in the style expected by `cps-proc` would be:

```

class process_main : public cps_proc {
    cps_delay pause;
    cps_proc *main() {
        CPS_PROC_BEGIN();
        pause.period = exp_rn(MTBP);
        CPS_PROC_CALL(pause);
    }
};

```



```

while (running) {
    generate_packet();
    pause.period = exp_rn(MTBP);
    CPS_PROC_CALL(pause);
}
CPS_PROC_END();
}
};

constructor() {
    spawn_process(new process_main());
}

```

The main function of the process must be named `main` and return a pointer to a `cps_proc`. This function must be wrapped in a class that inherits from `cps_proc`. Any variables that would normally be local variables of the `main` function that must persist over blocking calls should instead be declared as member variables. `main` must begin with an instantiation of the macro `CPS_PROC_BEGIN` and end with an instantiation of `CPS_PROC_END`. Blocking calls must be performed on objects of the same type with `CPS_PROC_CALL`. These can be class variables or allocated on the heap. Not seen in this example is the `CPS_PROC_WAIT` macro, which implements the `cond.wait` primitive as described in Section 4.

6. Performance.

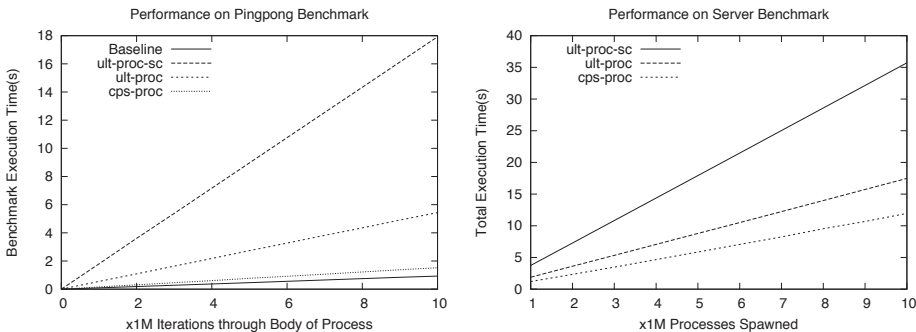


FIG. 6.1. The ping-pong synthetic benchmark can be used to roughly estimate the time required for a `wait` operation that blocks. Normalized averages for the time of a single iteration through the process loop of a single ping-pong component are 1, 3.6, and 11.8 for `cps-proc`, `ult-proc`, and `ult-proc-swapcontext` respectively. The server synthetic benchmark can be used to roughly estimate the time required for a `spawn` operation. Normalized estimates for the time of a single `spawn` in the server benchmark are 15, 19, and 35 for `cps-proc`, `ult-proc`, and `ult-proc-swapcontext` respectively.

6.1. Synthetic Benchmarks. Two synthetic benchmarks were devised to find rough approximations of the amount of time required to block on a condition variable with `cond.wait` and the amount of time required to `spawn` a new process. These benchmarks can be used to provide a first-order approximation of the relative performance of actual simulation components that spend significant portions of their time invoking the process system.

The first benchmark, built for the purpose of estimating the time required for a blocking `wait`, is called ping-pong. It consists of two components each running a single process constantly passing an event back-and-forth until the simulation has ended.

Figure 6.1 displays the execution time with respect to the number of loop iterations allowed to complete. Each iteration contains two `cond.wait` operations that are guaranteed to block. It can be seen that they are straight lines with origins near zero, and in fact their execution time can be predicted using a linear approximation based on the computed iteration time to within an RMS error of 5.6ms. On the hardware used for the experiment, an Intel Xeon workstation clocked at 3GHz, half of the iteration time, a rough estimate of the time required to perform a blocking `cond.wait`, is 76ns for `cps-proc` and 272ns for `ult-proc`. The benefit of a context switch without system calls can be seen by noticing that the iteration period more than triples to 896ns for `ult-proc` using `swapcontext`. Normalizing these values to 1, 3.6, and 11.8 for `cps-proc`, `ult-proc`, and `ult-proc-swapcontext` respectively gives a platform independent measure of the relative time required for a blocking `cond.wait`.

The “baseline” time given in Figure 6.1 is for a version of the ping-pong benchmark implemented without using a process library. Instead of blocking twice per iteration of a loop, the baseline time is simply the time required to schedule a new event when necessary, 0.61 on our normalized scale.

The second synthetic benchmark is designed to provide an estimate of the time required to spawn a process. It is a server that, upon the receipt of a request, spawns a new process to handle it. This process delays for a predetermined period and then schedules an event corresponding to returning a result.

The results of ten runs of this benchmark are also in Figure 6.1. Again, assuming that iteration times add linearly is a valid approximation, within an RMS error of 49ms. On our test system, the time for an iteration of the server benchmark less the half-iteration time for the ping-pong benchmark, an approximation of the time required to spawn a process, has been determined to be 1.12 μ s, 1.46 μ s, and 2.65 μ s for `cps-proc`, `ult-proc`, and `ult-proc-swapcontext` respectively. These values, normalized on the same scale as the normalization of the ping-pong iterations are 15, 19, and 35. Spawning a new process has a higher cost than blocking in an existing process.

6.2. Profiling. The QSim front end provides a statistically sampled profiler that produces a flat profile of dynamic instructions spent in each function in both the kernel and the application. The advantages to this over a standard profiler like `gprof` are that it imposes no overhead for instrumentation and provides an accurate picture of instructions executed within the operating system as well as user mode. The disadvantages are that it offers no call graph and without a timing model can only provide profiles in terms of dynamic instructions, not time. The greatest deficiency in our experiments of the use of dynamic instruction counts is understanding the impact of making frequent system calls. Dynamic instruction count is a poor estimator of execution rate when two frequently executed instructions cause a permission level switch costing hundreds of cycles.

To illustrate how much the cost of making system calls is underestimated by dynamic instruction counts, the RMS error for a fixed-time-per-instruction estimation of the runtime of the various forms of both benchmarks is 0.44s. Omitting `ult-swapcontext` this average error drops down to 0.17s.

In Figure 6.2, we see that most of the dynamic instructions in both benchmarks are spent allocating memory, both for events and process state. The profile also nicely demonstrates the advantage of relying on a simple user-level context switch in `ult-proc` even though the real cost of this context switch is much greater. Not only is there significantly less context switch overhead, but the cost of allocating a process has also decreased due to the smaller size of a process.

7. Conclusions. We have demonstrated a portable, serializable process implementation that works as a layer separate from an underlying event-based simulation kernel, intended

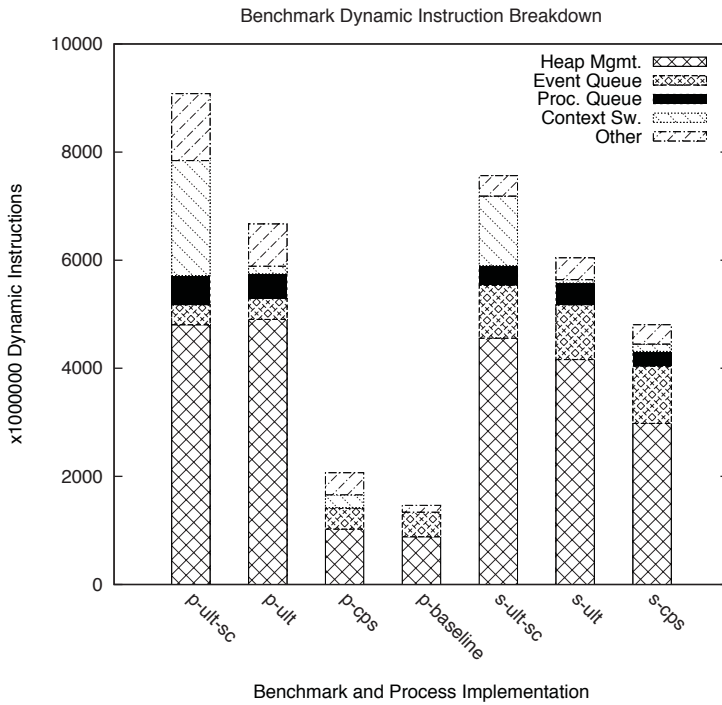


FIG. 6.2. Summary of dynamic instruction breakdown gathered with QSim profiler.

for use in the implementation of hardware component models for parallel simulation frameworks like SST and Manifold. In addition to this we have demonstrated a portable but non-serializable process implementation that fills the same role and provides a more convenient programming environment to the component writer. We have also shown that the system context switch functions are inappropriate for implementations of processes in simulations and replaced them with a more adequate solution. Use of the same context switch library within QSim increased its performance in certain applications threefold.

8. Future Work. It is clear from Figure 6.2 that more work can be done to improve the efficiency of memory allocation in simulations running on SST and Manifold. Even the implementation of the ping-pong benchmark that used no process system at all had comparable memory allocation overhead to the implementation using cps-proc. This is largely due to the fact that every event created must be allocated on the heap by the component. The optimization of allocators could have a significant impact on simulation performance.

REFERENCES

- [1] *IEEE standard VHDL language reference manual*, IEEE Std 1076-1987, (1988).
- [2] F. BELLARD, *Qemu, a fast and portable dynamic translator*, in ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference, Berkeley, CA, USA, 2005, USENIX Association, pp. 41–41.
- [3] B. CMELIK AND D. KEPPEL, *Shade: a fast instruction-set simulator for execution profiling*, in SIGMETRICS '94: Proceedings of the 1994 ACM SIGMETRICS conference on Measurement and modeling of computer systems, New York, NY, USA, 1994, ACM, pp. 128–137.
- [4] A. DUNKELS, B. GRÖNVALL, AND T. VOIGT, *Contiki - a lightweight and flexible operating system for tiny networked sensors*, in Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I), Tampa, Florida, USA, Nov. 2004.

- [5] A. DUNKELS, O. SCHMIDT, T. VOIGT, AND M. ALI, *Protothreads: Simplifying event-driven programming of memory-constrained embedded systems*, in Proceedings of the Fourth ACM Conference on Embedded Networked Sensor Systems (SenSys 2006), Boulder, Colorado, USA, Nov. 2006.
- [6] J. FISCHER, R. MAJUMDAR, AND T. MILLSTEIN, *Tasks: language support for event-driven programming*, in PEPM '07: Proceedings of the 2007 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation, New York, NY, USA, 2007, ACM, pp. 134–143.
- [7] R. M. FUJIMOTO, *Parallel discrete event simulation*, Commun. ACM, 33 (1990), pp. 30–53.
- [8] A. GUILLON AND D. LOACH, *YetiSim: a C++ simulation library with execution graphs instead of coroutines*, in SpringSim '08: Proceedings of the 2008 Spring simulation multiconference, San Diego, CA, USA, 2008, Society for Computer Simulation International, pp. 1–6.
- [9] M. KROHN, E. KOHLER, AND M. F. KAASHOEK, *Events can make sense*, in Proceedings of the 2007 USENIX Annual Technical Conference, June 2007.
- [10] J. B. MAJOR, *Sizing a federated computer system*, in WSC '74: Proceedings of the 7th conference on Winter simulation, Winter Simulation Conference, 1974, pp. 753–754.
- [11] A. RODRIGUES, *The structural simulation toolkit*, 2007. <http://www.cs.sandia.gov/sst>.
- [12] H. SCHWETMAN, *CSIM: a C-based process-oriented simulation language*, in WSC '86: Proceedings of the 18th conference on Winter simulation, New York, NY, USA, 1986, ACM, pp. 387–396.
- [13] S. D. TEAM, *Simpy simulation package homepage*, 2010. <http://simpy.sourceforge.net/>.
- [14] R. VON BEHREN, J. CONDIT, F. ZHOU, G. C. NECULA, AND E. BREWER, *Capriccio: scalable threads for internet services*, in SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles, New York, NY, USA, 2003, ACM, pp. 268–281.

RELIABILITY SIMULATION FOR STRUCTURAL SIMULATION TOOLKIT

AARON S. WILLIAMS* AND ARUN F. RODRIGUES†

Abstract. Large scale reliability simulation is an important topic in high performance computing. It is important to determine how long a system will be available to complete a calculation without a failure. These failures can happen on multiple levels throughout the system and these failure rates are all different so having a simulator where each components failure rate can be modeled independently is important. In this paper we discuss the motivation behind reliability simulation, the Structural Simulation Toolkit on which the reliability simulation is being based, and then give the basics of the various components needed to complete the simulator. We also discuss a possible addition to the project as well as the desired output once it is complete.

1. Introduction. The goal of the ACS Reliability and Resilience project is to estimate the failure rates of system components and do this using only monitoring data. Our approach is to use the Structural Simulator Toolkit (SST) to generate realistic data in order to test the new algorithms being developed. SST is being adapted to provide a simulation of jobs being executed on a set of connected components with specified failure rates. Failure statistics are collected from this simulation to be fed into some algorithms for testing purposes. In section 2 we describe the project tasks and the types of data needed to be generated. Section 3 provides a brief description of SST. In section 4 we describe how the system must be described in XML to properly create the connections of the graph of components being simulated. Section 5 describes the reliability simulation workload/job list. In section 6 we describe the components which were added to SST for resilience simulation. Section 7 describes the output results acquired from SST and section 8 provides a summary of the conclusion of this project.

2. Simulation Task. In order to develop this failure simulation capability in SST we had to do several tasks. First, we had to design and create simulation components. There are several types of components necessary, the leaf or compute nodes, the reliability nodes, the scheduler node, and the infrequent component. The leaf or compute nodes are the nodes on which jobs get assigned and they take a certain scheduled time to complete these jobs. The reliability nodes create the tree structure that connects the leaves. They can be pictured as a rack in which four compute nodes are connected or a cabinet in which multiple racks are connected. The scheduler node takes the set of jobs to be accomplished and maps them across the leaf nodes and also takes care of bookkeeping for successes and failures of jobs. The infrequent component is optional to this task but is a component that is rarely needed by the compute nodes but where failure would still affect the completion of a task. Secondly we are working on a method to distill the records into an output useful by the team modifying the algorithm which will use this test data. Lastly we will need to continue to support development of the simulator as the algorithm team finds bugs in the simulator code.

3. Overview of Structural Simulator Toolkit (SST). We are in the process of developing the second generation of an event driven computer simulation tool called Structural Simulation Toolkit[1]. This reliability simulation capability is being built as an addition to many other capabilities SST contains. At the beginning of a run of the simulator, an XML file is read which defines how the user wants the system connected or “wired up”. These connections are made using links on which events can be sent between components. These events are developer defined. Event driven simulation means that everything that happens in the simulator is triggered by an event on a link from either another component or the same component trying to trigger an action internally. These events are received and handled by

*Arizona State University, aswilli4@asu.edu

†Sandia National Laboratories, afrodri@sandia.gov

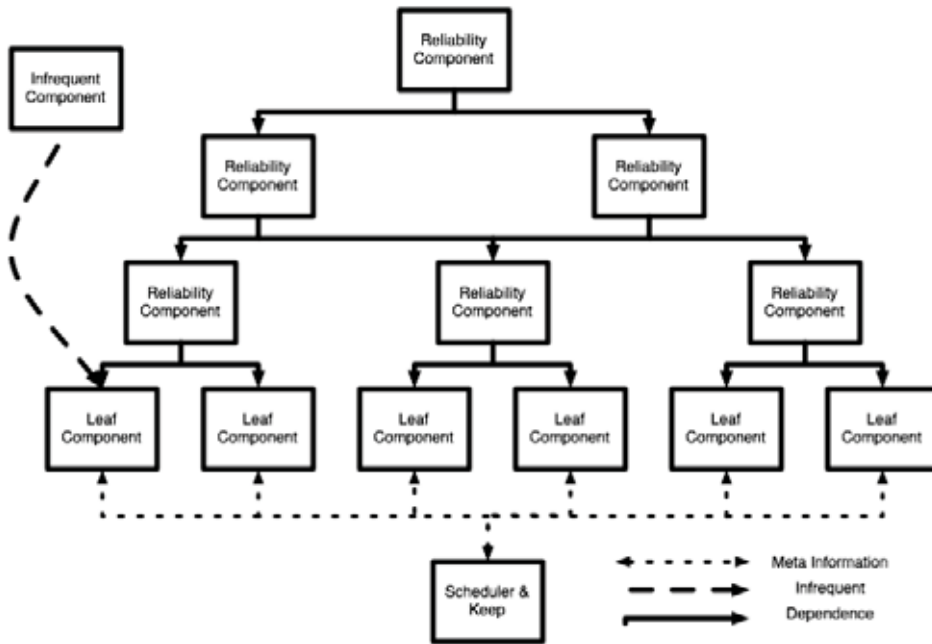


Fig. 4.1. Partial example of XML configuration for simulation

an “event handler” which is defined by the developer. This event system is what we utilize to simulate failures and communication down through the tree that a failure has occurred.

4. Establishing System to Simulate in XML. The requirement for the simulator was a script that configures the system that is being tested. We refer to this phase as “Instantiating the System”. A graphical example of a way to lay out these nodes is provided in figure 4.1. This script is given a number of leaf/compute nodes we intend to implement. There is a specific tree structure required for the algorithms being implemented. This script takes this number of leaf nodes and automatically generates an XML file of nodes including these leaf nodes and the other nodes required to complete the tree. The script generates the links between these nodes as required. In figure 4.2, a portion of a 6 compute node tree is given. The two components you see defined are the root node at the top of the tree and one just below it. We observe that each component is given a unique id and is of type “resil”. This tells SST which component class to use. We then see listed below that the links that are connected to the component, which port to connect them to, and what the title of the edge is called. The title of the edge must match between the two components you would like to connect on that link, so that SST knows how to wire correctly. It can also be noted that a required latency of 1 unit of time, in this case 1 nanosecond, is a parameter of each link because SST does not allow an event to arrive at the exact same time it was sent unless it is a self link to the component. Each component also has a parameter “lambda”. This parameter is used along with an exponential distribution to determine the random time at which this component will fail.

5. Establishing the list of jobs to simulate. A script has also been written which outputs a list of jobs to be accomplished by the given N leaf nodes. The job list has a list of jobs each of which is assigned a “Job ID”, a “size” or number of leaves necessary to compute, and

```

<component id="1.1">
  <resil>
    <params><lambda>0.255809</lambda>
      <port0>1.1-2.1</port0> <port1>1.1-3.1</port1></params>
    <links>
      <link id="1.1-2.1">
        <params><lat>1NS</lat><name>link0</name></params>
      </link>
      <link id="1.1-3.1">
        <params><lat>1NS</lat><name>link1</name></params>
      </link>
    </links>
  </resil>
</component>
<component id="1.2">
  <resil>
    <params><lambda>0.255809</lambda><port0>1.2-2.1</port0>
      <port1>1.2-3.1</port1>
    </params>
    <links>
      <link id="1.2-2.1">
        <params> <lat>1NS</lat> <name>link0</name>
      </params>
      </link>
      <link id="1.2-3.1">
        <params> <lat>1NS</lat> <name>link1</name>
      </params>
      </link>
    </links>
  </resil>
</component>

```

FIG. 4.2. Partial example of XML configuration for simulation

a “duration” or number of cycles to complete the task. An optional parameter on this list that may be used later is a list of required components that this job must have to complete. This list would be used by the scheduler when it is implemented.

6. Implemented Components. As discussed in the introduction, there are several types of components required to simulate the failure modes required.

6.1. Leaf/Reliability Components. In figure 4.1 we see that there are leaf nodes described as well as reliability nodes. When first conceptualizing this project we thought that these would have to be two different types of components in SST. We soon realized that their behaviors are very similar and can be combined into the same component type. The incoming link titles help these components determine if they are actually leaves at the bottom which are being assigned jobs or if they are above that point in the graph. Leaf/Compute component pseudo code can be found in figure 6.1

Here we can see that the component is polling to see if it has failed while running a job in the pseudo code. In the event driven simulation however, the failure is actually sent as an

```

if(runningAJob()){
    failed = checkForFailure()
    if(failed){
        t = computeDowntime();
        tellScheduler(failed,t);
    } else if (jobDone()) {
        tellScheduler(completed);
    } else if( needInfrequentComponent()) {
        requestFromInfrequentComponent()
    }
}
while(e = GetNextEvent()) {
    switch(e) {
        case jobLaunch:
            startJob();
            informParentComponents();
        case failureNotice: //from another component
            killJob()
            tellScheduler(failed);
        case jobKill: //from scheduler
            killJob()
        case infrequentReply: //an infrequent node gets back to us
            if(e.reply == BROKEN) {
                killJob();
                tellScheduler(failed);
            }
    }
}

```

FIG. 6.1. *Leaf/Compute node Pseudo code*

event to the same component in the future, and if that event arrives before the job is completed, the job fails. Once a job fails we assume instant repair time for these components. This will be modified in the future to be a variable length downtime. Once a job has completed, the component tells the scheduler that it is free again and can be utilized. Also the scheduler can tell a component to stop mid job if the job a component is computing has failed because of a failure elsewhere in the tree that didn't directly affect the component itself. The same component type is used for the reliability components in figure 4.1. Reliability component pseudo code can be found in figure 6.2.

Here we can see that the functionality is very similar to that of the leaf/compute nodes with the only difference being communicating with the scheduler to indicate that the component is free. Also this component has an additional portion of code where it sends events to the children below it in the tree saying that it has failed in the event of a failure arriving on a self link. A failure on one of these nodes will cause a stop on all of the nodes directly connected below this component all the way down to the set of leaf nodes connected on its branch. This would cause the leaf node to notify the scheduler that a failure occurred and thus its part of the job failed.

The failure distributions are modeled as a exponential distribution with each layer of the tree having a different "lambda" value given. The equations associated with the exponential


```

if (state == BROKEN) {
    r = checkIfRepaired();
    if (r) {
        state = REPAIRED;
    }
} else if (state == RUNNING) {
    failed = checkForFailure()
    if (failed) {
        t = computeDowntime();
        tellChildren(failed, t);
        state = BROKEN;
    } else {
        if (time() > runTill) {
            state = REPAIRED;
        }
    }
}
while(e = GetNextEvent()) {
    switch (e) {
        case jobLaunch:
            runTill = max(runTill, e.runTill);
            state = RUNNING;
        case failureNotice:
            tellChildren(failed, t);
            state = BROKEN;
    }
}

```

FIG. 6.2. *Reliability node Pseudo code*

distribution are widely known so we have excluded them. This lambda value determines when the self link is sent in the future to trigger a fail on one of the reliability or leaf components.

6.2. Scheduler Component. The scheduler component is the utilized to assign the job list input onto the compute nodes. It is also the component that keeps track of failures and writes the output to a file to be used later on. This first implementation of the scheduler is very basic. It has a list of compute nodes of the system and which ones of those are currently available. It implements a first come first served scheduling scheme. This has no look ahead into the jobs list or concept of when jobs will complete on the compute nodes. If a job requires 3 nodes to complete, the scheduler waits for three nodes to be free then schedules this job onto these nodes. If a failure causes the failure of a job, this job is moved to the bottom of the queue and is completed at a later time. There is a future scheduler planned which will implement more sophisticated scheduling tactics to optimize the throughput of the system to minimize the time taken to finish all jobs on the list. Unlike other components, the scheduler is assumed to never fail. The Scheduler and Book keeping Component pseudo code can be found in figure 6.3.

6.3. Infrequent Component. A future component that may be implemented will be the concept of an “infrequent component”. This will be a component which would be required occasionally for some jobs. It would be modeled after components such as a filesystem on a high performance computer. It would simply have an available or unavailable state variable

```

while(!done){
    if(nodeTable.hasRoom()){
        ScheduleNextJob();
    }
    while(e = GetNextEvent()){
        switch(e) {
            case nodeFailure:
                killJob(e.job)    //kill the job on all nodes it is running
                recordStats(e);    //record
                nodeTable.freeNodes(e.job); //record the nodes as free
            case jobCompletion:
                recordStats(e);        //record
                nodeTable.freeNodes(e.job); //record the nodes as free
        }
    }
}

```

FIG. 6.3. Scheduler node Pseudo code

and some repair time distribution associated with it and jobs would have to wait for it to be available to complete if they required this type of component. We are not certain this type of component will be required for simulations.

7. Simulation Outputs. There are two different outputs required from the simulator. The first is a “job trace”. This is a file that lists every job. For each job the Job ID is given followed by the result, pass or fail, the start time, the end time and which compute nodes were used. The second output is the “failure trace”. In this output you get a list of every component in the system, both compute and reliability nodes, and the times they failed. These two files will allow failure statistics to be compiled and used in the testing of the modified algorithms for the ACS Reliability and Resilience project.

8. Conclusions. We have provided some background for reliability simulation as well as the basic components necessary as well as the inputs and outputs desired. We have also described some of the basics of event based computer system simulation as well as information on SST. We described how SST needs a system to be defined in XML in order to wire up components and transmit events between them on links. The components described above complete a reliability simulation system. They will help with the verification and testing of some modified algorithms with the ACS Reliability and resilience project here in the Computer Science Research Institute.

REFERENCES

- [1] A. RODRIGUES, *The structural simulation toolkit*, 2007.

Visualization and Software Engineering

The methods and architectures in the previous section have the potential to generate massive amounts of data with increasingly complex software infrastructures. The articles in this section are dedicated to development and understanding of visualization and advanced software engineering techniques to tackle these challenging issues at the core of large-scale computing and computational science.

Silva and Shepherd describe the development of the Titan informatics toolkit for the VisTrails workflow management and information visualization system. Using Titan's informatics processing algorithms this coupling enables, for example, simplified workflow based multiple-view comparative visualization. *Harger and Crossno* compare a number of open source visual analytics toolkits. Using a feature based comparison several applications were chosen for a more in depth study using a benchmark document 'explorer' application. *Nusbaum and Heroux* presents a new graphical user interface framework for parameterized scientific applications. This software package replaces traditionally used input decks, providing a more intuitive way for a user to specify application parameters. *Fermoyle and Heroux* discuss the usage and functionality of the newly developed Tpetra/Stratimikos linear algebra capability in Trilinos. The authors presentation provides a step-by-step comparison of the new technology with the Epetra/AztecOO software stack, emphasizing differences and the enhanced capabilities offered by Tpetra. *Hardy and Clark* overview the development of testing capability for the Python interface in Cubit. The authors also discuss current and future metrics that can be used in testing.

E.C. Cyr
S.S. Collis

December 17, 2010

INFORMATION VISUALIZATION USING VISTRAILS TECHNOLOGY

WENDEL B. SILVA* AND JASON F. SHEPHERD†

Abstract. The Titan project is an informatics toolkit that provides a flexible, component-based pipeline architecture for ingestion, processing, and display of informatics data. It integrates capabilities from a series of best-in-class open-source toolkits for scientific visualization, graph algorithms, linear algebra, and more. VisTrails is an open-source scientific workflow and provenance management system that provides support for data exploration and visualization. Moreover, a visualization pipeline created using VisTrails will be able to interact with other software tools, including VisMashup and crowdLabs. Both software tools, Titan and VisTrails, represent the state of the art in their research areas. In this paper we present the effort to develop a package allowing the Titan toolkit to interact with VisTrails. Using the package developed, two projects within the Titan toolkit have been ported to VisTrails, VisMashup and crowdLabs.

1. Introduction. Nowadays, there are a number of established toolkits for scientific visualization and data analysis (e.g., VisTrails, Opendx, Paraview, SCIRun). Many of these toolkits use workflow based interfaces that expose computational components as modules, and represent its pipeline as sequential and intuitive steps. In addition, workflows allow the creation of complex visualization pipelines that combine these modules, using the modules to represent computational components with connections, to express the flow of data through the pipeline. The use of workflow based interface is useful for comparative visualization and efficient exploration of parameter spaces. Although systems are being developed with a more sophisticated visual programming interfaces, the path from the raw data to insightful visualizations is laborious and error-prone.

Another important feature in scientific data analysis and exploration is provenance. Provenance helps users interpret and understand results. By examining the sequence of steps that led to a result, we can gain insights into the chain of reasoning used in it's production, verify that the experiment was performed according to acceptable procedures, identify the experiment's inputs and reproduce the result.

This paper presents and describes the creation of the Titan package for VisTrails, allowing the construction and execution of informatics pipelines in a workflow based interface visualization system with provenance. In addition the algorithms available in Titan, Titan coupled with VisTrails also provides a user interface that simplifies multiple-view comparative visualization, efficient execution of complex visualization pipelines, an infrastructure for the provenance, an interface to VTK algorithms[12] in the pipeline, support for python scripting, and a framework to easily import python-based libraries. Using the Titan package in VisTrails, we ported two document analysis projects associated with the Titan project (namely, ParaText[9] and Prototype 2 'P2'). Utilizing these new VisTrails pipelines enabled their execution with VisMashup[15] and crowdLabs[1].

In section 2 we present the Titan toolkit as well as the two projects ported in this paper. In section 3 we describe the creation of the Titan package for VisTrails, and we detail the process of building the ParaText and Prototype 2 pipelines using the Titan package (section 4). Section 5 describes the VisMashup and crowdLabs software and show the projects running on it. Finally, a conclusion is presented and an outline of directions for future work.

2. Titan Toolkit. The Titan Informatics Toolkit[7] is a collaborative effort led by Sandia National Laboratories. It represents a significant expansion of the Visualization ToolKit (VTK) to support the ingestion, processing, and display of informatics data. The Titan project

*School of Computing, University of Utah, wendelbsilva@sci.utah.edu

†Sandia National Laboratories, jfshep@sandia.gov

represents one of the first software development efforts to address the merging of scientific visualization and information visualization on a substantive level.

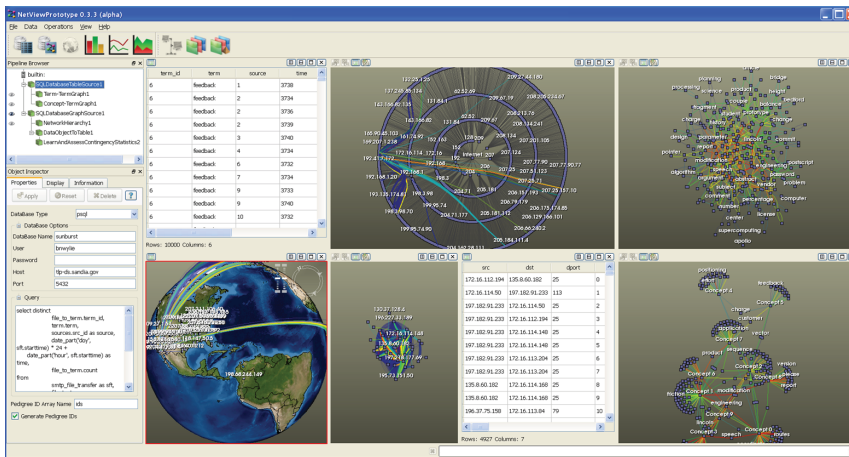


FIG. 2.1. A document analysis application using Titan Informatics Toolkit is shown.

By leveraging the VTK engine, Titan provides a flexible, component based, pipeline architecture for the integration and deployment of algorithms in the fields of intelligence, semantic graph and information analysis. The VTK parallel client-server layer will provide an excellent framework for doing scalable analysis on distributed memory platforms. The benefits of combining the two fields are already reaping rewards in the form of functionality.

Titan integrates capabilities with a series of best-in-class open-source toolkits, including scientific visualization (VTK[12]), graph algorithms (Boost Graph Library [11]), linear algebra (Trilinos[3]), named entity recognition (StanfordNER[10]), and others. Components may be used by application developers using their native C++ API on all popular platforms, or by using any of a broad set of language bindings that include Python, Java, and TCL. As shown in Figures 2.1 and 2.2, applications are built by combining Titan components to address problems in a specific domain.

To simplify the use of the toolkit, Titan has a set of readers, database connectors, components, views and interfaces that allow end-user applications to be constructed from the toolkit components. The readers allow the user to easily import data and transfer it into one of the Titan data structures. With the data in one of the data structures available, the user can then use one of many Titan algorithms for the processing, transforming, modifying or visualizing the data.

In the same way that scientific visualization applications can be built with VTK, Titan can be used to build information visualization and analysis applications. In the subsections we describe two projects currently present in the Titan toolkit: Prototype 2, ParaText.

2.1. Prototype 2. Prototype 2 (P2) is a document analysis tool built from Titan algorithms allowing similar-document clustering, entity extraction, and document-entity relationships visualization coupled with powerful graph search algorithms in a stand-alone application. It organizes and classifies documents based on their semantic content and performs named-entity (people, location, organization, misc) extraction to allow the exploration of the relationships between entities and documents. P2 was developed to include document clustering using latent semantic analysis (LSA) and named-entity extraction to allow enhanced searching based on content (e.g., particular locations or people) within a corpus of docu-

ments. P2 facilitates the investigation of free text data, identifying the relationship between the selected documents, and its documents similarities and entities. In its current version, P2 supports the following file formats: MS Word, PDF and plain text.



FIG. 2.2. Main window of the project Prototype 2. The left figure show the application after analyzing a set of documents. At right, it show the same screen after some user interactions.

In Figure 2.2 is the main window of the application after the documents being analyzed have been processed. To be more precise, the view located in the upper-left of Figure 2.2 (cluster list view) shows the group of documents analyzed into conceptual clusters. The documents with the same topic are grouped in the same cluster. The cluster label represents the *meaningful* terms that occur in the documents within the cluster. Documents are also clustered into groups with similar content to each other.

A similarity graph in a tree-ring view is shown on the bottom-left corner of the application window (Figure 2.2). It visually reflects the cluster shown on cluster list view and shows the similarity links between the documents. High similarity links are represented in red, whereas links between clusters are blue and green.

The middle of the window display in Figure 2.2 A shows the extracted text from the various document formats accepted by the application. The text is displayed with the extracted entities highlighted and hyperlinked. The words highlighted are color coded, based on entity type (person, organization, location, misc). The hyperlink attached to the word will open the wikipedia webpage containing information about that term (Figure 2.2 B).

Users are able to select specific entities of interest by dragging them into the hotlist, which appears just below the list of named entities (Located in the middle-right of the window Figure 2.2). The application then computes the paths between entities using a connection subgraph algorithm. The algorithm attempts to find short, direct paths between nodes in a graph by penalizing long paths as well as paths which pass through high-degree nodes. The resulting graph can be seen in the lower-right part of Figure 2.2 B, bottom-right graph.

2.2. ParaText. The ParaText system is an end-to-end process for scalable distributed memory analysis of large document collections. It is composed of a set of text analysis components designed to function within a Titan data processing pipeline, where data sources, filters, and sinks can be combined in arbitrary ways. ParaText presents a full LSA process from text ingestion and modeling data to analysis tasks including information retrieval and document similarity.

ParaText components are chained together into data-parallel pipelines that are replicated across processes on distributed-memory architectures. Individual components can be replaced or rewired to explore different computational strategies and implement new functionality. The retrieval method employed is latent semantic analysis (LSA)[14] using singular

value decomposition (SVD)[8].

ParaText has been used in applications spanning a broad set of domains. Figure 2.3 shows the visualization of some data generated using ParaText.

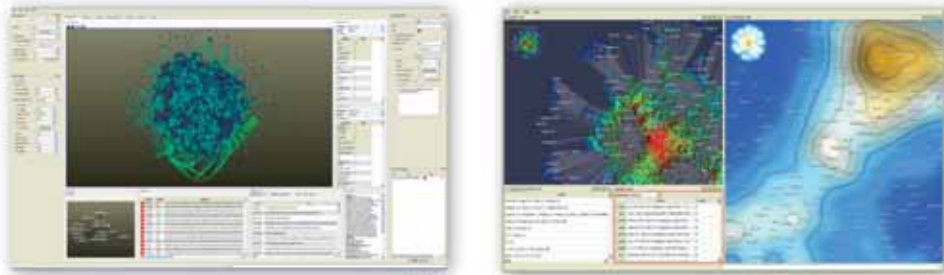


Fig. 2.3. Visualization of data generated using ParaText and visualized using ParaView. At the left shows the result of a bibliometric analysis and in the right, funding portfolio analysis.

The first part of the pipeline consists of filters for extracting and transforming text. With the exception of determining which files should be processed on which processors, the filters described in this section all parallelize extremely well[9]. Once each processor determines the list of terms in its local data (i.e., documents), the *Term Dictionary* filter creates a global dictionary with each term listed exactly once. Given the list of local and global terms dictionary computed, each processor uses the *Term Document Matrix* filter to create its portion of a sparse, distributed term-document frequency matrix. Following creation of the frequency matrix, the matrix must be weighted to incorporate the importance of the terms throughout the collection.

Once the local and global term weights are computed, the *Scale Dimension* filter then applies these weights to the matrix. To compute the SVD of the weighted term-document matrix, ParaText wraps the distributed block Krylov-Schur method from the Anasazi package of the Trilinos solver library. In ParaText, document similarities are computed as the cosine values between scaled LSA document vectors.

3. Titan Package for VisTrails. VisTrails[6] is an open-source system designed to support exploratory computational tasks, including visualization and data mining. VisTrails combines features of workflow and visualization systems, and provides a comprehensive provenance management infrastructure. The availability of provenance information enables a series of operations that simplify exploratory processes and foster reflective reasoning. For example, scientists can easily navigate through the space of workflows created for a given exploration task, visually compare workflows and their results, and explore large parameter spaces.

VisTrails provides a plugin infrastructure to integrate user-defined functions and libraries. Specifically, users can incorporate their own visualization and simulation codes into pipelines by defining custom modules (or wrappers). These modules are bundled into *packages*. A VisTrails package is simply a collection of Python classes where each of these classes represents a new module.

VisTrails presents an easy way to create an user-defined module as can be seen in Algorithm 1. New VisTrails modules must subclass from *Module*, the base class that defines basic functionality. The only required override is the *compute()* method, which performs the actual module computation. Input and output is specified through ports, which currently have to be explicitly registered with VisTrails.


```

1 class Div(Module):
2     def compute(self):
3         arg1 = self.getInputFromPort("arg1")
4         arg2 = self.getInputFromPort("arg2")
5         if arg2 == 0.0:
6             raise ModuleError(self, "Division by zero")
7         self.setResult("result", arg1 / arg2)
8
9 registry.addModule(Div)
10 registry.addInputPort(Div, "arg1", (basic.Float, 'x'))
11 registry.addInputPort(Div, "arg2", (basic.Float, 'y'))
12 registry.addOutputPort(Div, "result", (basic.Float, 'x/y'))

```

ALGORITHM 1. Example of a very simple user-defined module

Although the Titan toolkit provides python bindings, the Titan wrapping doesn't adopt the class format expected in VisTrails. To use the Titan classes in VisTrails, a python script was developed (i.e., the Titan package) that accesses and registers Titan classes as VisTrails modules. This package loads all Titan classes and translates them in a format recognizable to VisTrails. The script searches all python-wrapped Titan classes and generates a corresponding module.

For optimization purposes, the python version of Titan doesn't define or allocate its own submodules. Because of this, all titan submodules had to be specified within the VisTrails package to load to make them accessible within VisTrails.

During package loading, the script executes the following steps. First, all Titan modules are loaded and module properties are extracted from each submodule. This process is executed recursively ignoring all unnecessary information, generating a tree with all classes present in the python wrap.

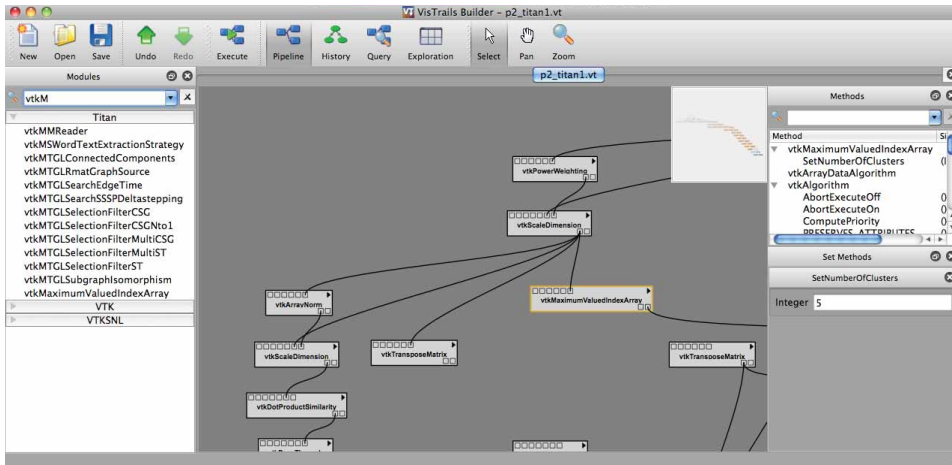


FIG. 3.1. Titan and VTK package working together on VisTrails.

Following the package load, the Titan class tree is traversed and all modules are registered into the VisTrails module registry. For each module registered, the script identifies the get/set methods of the class and adds them as module's input and output. During this process, for each input/output the script also identifies the input/output class type. At this step, the package checks if the class is also present in VTK. If so, the package change the input/out-

put type to use the VTK version of the class. This step is necessary to allow Titan/VTK to connect and work together.

In addition to the *get* and *set* methods of the class, others methods may also be included in the module as input/output, and some methods may be designated as disallowed methods. The package also keeps a list of disallowed classes and modules. Since VisTrails doesn't allow input/output overload, the package searches for overloaded methods and changes the names appropriately.

After the package is loaded, both VTK and Titan packages can work together in the same pipeline (Figure 3.1). As you may observe in the middle part of the Figure 3.1, the package identifies common modules and support their connection and interaction.

Although the Titan package for VisTrails was constructed to allow connectivity between Titan and VTK, the approach utilized doesn't support polymorphism. The Titan package does not check the module's superclass, requiring the user to do the *downcasting* when using the superclass of a module within a *PythonSource* module (Figure 3.2). As shown in Figure 3.2, a situation of polymorphism is found during the port of the P2 to VisTrails. In this situation, Titan module output is an *vtkPDFTextExtractionStrategy* and the VTK module only accepts *vtkTextExtractionStrategy*.

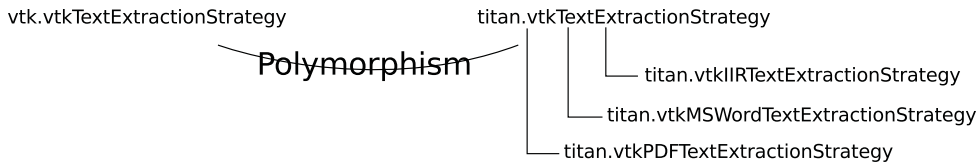


FIG. 3.2. This image show the polymorphism problem between the Titan and VTK packages.

Finally, the Titan package for VisTrails has a dependence on the Spreadsheet and VTK packages. Furthermore, since VisTrails only supports modules implemented in Python, during the Titan compilation it is necessary to enable the creation of the Python wrappers.

4. Porting to VisTrails. A novice user (i.e., an experienced programmer that is unfamiliar with the modules and the dataflow paradigm), or even an advanced user performing a new task, often resorts to manually searching for existing pipelines to use as examples. These examples are then adapted and iteratively refined until a solution is found. Unfortunately, this manual, time-consuming process is the current standard for creating visualizations rather than the exception [13]. Although we had access to the P2 pipeline[2] as well as ParaText and P2 source code to use as reference, the porting was not straightforward. In the next subsection we will give insight on the porting process as well as its results.

4.1. P2 running on VisTrails. P2 was originally developed as a stand-alone application with a Qt interface. As such, P2 uses several different Qt widgets in the GUI front-end (e.g. *QListView*, *QTreeView*, *QTextView*). Since VisTrails does not have modules to wrap QT widgets to display some of the P2 results, the P2 pipeline was adapted to use the modules and features available. VisTrails provides different ways to visualize data (e.g. *RichTextCell* to display HTML and plain text, *VTKCell*, *MplFigureCell*, and *OpenGLCell*[5]). Furthermore, It is possible to build your own view cell with the restriction that it must inherit Qt widget.

Although VisTrails provides enough support to allow the user to build their own visualization module, the P2 pipeline running on VisTrails uses *PyQt* inside the *PythonSource*. For visualization the ported pipeline also uses the modules *RichTextCell* and *VTKCell*. For example, the pipeline uses *RichTextCell* to visualize the documents analyzed and *VTKCell* for the ring graph. Since *RichTextCell* allow the visualization of html files, the pipeline is able

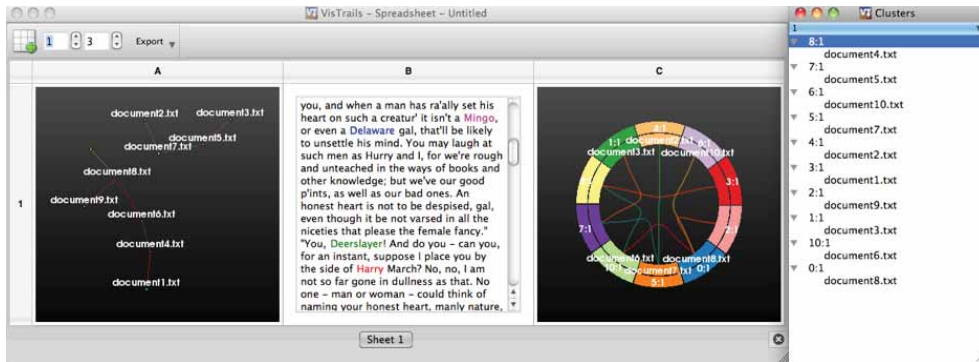


FIG. 4.1. Prototype 2 running on VisTrails using Titan.

```

1 from PyQt4 import QtGui
2 self.TreeView = QtGui.QTreeView()
3
4 #Set the QTreeView parameters
5 #Fill the QTreeView with the information
6
7 self.TreeView.setModel( QtGui.QStandardItemModel() )
8 self.TreeView.show()

```

ALGORITHM 2. Example of a python script used in the module *PythonSource* to use Qt Widgets

to display some text documents in it, as well as highlight and hyperlink the entities inside the document. For these views, during the pipeline a script accesses the document analyzed and it's entities to generate the *html* (Spreadsheet middle cell seen in the figure 4.1).

The figure 4.1 also shows the use of *VTKCell* and *PyQt*. As shown in the middle cell of the spreadsheet, the text has been highlighted with 4 different colors: green, red, magenta and blue, representing the entities person, organization, location, misc, respectively. The right cell, discussed in section 2.1, shows the tree-ring view. The left cell shows the same information as the tree-ring but in a different graph format.

To visualize a tree, Qt widgets are used. Using the VisTrails module *PythonSource* it is possible to write and execute a python script in the pipeline. Moreover, this script has access to other python modules allowing the use of the package *PyQt*. Since *PythonSource* is a base VisTrails module, it can communicate with any other VisTrails modules, even third party packages.

In algorithm 2, an example of the *PythonSource* that creates a *QTreeView* is demonstrated. To use *PyQt* in the *PythonSource* it is not necessary to create Qt instances such as *QApplication* and *QMainWindow*. One important part of the script is to be sure to use *self* to define the widget instance in order to increase it's scope. Having the application scope, the widget will be active until the user closes it or the VisTrails application is closed. If the variable is only on the module scope, the widget will disappear as soon as the module finish its execution.

4.2. ParaText running on VisTrails. Although ParaText was originally designed for execution in a distributed processing environment, the ParaText port to VisTrails is currently executed only in a single processor environment. The files may be sent to the pipeline in two ways. Searching for files in a specified directory, or getting selected files from the file system or the internet. In the second case, if files are specified from the internet the pipeline will

download and use the files specified. Like the P2 port for VisTrails, ParaText also uses the combination of the *PythonSource* with *RichTextCell* to display results.

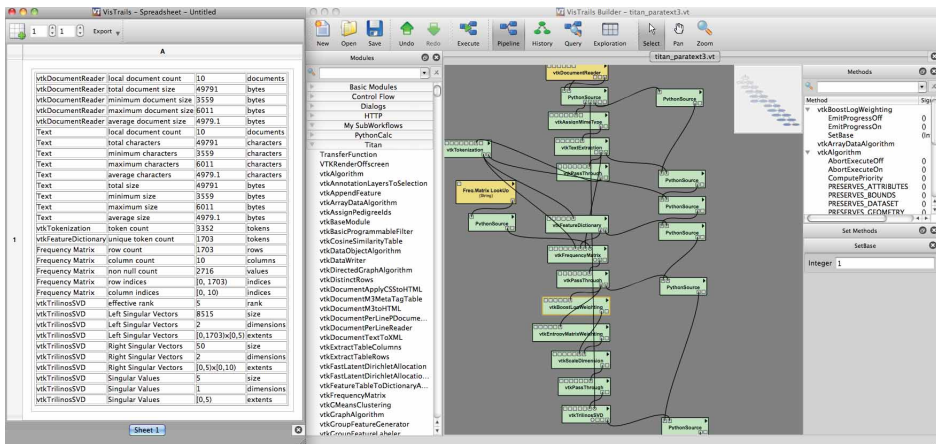


FIG. 4.2. ParaText being executed as a VisTrails workflow.

As shown in figure 4.2, the ParaText pipeline generates an html table and displays this in the spreadsheet. The following information is displayed as result of the pipeline execution: dictionary token count, unique token count, frequency matrix, SVD best rank, and left and right singular and singular values.

4.3. Other comments. *PythonSource* was one of the most used modules during the creation of the P2 pipeline. Using this module the user has the freedom to add almost any python script to the pipeline. This module was used heavily during the creation of the pipeline for debugging purposes. In addition, it was used to do *downcast*, generate *PyQt* visualizations, and generation of the html, among other uses.

Another important use of *PythonSource* was to update a module. When the same module is being used in different parts of the pipeline, for memory maintenance reasons, python may remove the module instance from memory. In this case, the module has to be 'updated' before each usage.

Some of the Titan modules accept multiple connections to an input port. For compatibility reasons VisTrails allows it but the method is hidden. During the construction of the pipelines, some modules used this feature.

5. VisTrails Technology. There are some applications that accept a VisTrails file as input (e.g. VisMashup and crowdLabs). To take advantage of these tools, the next step after porting the projects to VisTrails was to see if we could improve the projects using those applications. In the next subsections we will be present these applications.

5.1. VisMashup. VisMashup is a tool to simplify the creation, maintenance, and use of customized visualization applications. A VisMashup can be combined with any workflow system as long as the workflow system provides access to workflow specifications, the ability to identify and change workflow components, and to execute the workflows. The VisMashup application is flexible, with a GUI automatically generated from a medley specification.

Instead of interacting directly with a dataflow network or a very general and complex GUI, users manipulate and execute a set of pipelines in a medley through a small number of graphical widgets. Application maintenance is simplified, since when the underlying medley

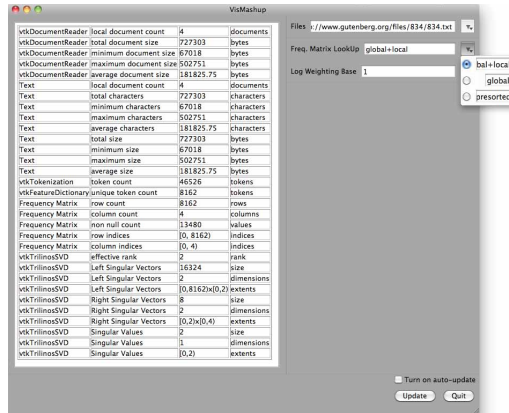


Fig. 5.1. ParaText running on VisMashup.

changes, the GUI can be automatically updated. Furthermore, besides setting parameter values and synchronizing parameters, users can customize a mashup: they can hide, show, and move widgets around. The user may do changes to the pipeline views through the mashup, it is propagated down to the pipeline level and sent to the visualization system for execution.

The port of a VisTrails pipeline to a VisMashup is straightforward[4]. Figure 5.1 shows ParaText running on VisMashup. With the mashup the user does not need to know the pipeline, or need to change the parameters on specific modules within the pipeline. As shown in Figure 5.1, the user can interact with the pipeline parameters using only mashup widgets and immediately see the results.

5.2. crowdLabs. CrowdLabs is a social visualization repository for scientific workflow management systems. It adopts the model used by social Web sites and integrates a set of usable tools and a scalable infrastructure to provide an environment for scientists to collaboratively analyze and visualize data. This kind of website may lower the barriers for the use of scientific analyses and enables broader audiences to contribute insights to the scientific exploration process, without the high costs incurred by traditional portals. In addition, it supports a more dynamic environment where new exploratory analyses can be added on-the-fly. CrowdLabs enables sharing of both VisTrails pipelines and VisMashups.

CrowdLabs provides the most advanced integration of provenance, workflows, visualizations, and social collaboration features to date. The system goes beyond Web access, and provides a powerful API that can be used for developing collaborative analysis applications. CrowdLabs enables the creation of a platform that simplifies the process of integrating publications with computations, visualizations, workflows, datasets and parameter values used to create them.

CrowdLabs is open to the community allowing anyone to create an account and have full access to the website features. The upload of a VisTrails file is done using the upload section present in the VisTrails tab. Anyone with the right access may access the file uploaded and its information that includes version history, workflows, modules, etc. In addition, its embed version of the workflow can be shared in others format such as wiki, blog and L^AT_EX. To create and upload a VisMashup on crowdLabs, the VisTrails files first have to be uploaded in the website. After that, using the VisMashup application you may select the VisTrails that contain the workflow you want to use to create a mashup. It is important to stress that the workflow needs a tag (a name) to make the mashup work. Finally, when the *mashup* is created, there's the option to automatically upload it to crowdLabs. Like the workflow it is

possible to embed the mashup in a webpage.

6. Conclusions and Future Work. In this paper we presented and described the development of the Titan package for VisTrails. In addition, two Titan projects were ported to VisTrails, as well as to VisMashup and crowdLabs. As result of the porting to VisTrails, both projects gained features including workflow based interface visualization and provenance. Although some modifications to the pipeline were necessary, the original and the ported pipelines have the same core functionality. For both projects a VisMashup was created that allowed parameter changes and updates on the pipeline in a simplified interface. Finally, both VisTrails and VisMashup were attached to the web visualization repository crowdLabs.

In the future, ParaText could be implemented on VisTrails to use MPI to enable more efficient runs on larger datasets. Furthermore, both projects could be visualized on mobile devices (e.g. ipod, iphone, ipad) using VisMashup.

7. Acknowledgements. The authors would like to thank Emanuele Santos of University of Utah for her help with VisTrails technology.

REFERENCES

- [1] *crowdLabs*. <http://www.crowdlabs.org/>, Last time accessed: July 2010. Vistrails Inc.
- [2] *P2 Pipeline*. https://www.kitware.com/InfovisWiki/index.php/Prototype.2_Actual_Pipeline, Last time accessed: July 2010. Kitware Inc. and Sandia National Labs.
- [3] *The Trilinos Project*. <http://trilinos.sandia.gov/>, Last time accessed: July 2010. Sandia National Labs.
- [4] *VisMashup Tutorial*. <http://www.sci.utah.edu/emanuele/movies/vismash.mov>, Last time accessed: July 2010.
- [5] *VisTrailsWiki - FAQ*. <http://www.vistrails.org/index.php/FAQ>, Last time accessed: August 2010. Vistrails Inc.
- [6] L. BAVOIL, S. P. CALLAHAN, P. J. CROSSNO, J. FREIRE, AND H. T. VO, *VisTrails: Enabling interactive multiple-view visualizations*, in In IEEE Visualization 2005, 2005, pp. 135–142.
- [7] P. CROSSNO, B. WYLIE, A. WILSON, J. GREENFIELD, E. STANTON, T. SHEAD, L. ICE, K. MORELAND, J. BAUMES, AND B. GEVECI, *Intelligence analysis using titan*, in VAST '07: Proceedings of the 2007 IEEE Symposium on Visual Analytics Science and Technology, Washington, DC, USA, 2007, IEEE Computer Society, pp. 241–242.
- [8] J. W. DEMMEL, *Applied Numerical Linear Algebra*, SIAM, 1997.
- [9] D. M. DUNLAVY, T. M. SHEAD, AND E. T. STANTON, *ParaText: Scalable Text Modeling and Analysis*, in Proceedings of the 19th International ACM Symposium on High Performance Distributed Computing, 2010.
- [10] J. R. FINKEL, T. GRENAGER, AND C. MANNING, *Incorporating non-local information into information extraction systems by Gibbs sampling*, in ACL '05: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics, Morristown, NJ, USA, 2005, Association for Computational Linguistics, pp. 363–370.
- [11] A. L. JEREMY G. SIEK, LIE-QUAN LEE, *The Boost Graph Library: User Guide and Reference Manual*, Addison-Wesley, 2001.
- [12] I. KITWARE, *VTk User's Guide*, Kitware, Inc., 2010.
- [13] D. KOOP, *VisComplete: Automating Suggestions for Visualization Pipelines*, IEEE Transactions on Visualization and Computer Graphics, 14 (2008), pp. 1691–1698.
- [14] T. K. LANDAUER, *Handbook of Latent Semantic Analysis*, Psychology Press and Routledge, 2007.
- [15] E. SANTOS, L. LINS, J. AHRENS, J. FREIRE, AND C. SILVA, *VisMashup: Streamlining the Creation of Custom Visualization Applications*, IEEE Transactions on Visualization and Computer Graphics, 15 (2009), pp. 1539–1546.

COMPARISON OF OPEN SOURCE VISUAL ANALYTICS TOOLKITS

JOHN R. HARGER* AND PATRICIA J. CROSSNO†

Abstract. We present the results of a two-stage evaluation of open source visual analytics packages. The first stage is a broad feature comparison over a range of open source toolkits. Although we had originally intended to restrict ourselves to comparing visual analytics toolkits, we quickly found that very few were available. So we expanded our study to include information visualization, graph analysis, and statistical packages. We examine three aspects of each toolkit: visualization functions, analysis capabilities, and development environments. With respect to development environments, we look at platforms, language bindings, multi-threading/parallelism, user interface frameworks, ease of installation, documentation, and whether the package is still being actively developed. The second stage of our evaluation delves more deeply into a subset of the toolkits and reports our experiences using them to implement a simple visual analytics application. Based on the number and variety of visualization and analysis components found within a package during the first round, we selected Prefuse Flare, Tulip, and Titan for the second round. Our benchmarking application, which was picked based on similar functionality in our second round toolkits, is a document 'explorer' application that allows users to understand and explore a corpus of document relationships.

1. Introduction. With the growth of Visual Analytics, more applications are being written that tightly couple analysis and visualization. Developers of these applications have the choice of leveraging third-party toolkits or implementing desired functionality from scratch. There are many advantages in working with toolkits, yet finding a toolkit that combines the desired characteristics for analysis, visualization, and development platform may not be possible. Open source toolkits provide a solution in which packages that come close to developer needs can be extended or modified. Although individual toolkits have been reviewed at websites such as infosthetics.com, no single reference provides developers with the comparative information they need. However, the motivation for our evaluation stems from the need to understand how our toolkit compares to other similar packages.

Sandia National Laboratories, in collaboration with Kitware, Inc., has developed the Titan Informatics Toolkit. Titan is an open source project built upon the scalable architecture of the Visualization Toolkit (VTK). [28, 21]

2. Prior Work. Laramee [22] provided a practical comparison aimed at assisting developers select the best graphics or visualization library for building their applications through the evaluation of feature sets, software source and development environment, and the ease of implementing a benchmark application. However, the evaluation was limited to an in-depth examination of four pre-selected packages: VTK, Open Inventor, Coin3D, and Hoops 3D. Although we took inspiration from Laramee's approach, we wanted to provide greater breadth by first doing a feature evaluation stage that looked at a wide-ranging set of open source visualization and analysis libraries as a basis for selecting the toolkits used in the implementation stage. Another significant difference is that our evaluation focuses on open source visual analytics toolkits, so we are looking for packages with a combination of visualization and analysis functionality.






3. Functional Comparison. Stage one involved comparing features of twenty-four toolkits and applications to narrow them down to a set for deeper analysis. The sources for finding these toolkits were the Infovis Wiki [16] and Internet searches.

This section describes those areas of functionality that we compared across the toolkits we looked at for the first stage of the evaluation. These include visualization functionality,

*University of New Mexico, jharger@unm.edu

†Sandia National Laboratories, pjcross@sandia.gov

Fig. 3.1. Comparison of Visualization Functionality (Graphs)

	Circular Layout	Radial Layout	Force Directed	Hierarchical Layout	Adjacency Matrix
					
Axiis					
birdeye	X	X	X		
ESTAT					
GeoVista Studio					
Gephi			X	X	
Google Vis. API					
GraphViz	X	X	X	X	
Improvise				X	
Infovis Toolkit	X		X		X
JIT		X	X		
JFreeChart					
JGraphX	X		X	X	
JUNG	X		X		
Mondrian					
NetworkX	X	X	X		
Prefuse	X		X		
Flare	X		X		
Protovis			X		X
R					
Titan	X		X	X	
Tulip	X			X	
VisAD					
WilmaScope			X		
Zest		X	X		

analysis functionality and a look at the development environment used by the toolkit.

3.1. Visualization Functions. Visualization is important for enabling analysts to see relationships in data. Below we list several visualization techniques that are relatively common in modern toolkits and applications. We have divided these into three categories: visualizations of graphs and networks, visualizations of hierarchical trees and visualizations on arbitrary data that can be represented in tabular form.

3.1.1. Graph Layout Views. The *circular graph* layout places all vertices on a circle, and draws edges as lines between the vertices across the inside of the circle. These layouts usually involve a sorting heuristic to group closely related vertices near each other.

A *radial graph* layout requires that a "root" node be chosen, and then vertices are laid out on concentric circles based on their distance from the root.






More common for complex graphs with many possible cycles is the family of *force directed* layouts. These involve running simple spring force simulations on edges as well as repulsion between vertices. Normally done in 2D, these can also be performed in 3D, although that can result in vertices and edges being occluded [6].

Hierarchical [7] graph layouts arrange directed graphs on a 2D plane by using directed relationships and minimizing edge crossovers to choose node position. This can be used with any directed graph, and algorithms typically eliminate cycles in graphs by removing less important edges identified by some heuristic during layout calculations.

A view suited to smaller graphs, the *adjacency matrix* view displays a graphical representation of an adjacency matrix. The matrix is shown as a grid, with vertex labels along the top and side edges. For every edge, the cell corresponding to the two vertices connected is filled in with a color. Different colors can be used to represent different edge properties.

3.1.2. Tree Layout Views. *Tree maps* nest each level of the tree within the rectangular region of its parent. The size of the region depends on some vertex attribute.

FIG. 3.2. *Comparison of Visualization Functionality (Trees)*

					
Axiis	X				
birdeye	X	X	X		
ESTAT					
GeoVista Studio					
Gephi					
Google Vis. API	X				
GraphViz					
Improvise					
Infovis Toolkit	X	X	X		X
JIT	X	X	X		X
JFreeChart					
JGraphX		X			
JUNG		X	X		
Mondrian					
NetworkX					
Prefuse	X	X	X	X	
Flare		X	X		X
Protovis	X	X	X		X
R					
Titan	X	X	X	X	X
Tulip	X	X	X		
VisAD					
WilmaScope					
Zest		X	X		

A *dendrogram* places the root node of a tree on one edge (top, bottom, left or right) and aligns the vertices of each level of the tree together. Unlike the hierarchical graph view, this is much a simpler algorithms and only works with trees.

Similar to the radial graph view, a *radial tree* view places the tree root at the center, and aligns child vertices in concentric circles around it. However, unlike in the general graph case, there are no edge crossovers because of the tree structure, resulting in a cleaner display.

A view similar to the radial tree view is the *balloon tree* and the related *cosmic tree* view. In this view, child vertices are circularly arranged round their parent vertices.

Icicle trees display trees in a descending vertical structure, which results in an appearance much like hanging icicles. The root of the tree is at the top level and the children are drawn to fit within the width of the parent.










3.1.3. Tabular Data Views. *Scatter plots* are plots of two- or three-dimensional data, which is drawn by representing each data point as a point in space. Two-dimensional data is usually shown on a plane, whereas three-dimensional data is shown as a point cloud.

For statistical information, *box and whisker plots* (or simply *box plots*) show more information than standard bar- or line-charts. They display the "five-number summaries" of a population: the sample minimum, lower quartile, median, upper quartile and sample maximum.

Parallel coordinates are visualizations of multidimensional data, which represent the axis for each dimension as a vertical line and high-dimensional points as polylines connecting points on each axis.

Some visualization views are common in spreadsheet software programs. A *bar- or column-chart* is a chart with rectangular bars. The lengths of the bars are proportional to some values. *Line-charts* are a 2D graph where points from the same data set are connected by a line. A *pie chart* is a circular chart containing colored sectors where the arc-lengths of the sectors correspond to the proportion of the quantity to the whole. *Area charts* are similar to line charts, except that the area below the line is filled in.

FIG. 3.3. Comparison of Visualization Functionality (Tabular Data)

	Bar chart	Line chart	Scatterplot	Box plot	Pie chart	Contour Plot	Stacked Bar Chart	Stacked Area Chart	Parallel Coordinates
									
Axiis	X	X	X		X		X	X	
birdeye	X	X	X		X		X	X	X
ESTAT		X	X						X
GeoVista Studio	X	X	X						
Gephi									
Google Vis. API	X	X			X			X	X
GraphViz									
Improvise	X		X		X				
Infovis Toolkit	X	X	X						X
JIT	X				X		X	X	
JFreeChart	X	X	X		X		X	X	
JGraphX									
JUNG									
Mondrian	X		X	X					X
NetworkX									
Prefuse	X	X	X				X	X	
Flare	X	X	X		X		X	X	
Protovis	X	X	X	X	X		X	X	X
R	X	X	X	X	X	X			
Titan	X	X	X		X	X			
Tulip	X	X	X						X
VisAD		X	X			X			
WilmaScope									
Zest									

There are also variations in the previously mentioned charts. One is the *stacked bar-chart*, which is a bar chart where the lengths of the bars are the sum of the data values, and each value is represented as a portion of the total bar. There are also *stacked line-charts* and *stacked area-chart* where the total vertical height at any given horizontal point is the sum of all data values at that point.

3.1.4. Geospatial and Spatio-Temporal Visualizations. *Map overlays* are symbols or polygons drawn on top of a map, either a two-dimensional projection or a three-dimensional globe. *Graduated symbol maps* are similar but the symbols drawn vary in size based on some data set, for example relative populations. *Choropleth maps* use color coding to represent aspects of regions, such as population, crime rate or average incomes.

There are many different map projections available in modern software. *Dymaxion maps* are global maps projected onto a polyhedron and unfolded into a two-dimensional image. *Cartograms* are two-dimensional projections where the area of a region is controlled by a data value. A *three-dimensional globe* is simply a textured sphere that the user can view from any angle.

3.2. Analysis Capabilities. Analytics are also important in modern toolkits. An analyst can calculate and observe properties within the data. We have divided analysis capabilities into two main categories: analysis on graphs and networks, and statistical analysis on arbitrary data.

3.2.1. Graph Analysis. *Adjacency* refers to being able to identify adjacent neighbor vertices. *Accessibility* is determining if one vertex can be reached from another. *Centrality* means calculating measures of relative importance of vertices in a graph. *Common connection* is the process of identifying the set of vertices that can be reached by all of the vertices in a given set. *Connectivity* means identifying connected components in a graph.

FIG. 3.4. *Comparison of Visualization Functionality (Geospatial/Spatio-temporal Data)*







	Map Overlays	Choropleth Maps 	Graduated Symbol Maps 	Map Projections 	Dymaxion Maps 	Cartograms 	3D Globe 
Axiis							
birdeye	X	X	X	X		X	
ESTAT		X					
GeoVista Studio	X						
Gephi							
Google Vis. API	X	X					
GraphViz							
Improvise							
Infovis Toolkit							
JIT							
JFreeChart							
JGraphX							
JUNG							
Mondrian		X					
NetworkX							
Prefuse							
Flare							
Protovis	X	X	X	X	X	X	
R							
Titan							X
Tulip	X						
VisAD							
WilmaScope							
Zest							

FIG. 3.5. *Comparison of Analysis Functionality (Graph)*

	Adjacency	Accessibility	Centrality	Common Connection	Connectivity	Minimum Spanning Tree	Connected Component Analysis
Axiis							
birdeye	X				X		
ESTAT							
GeoVista Studio							
Gephi	X	X	X	X	X		
Google Vis. API							
GraphViz							
Improvise							
Infovis Toolkit	X	X			X		
JIT							
JFreeChart							
JGraphX	X	X			X	X	
JUNG	X	X	X		X	X	
Mondrian							
NetworkX	X	X	X	X	X	X	X
Prefuse	X						
Flare	X		X		X	X	
Protovis							
R							
Titan	X	X	X	X	X	X	X
Tulip	X	X	X		X	X	
VisAD							
WilmaScope							
Zest							

Finding a *minimum spanning tree* for a graph is identifying a tree that is a subgraph of the given graph that has the minimum cost in terms of its edge costs.

Finally, *Connected component analysis* refers to labeling subsets of connected components based on a heuristic.

3.2.2. Statistical Analysis. *Univariate analysis* refers to statistical analysis of single-dimensional data, such as finding the mean and variance, calculating histograms, etc. *Bivari-*

FIG. 3.6. Comparison of Analysis Functionality (Other)

	Univariate	Bivariate	Multivariate	Dimensionality Reduction	Clustering	Latent Semantic Analysis (LSA)	Latent Dirichlet Allocation (LDA)
Axiis							
birdeye							
ESTAT							
GeoVista Studio							
Gephi					X		
Google Vis. API							
GraphViz							
Improvise							
Infovis Toolkit							
JIT							
JFreeChart							
JGraphX							
JUNG					X		
Mondrian							
NetworkX					X		
Prefuse							
Flare					X		
Protovis							
R	X	X	X	X		X	X
Titan	X	X	X	X	X	X	X
Tulip					X		
VisAD							
WilmaScope							
Zest							

ate analysis means statistical analysis of two-dimensional data, such as correlation. *Multi-variate analysis* is any sort of analysis of data in three or more dimensions.

Special purpose techniques include the following: *Dimensionality reduction*, which is reducing the number of dimensions in multivariate data by removing less significant dimensions; *clustering*, which is grouping data points in clusters based on some relatedness criteria; *latent semantic analysis (LSA)*, which uses linear algebra techniques to identify concepts in documents; and *latent dirichlet allocation (LDA)* which uses statistical techniques to identify concepts in documents.

3.3. Development Environment. *Platform* is an important aspect of development. A toolkit can be designed to run on Windows, Mac OS X, Unix-like systems or even a virtual platform like Java or .NET. These days the World Wide Web is becoming more popular for deploying visualization applications, and platforms to accomplish this include basic JavaScript/HTML and Adobe Flex.

The *language* that a toolkit is written in can be useful to know too; most often the application developer must know the implementation language to use the toolkit. These languages are usually limited to C, C++ or Java for desktop applications, and JavaScript or ActionScript for the web. Scripting can be a powerful method for prototyping applications or writing simple one-off programs and these languages can be easier for non-professional programmers to work with [25]. Therefore scripting languages that are directly supported by the toolkits, such as Python and TCL are also listed here.

GUI frameworks are the front-end interfaces available for each toolkit. These include frameworks such as Qt in C++, Swing and SWT in Java, and web GUI interfaces such as HTML and Adobe Flash.

SQL database support comes in two flavors. The most useful is direct toolkit support for loading data from databases. This provides a quick solution to loading data without needing to know much about how databases work. The other is lower level framework support for databases. In this case, the application developer must write the interface to convert between data formats themselves. While this is still a useful feature, it can lead to longer development times than direct integration.

Fig. 3.7. *Development Environment Comparison*

	Platform	Languages	GUI	SQL	File Formats	Documentation
Axiiis	Web (Flex)	ActionScript	Flash	-	-	API Reference, examples
birdseye	Web (Flex)	ActionScript	Flash	-	GeoRSS, GeoML, KML, SHP	Wiki, API Reference, several examples
ESTAT	Java	Java	Swing	-	CSV, SHP	Tutorial, papers
GeoVista Studio	Java	Java	Swing	-	CSV, DBF, SHP	Quickstart guide
Gephi	Java	Java	Swing	MySQL, SQL Server	CSV, Dot, GML, GraphML, GEXL, Tulip, GDF, NET	Wiki, user manual, plugin API reference
Google Vis. API	Web	JavaScript	HTML	-	CSV, JSON, XML	User guide, reference
GraphViz	Linux, OSX, Cygwin	C	-	-	Dot	User guides, papers, DOT language spec.
Improvise	Java	Java	Swing	JDBC	CSV, XML, DBF, SHP	Paper, Ph.D Thesis
Infovis Toolkit	Java	Java	Swing	-	CSV, TM3, Dot, TreeML, GraphML	API reference, unfinished user manual
JIT	Web	JavaScript	HTML	-	JSON	Examples/demos w/ source, API reference
JFreeChart	Java	Java	Swing	JDBC	CSV, XML	Demo app w/ source, API reference, developer guide (\$)
JGraphX	Java	Java	Swing	-	GraphML, GD, VDX	Getting started guide, API reference
JUNG	Java	Java	Swing, SWT	-	GraphML, NET	Tutorial, very short manual, API reference
Mondrian	Java	-	Swing	JDBC	Tab, R	No developer documentation
NetworkX	Python	Python	matplotlib	-	CSV, GML, GraphML, NET	Several tutorials, API reference
Prefuse	Java	Java	Swing	-	CSV, GraphML, TreeML	API reference, unfinished user manual
Flare	Web (Flex)	ActionScript	Flash	-	GraphML, JSON, Tab delim.	Tutorial, API reference
Protovis	Web (SVG)	JavaScript	HTML	-	-	Tutorials, user guide, API reference
R	Windows, Linux, OSX	C, Fortran, R	-	ODBC	Many	User guides, API reference
Titan	Windows, Linux, OSX	C++, Java, Python, TCL	Qt, TK	MySQL, PostgreSQL, Qt	XGML, VTK, Tulip, HDF5, XML, JSON, CSV, Tab	Wiki, VTK user guide, API references
Tulip	Windows, Linux, OSX	C++	Qt	-	GML, Tulip, Dot	Developer guide, incomplete API reference
VisAD	Java	Java	Swing	-	Many	Tutorial, incomplete API reference
WilmaScope	Java	Java	Swing	-	Dot, GML	API reference, plugin developer guide
Zest	Eclipse	Java	SWT	-	-	Tutorial

File format support can be useful for exchanging data with other users or applications, especially when dealing with graphs. There are several formats that support graphs, such as DOT [12], GraphML [3], GML [14] and VTK's graph file format [21].

The *documentation* field lists the available documentation, such as tutorials, user's guides, reference manuals and online API documentation.

4. Benchmark Implementation. Stage two consisted of designing and implementing a simple benchmark application that attempts to demonstrate visualization as well as analysis capabilities. Based on the results of phase one, we chose the three toolkits that appeared to have the most visualization as well as analytic functionality out of those we evaluated in phase one: Prefuse Flare, Tulip and the Titan Informatics Toolkit.

4.1. Benchmark Application. The benchmark application had three basic tasks to perform.

First, it needed to be able to load a graph of documents and links between similar documents based on LSA results. In addition, it needed to be able to load the contents of each document. The data is from another project, and therefore the LSA is not part of this benchmark application.

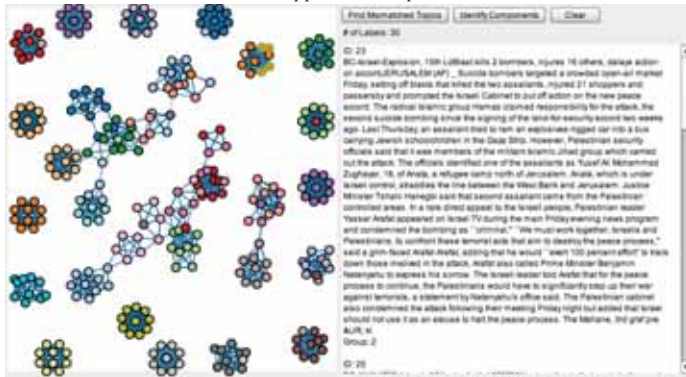
The second task was to display the graph, color-coding the vertices based on the document's concept labeled. The actual layout used was not important, however the vertices and the relationships between them needed to be apparent, so a force-directed layout was chosen for all three implementations.

The application also needed to be able to show the document contents when the vertices were selected in the graph view. This was to show the toolkit's ability to interface with GUI frameworks and load extra data alongside the graph definition.

To demonstrate the analytical capability, we looked for functionality shared between all three packages. We settled on using adjacency and connectivity functions, as described in section 3.2.1. The benchmark application then needed to use the adjacency functionality to

find all adjacent vertices and mark their edges with a different color when the concept labels did not match. The connectivity needed to be used to mark all vertices with an identified component, to compare those to the concept labeling results.

FIG. 4.1. Benchmark application implementation in Flare



4.2. Flare. Prefuse Flare is a visualization and analysis toolkit written in ActionScript for the Adobe Flex 3.0 platform [5]. It is the spiritual successor to the Java-based Prefuse, and contains much of the same functionality [13]. There has not been an official release since January 24, 2009, but this toolkit still had a larger number of visualization and analytics functions than most of the others.

The tutorial document contained not only an introduction to Flare's structure, but also a getting started guide on writing Adobe Flash/Flex applications. The API reference is mostly complete; almost all classes and methods are fully documented.

4.2.1. Importing Data. Originally we were going to use the delimited text parser in Flare, but it was unclear how to convert the resulting general data sets into a graph object. The examples provided all used hardcoded data in the source, so we were unable to use this converter.

Our second choice was to use the built-in JSON parser to convert the data into Flare's internal format. Unfortunately, the exact structure of the JSON objects required is undocumented.

Instead we used the GraphML converter, which handled every aspect of our data. We were able to import document nodes, including the contents of the original documents, as well as the edges between them. However, this method required us to convert our data from the delimited text files to the GraphML format.

4.2.2. Visualization. There were several demos and sample applications showing how to create a force-directed layout, so we implemented the similarity graph with no issues. There are several classes such as the ColorEncoder, which can set the color of a visualization item based on a property, which made the coloring of the vertices and edges a simple task.

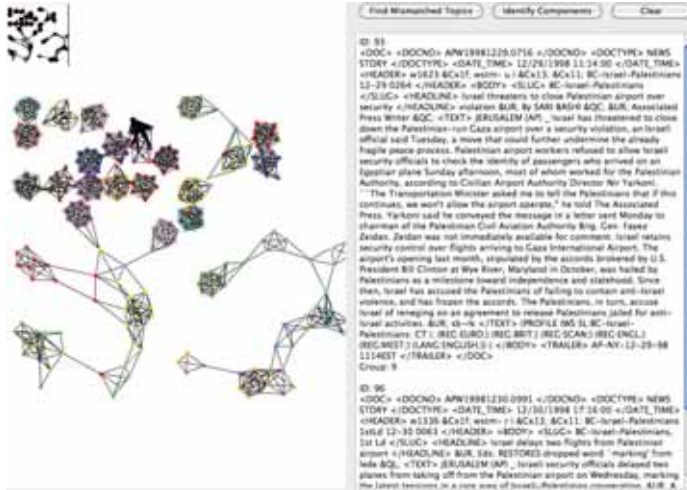
However, selecting vertices in this view was slightly awkward. There needed to be a "hit area" defined to for each vertex, and there was no default value for this. Once the vertex selection was in place, we added a simple Flash text view and set its contents to that of the document contents of the selected vertices.

4.2.3. Analytics. Finding connected vertices using the list of edges was very straightforward. We iterated through all of the edges in the graph, examining the vertices at either

end. The next step was simply applying a ColorEncoder with a filter, and all of the desired edges were highlighted.

Finding the connected components was slightly more complex. Flare does not have a built-in component identification filter, but we were able to use a simple iterative approach using spanning trees to label connected components within the graph.

FIG. 4.2. Benchmark application implementation in Tulip



4.3. Tulip. The implementation of the application in Tulip was the most challenging of the three. The Tulip project consists of two parts, one being the framework application which a user can write plug-ins for, and the second being a C++ library that provides external access to all of the functionality [30].

However, despite the existence of the library, the documentation was focused almost entirely on the application. There was an API reference, but many of the classes and methods were undocumented. Most of the examples provided were for writing plugins. However, by taking a look at the source code of the main application, it was possible to find the information we needed to write the benchmark application.

4.3.1. Importing Data. Tulip does not support loading arbitrary data into its system through delimited text files, so we were forced to convert our graph to GML. However, with the GML importer we were able to import the entire graph plus the document contents using it.

4.3.2. Visualization. Laying out the graph in a view initially appeared to be a simple task. There was a force-directed layout plugin that looked like it would fit our needs perfectly. However, there were several parameters in the function used to invoke this plugin that seemed to be optional due to having default values, and there was no documentation explaining otherwise, but this was not the case for this plugin. Only careful examination of the source for the main application showed the necessary arguments to get the layout working.

Attempting to color the vertices presented a problem. The entire application would crash due to what appeared to be a bad pointer when loading the plugin to color vertices based on a category parameter. We were forced to implement our own method for coloring the vertices and edges.

We used an interactor plugin to implement vertex selection. However, there was no documentation on how to use this plugin in a view. After looking through much of the source

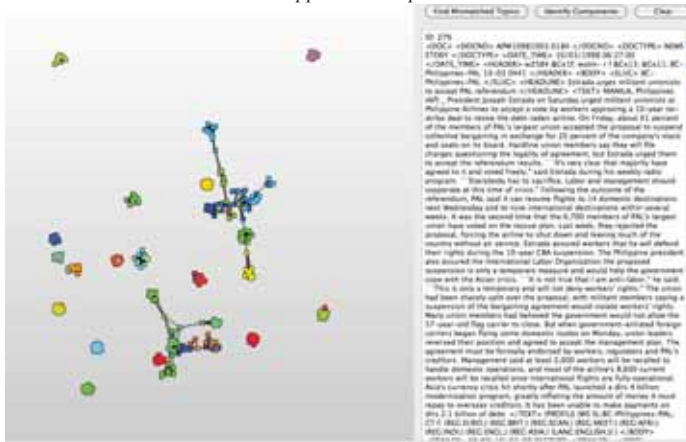
code, we finally found a function that would register an interactor plugin with the view, and then mouse selection was enabled.

Tulip uses Qt as its GUI library, and there were Qt signals [26] defined for selection events. However, these signals were never actually activated in any of the source code, and instead we had to create an observer for the "viewSelection" property of the graph object.

4.3.3. Analytics. Finding adjacent vertices in Tulip was the same process as in Flare. We simply iterated through every edge, and compared the document groups of the vertices at either end. By setting the property we used in our custom coloring implementation, we were able to highlight the edges easily.

Identifying connected components was very easy. Tulip provides a plugin to do exactly this, and it was a one-step process to make it work. The plugin set a property to a component ID, and we used that with our coloring to highlight the vertices in each component.

FIG. 4.3. Benchmark application implementation in Titan



4.4. Titan. Titan has a significantly different architecture than the other toolkits. Instead of calling methods on data as you need to, it uses the same pipeline architecture as VTK [29, 28]. The developer uses a chain of filters to process the data at each step in the pipeline. For example, the output of a table reader is connected to the input of a filter that converts tables to graphs, and the graph output of that filter is connected to the input for a graph layout view.

Since Titan sits on top of VTK, a lot of its functionality actually comes from the VTK library. Because of this, the documentation for these functions is very mature. However, it is a relatively new project, so the documentation for the new functionality that is specific to Titan is not complete, and there are not as many examples to draw from.

4.4.1. Importing Data. We were able to import our tabular data immediately using a few table readers. We connected the output of these to the input ports of a table to graph conversion filter. The output of this we then connected to other filters and then to a graph layout view to display the graph.

4.4.2. Visualization. The graph layout view worked immediately after setting a few properties. We needed to set a property to color the vertices on and to enable vertex coloring. However, the color scheme was a linear mapping, and so document groups that were adjacent in number were also very similar in color and difficult to distinguish. To correct this, we used

a programmable filter to implement a simple shuffling mapping to split up these groups. With this the colors appeared more random, and achieved the results we were looking for.

Vertex selection was a bit more complex. There are mechanisms for communicating selections between views, so selecting an item in a table view can be easily reflected in a graph view, and so on. However, we wanted to show associated text from a document in a simple Qt text browser view, and there was no built-in mechanism for doing that. The solution was a class that adapts selection events to Qt signals, which we were able to use to get the vertex selection and update the text browser view accordingly.

4.4.3. Analytics. To find the mismatched documents, we again created a programmable filter that iterated through all edges and compared concept groups of the vertices at each end. By setting a property used for edge coloring, we were able to highlight the mismatched edges. To find the connected components, we simply connected a filter that identifies components and labels them, and used the property with the label to highlight the components in the graph view.

The pipeline model seems awkward to use in interactive environments. We implemented the analysis functionality by splicing two filters into the pipeline. There were no alternative methods apparent in the documentation.

5. Conclusion. Every toolkit has its strengths and its shortcomings. Some toolkits, such as *Protovis* and *birdeye* clearly offer powerful visualization features. Others, like *R* and *NetworkX* offer strong analysis functionality. Many of these toolkits focus to one particular area, such as graphs/networks, statistics or geospatial analysis. Some are targeted for a particular platform, such as a web browser.

There is no clear winner in the three toolkits we reviewed in depth. *Flare* offers a powerful, interactive experience on a stable and portable web platform. *Tulip* has a powerful desktop application, but the lack of documentation for its library components prevents it from being a true contender. *Titan* brings both visualization and analysis to many platforms covering a broad range of functionality, but its massive size may be intimidating or overwhelming to new users.

REFERENCES

- [1] *birdeye information visualization and visual analytics library*. <http://code.google.com/p/birdeye/>.
- [2] M. BOSTOCK AND J. HEER, *Protovis: A graphical toolkit for visualization*, Visualization and Computer Graphics, IEEE Transactions on, 15 (2009), pp. 1121–1128.
- [3] U. BRANDES, M. EIGLSPERGER, I. HERMAN, M. HIMSOLT, AND M. MARSHALL, *Graphml progress report structural layer proposal*, in Graph Drawing, P. Mutzel, M. Jünger, and S. Leipert, eds., vol. 2265 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2002, pp. 109–112.
- [4] *ESTAT - the Exploratory Spatio-Temporal Analysis Toolkit*. <http://www.geovista.psu.edu/ESTAT/>.
- [5] *Flare: Data visualization for the web*. <http://flare.prefuse.org/>.
- [6] T. M. J. FRUCHTERMAN AND E. M. REINGOLD, *Graph drawing by force-directed placement*, Software: Practice and Experience, 21 (1991), pp. 1129–1164.
- [7] E. R. GANSNER, E. KOUTSOFIOS, S. C. NORTH, AND K.-P. VO, *A technique for drawing directed graphs*, IEEE Trans. Softw. Eng., 19 (1993), pp. 214–230.
- [8] *GeoVista studio*. <http://www.geovistastudio.psu.edu/jsp/index.jsp>.
- [9] *Gephi*. <http://gephi.org/>.
- [10] T. GONZALEZ AND M. VANDANIKER, *Axiis*. <http://www.axiis.org/>.
- [11] *Google Visualization API*. <http://code.google.com/apis/charttools/index.html>.
- [12] *Graphviz*. <http://www.graphviz.org/>.
- [13] J. HEER, S. K. CARD, AND J. A. LANDAY, *prefuse: a toolkit for interactive information visualization*, in CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems, New York, NY, USA, 2005, ACM, pp. 421–430.
- [14] M. HIMSOLT, *GML: A portable Graph File Format*, tech. rep., Universität Passau, 1997.
- [15] *InfoVis Toolkit*. <http://ivtk.sourceforge.net/>.

- [16] *InfoVis:Wiki*. <http://www.infovis-wiki.net/>.
- [17] *JFreeChart*. <http://www.jfree.org/jfreechart/>.
- [18] *JGraphX*. <http://www.jgraph.com/>.
- [19] *JavaScript InfoVis Toolkit*. <http://thejit.org/home/>.
- [20] *JUNG - The Java Universal Network/Graph framework*. <http://jung.sourceforge.net/>.
- [21] KITWARE, INC., *The VTK User's Guide*, Kitware, Inc., 11th ed., March 2010.
- [22] R. S. LARAMEE, *Comparing and evaluating computer graphics and visualization software*, Software: Practice and Experience, 38 (2008), pp. 735–760.
- [23] *Mondrian*. <http://theusrus.de/Mondrian/index.html>.
- [24] *NetworkX*. <http://networkx.lanl.gov/>.
- [25] J. OUSTERHOUT, *Scripting: higher level programming for the 21st century*, Computer, 31 (1998), pp. 23–30.
- [26] *Qt - a cross-platform application and UI framework*. <http://qt.nokia.com/>.
- [27] *The R project for statistical computing*. <http://www.r-project.org/>.
- [28] W. SCHROEDER, K. MARTIN, AND B. LORENSEN, *Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*, Kitware, Inc., 4th ed., December 2006.
- [29] *The Titan Informatics Toolkit*. <http://titan.sandia.gov>.
- [30] *Tulip*. <http://tulip.labri.fr/TulipDrupal/>.
- [31] *VisAD - Visualization for Algorithm Development*. <http://www.ssec.wisc.edu/~billh/visad.html>.
- [32] C. E. WEAVER, *Improvise: a user interface for interactive construction of highly-coordinated visualizations*, PhD thesis, University of Wisconsin - Madison, Madison, WI, USA, 2006. Adviser-Livny, Miron.
- [33] *Wilmascope 3D graph visualisation system*. <http://wilma.sourceforge.net/>.
- [34] *Zest: The Eclipse visualization toolkit*. <http://www.eclipse.org/gef/zest/>.

OPTIKA: A GUI FRAMEWORK FOR PARAMETERIZED APPLICATIONS

KURTIS L. NUSBAUM* AND MICHAEL A. HEROUX†

Abstract. In the field of scientific computing there are many specialized programs designed for specific applications in areas like biology, chemistry, and physics. These applications are often very powerful and extraordinarily useful in their respective domains. However, many suffer from a common problem: a non-intuitive, poorly designed user interface. Many of these programs are homegrown, and the concern of the designer was not ease of use but rather functionality. The purpose of Optika is to address this problem and provide a simple, viable solution. Using only a list of parameters passed to it, Optika can dynamically generate a GUI. This allows the user to specify parameter's values in a fashion that is much more intuitive than the traditional "input decks" used by many parameterized scientific applications. By leveraging the power of Optika, these scientific applications will become more accessible and thus allow their designers to reach a much wider audience while requiring minimal extra development effort.

1. Introduction. In the world of scientific computing there is a problem: most software developers are far more concerned with the functionality of their software rather than their user interface. This is understandable given the limited time and pressures of scientific computing environments. And in cases where there are only a few users of a piece of software this type of development is tolerable. However, when a piece of software starts to be used by a wider audience, poor user interface design issues come to the forefront and can greatly hinder further adoption of a particular piece of software. Optika¹ is an attempt to solve this problem in a generic fashion for parameterized scientific applications.

Since developers of scientific applications don't really care about user interfaces, Optika needs to provide a minimal amount of hurdles for developers. Also, the end result needs to be an intuitive user interface that can be easily navigated and utilized regardless of the underlying computations being done.

The purpose of this paper is to discuss the development of the Optika package. In doing so we hope to demonstrate how Optika solved some of the issues associated with developing a generic user interface for scientific applications and provide justification for why we chose particular solutions. We will proceed to discuss Optika development in a semi-chronological fashion.

2. Initial Planning and Development. In Fall of 2008, Dr. Mike Heroux identified a need for the Trilinos Framework² to include some sort of GUI package. Dr. Heroux wanted to give users of the framework the ability to easily generate GUIs for their programs, while still providing a good experience for the end-user. Based on previous GUI work we'd done for the Tramonto project [6], a few initial problems were identified:

- How would the GUI be laid out?
- What GUI framework would be used to build the GUI?
- Different types of parameters require different methods of input. How would the program decide how to obtain input for a particular parameter?
- How would the application developer specify parameters for the GUI to obtain?
- How would the application developer specify dependencies between parameters. This was a crucial problem/needed-feature that was identified in previous development of an unsuccessful Tramonto GUI.

After some deliberation, the following initial solutions were decided upon:

*St. John's University, klnusbaum@csbsju.edu

†Sandia National Laboratories, maherou@sandia.gov

¹For more information on Optika, please see its documentation [3]

²Optika is part of the Trilinos Project [7].

ParameterList

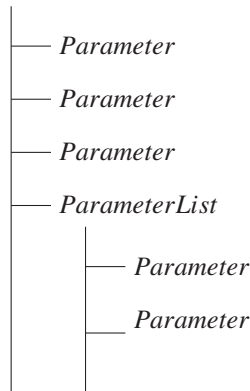


FIG. 2.1. *The hierarchical layout of the GUI*

- The GUI would be laid out in a hierarchical fashion as shown in Figure 2.1. Parameters would be organized into lists and sublists. This would allow for a clear organization of the parameters as well as intrinsically demonstrate the relationships between them.
- QT³ was chosen as the GUI framework for several reasons:
 - It is cross-platform.
 - It is mature and has a comprehensive set of development tools.
 - It has a rich feature-set.
 - It has been used by Sandia in the past.
 - The Optika lead developer was familiar with it.
- It would be required that all parameters specify their type and the following types would be accepted:

<ul style="list-style-type: none"> – int – short – float – double 	<ul style="list-style-type: none"> – string – boolean – arrays of int, short, double, and string
---	---

The supported data type were chosen for two main reasons: (a) The number of input widgets that need to be supported is a direct function of the supported data types. By only supporting a small set of basic data types, we could stick to supporting only a small number of input widgets, most of which were already pre-built and part of Qt. (b) The development team felt that these data types would be adequate for 95% of the developers who would be using Optika.

For number types, a spin box (figure 2.2(a)) would be used as input. If the valid values for a string type were specified, a combo box (figure 2.2(b)) would be used. Otherwise a line edit (figure 2.2(c)) would be used. For booleans, a combo box (figure 2.2(b)) would also be used. For arrays, a pop-up box containing numerous input widgets would be used. The widget type would be determined by the array type (e.g. for numerical types a series of spinboxes would be used).

- Initially it was decided that the application developer would specify parameters via an XML file. A DTD would be created specifying the legal tags and namespaces.

³For documentation on all Qt classes please visit [2].

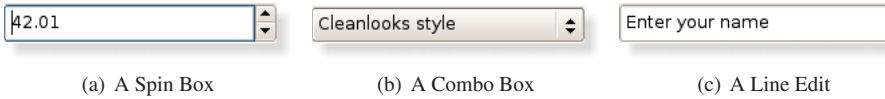


FIG. 2.2. *Some of the various widgets used for editing data [4]*

- Dependencies would be handled through special tags in the DTD.

3. Early Development. The first several months of development were spent on creating and implementing the XML specification. The name of the XML specification went through several revisions but was eventually called Dependent Parameter Markup Language (DPML).

After several months of development we realized that creating an entirely new way of specifying parameters might hinder adoption of Optika. We also realized that Trilinos actually had a `ParameterList` class in the Teuchos [5] package. The `ParameterList` seemed to be better than DPML for several reasons:

- It was already heavily adopted.
- It had the necessary hierarchical nature (like described in figure 2.1).
- It was serializable to and from XML.

For these reasons, DPML was scrapped in favor of using Teuchos's `ParameterLists`. Development moved forward with the goal of creating a GUI framework that, in addition to meeting all the challenges outlined above, would also be compatible with any existing program using Teuchos's `ParameterLists`.

4. Heavy development. Starting in May 2009 a more heavy focus was put on development of the Trilinos GUI package. With the back-end data-structure of the Teuchos `ParameterList` already in place, attention was turned to developing the actually GUI portions of the framework. A key technology provided by Qt was its Model/View framework [1]. Using the Model/View paradigm, a wrapper class named `TreeModel` was created around the `ParameterList` class by subclassing `QAbstractItemModel`.

However, in subclassing the `QAbstractItemModel` it was realized that the `ParameterList` class fell short in terms of providing certain features. The main issue was that a given `ParameterEntry` located within a `ParameterList` or a given sublist located within a `ParameterList` was not aware of its parent. This was an issue because Qt's Model/View framework requires items within a model to be aware of their parents. In order to circumvent this issue the `TreeItem` class was created. Now the `TreeModel` class became more than just a simple wrapper class. A `TreeModel` was created by giving it a `ParameterList`. It would then read in the `ParameterList` and create a structure of `TreeItems`. Each `TreeItem` then contained a pointer to its corresponding `ParameterEntry`. This allowed parent-child relationship data to be maintained while still using `ParameterLists` as the true backend data-structure.

Once the `TreeModel` and `TreeItem` class were complete, an appropriate delegate to go between a view and the `TreeModel` was needed. A new class simply called `Delegate` was created to fill this role by subclassing `QItemDelegate`. As specified above, the delegate would return the appropriate editing widget based on the data type carried within a given `TreeItem`.

With the model and delegate classes in place, an appropriate view could be applied. At first a simple `QTreeView` was applied to the model. Later, as additional functionality was added the view class needed to perform more functions. To fill these needs, the `QTreeView` class was subclassed, creating the `TreeView` class. Its main duties were to show and hide parameters as needed and handle any bad parameter values that might come up during the course of the GUI execution. These features were needed due to requirements that arose from dependencies (something that will be discussed later).

Finally, the OptikaGUI class was created. It had one static function, `getInput`. A `ParameterList` is passed to this function, a GUI is generated, and all end-user input is stored in the `ParameterList` that was passed to the function. When the end-user hits the submit button the GUI closes and the `ParameterList` that was passed to the `getInput` function now contains all of the end-user input. The end result was something like that in figure 4.

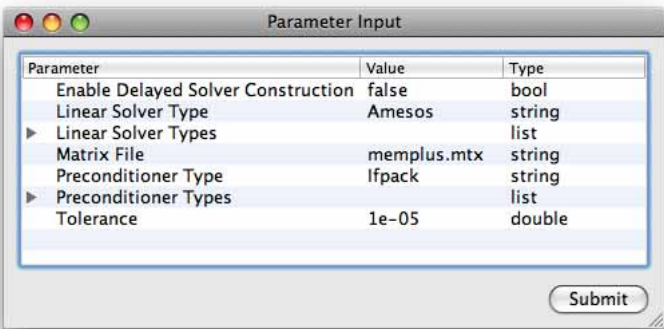


FIG. 4.1. The end result of the initial Optika development

5. Advanced Features. With the basic framework in place, we were now able to move on to more advanced features. As these advanced features were developed various refactorings were made to the already existing code in order to support these new features.

5.1. Validators. One of the goals of Optika is to make life easier for the end-user. It's not enough to simply give the end-user information, it must be conveyed in a meaningful way. Validators are a great way of informing an end-user what the valid set of values for a particular parameter are. Teuchos `ParameterLists` already came with built in validator functionality, but the default validators that were available were sorely lacking in capability. Three initial sets of validators were created to help deal with the short comings of the available validator classes:

EnhancedNumberValidators allowed for validating various number types. `EnhancedNumberValidators` have the following abilities:

- Set min and max.
- Set the step with which the number value is incremented.
- Set the precision with which the number value is displayed.

StringValidator allowed for a parameter to be designated as only accepting values of type string and allowed for specifying a valid list of values.

ArrayValidators allowed for all validator types to be applied to an array of values. The validator that is applied to each entry in the array is called the prototype validator.

A fourth Validator type, a `FileNameValidator`, was added later. This validator designates a particular string parameter as containing a file path and allows the developer to indicate whether or not the file must already exist. Since filenames are such an important type of string, it made sense that they would have their own validator.

By interpreting these validators, Optika could either put certain restrictions on the input widget for a parameter or entirely change the type of input widget used. For instance: with `EnhancedNumberValidators` the min, max, step, and precision of the `EnhancedNumberValidator` are all to directly set their corresponding values in the `QSpinBox` class. But with

the `FileNameValidator` a `QFileDialog` would appear instead of the normal `QComboBox` or `QTextEdit` used for string validators.

5.2. Dependencies. Many times the state of one parameter depends on the state of another. Common inter-parameter dependencies and their requirements include:

Visual Dependencies: One parameter may become meaningless when another parameter takes on a particular value. In this case the end-user no longer needs to be aware of the meaningless parameter and it's best to just remove it from their view entirely so they don't potentially become confused. Visual dependencies should allow the developer to express that "if parameter x takes on a particular value, then don't display parameter y to the end-user anymore."

Validator Dependencies: Sometimes the valid set of values for one parameter changes if another parameter takes on a particular value. Validator Dependencies should allow the developer to express that "if parameter x takes on a particular value, change the validator on parameter y."

Validator Aspect Dependencies: Sometimes the developer doesn't want to change the validator on a particular parameter, but rather just a certain aspect of it. Validator Aspect Dependencies should allow the developer to express that "if parameter x takes on a particular value, change this aspect of the validator on parameter y based on the new value of parameter x"

Array Length Dependencies: Sometimes the length of an array in a parameter changes based on the value of another parameter. Array Length Dependencies should allow the developer to express that "if parameter x changes its value, change the length of the array in parameter y based on the new value of parameter x."

Coming up with a way for the developer to easily express these concepts was not a simple task. The first problem that had to be solved was how to keep track of all the dependencies. They couldn't just be stored in a `ParameterList` as a class member because of the recursive structure of `ParameterLists`. Eventually, it was decided that a new data structure called a `DependencySheet` would hold all the dependencies used for a certain `ParameterList`. Each `Dependency` would at minimum specify the dependent parameter and the dependee parameter. However, a complication arose. Because we wanted dependencies to be able to have arbitrary dependents and dependees, we needed a way to uniquely identify the dependee and the dependent. As the `ParameterEntry` class stood, there was no way of doing this and we didn't want to add this capability to the `ParameterEntry` class. The `Teuchos` package is fundamental to the `Trilinos` Project and before we started changing it for our purposes we wanted `Optika` to have a solid footing and be absolutely sure that any changes made to `Teuchos` were actually necessary. We needed to find another way to uniquely identify parameters within a `ParameterList`.

We decided to use the name of the parameter as the identifier because the accessor functions for a `ParameterList` usually use the parameter's name to identify a particular parameter. While within a `ParameterList` names of parameters are unique, names are not necessarily unique across a set of sublists. Therefore, in order to uniquely identify a parameter and allow dependencies across sublists `Optika` would need to know both the parameter name and the parent list containing it⁴.

So it became that every dependency, along with needing the names of the dependee and dependent, also needed their respective parent lists. The `DependencySheet` also needed

⁴The astute reader will notice that if there are two sublists with different parent lists and each sublist has a parameter with the same name, then `Optika` will not be able to uniquely identify the dependent and the dependee. Since solving this problem would most likely require a lot of refactoring of code not directly part of the `Optika` package, we decided to address it at a later date.

the root list which contained all of the dependees and dependents. This was so Optika could recursively search for the parameters and their parent sublists (the only way to find them using our method of identification). The following Dependency classes were created to address the use cases above (shown as a hierarchy of classes):

Dependency: Parent class for all Dependencies.

NumberArrayLengthDependency: Changes an array's length.

NumberValidatorAspectDependency<T>: Changes various aspects of an Enhanced-NumberValidator.

ValidatorDependency: Changes the validator used for particular parameter.

BoolValidatorDependency: Changes the validator used for a particular parameter based on a boolean value.

RangeValidatorDependency<T>: Changes the validator used for a particular parameter based on a number value.

StringValidatorDependency: Changes the validator used for a particular parameter based on a string value.

VisualDependency: Shows or hides a particular parameter.

BoolVisualDependency: Shows or hides a particular parameter based on a boolean value.

NumberVisualDependency<T>: Shows or hides a particular parameter based on a supported number type value.

StringVisualDependency: Shows or hides a particular parameter based on a string value.

Some of these dependencies have fairly novel and robust capabilities. Namely, the NumberArrayLengthDependency, NumberValidatorAspectDependency, and NumberVisualDependency can all take a pointer to a function as an argument. In the case of the NumberArrayLengthDependency, this function can be applied to the value of the dependee parameter. The return value of this function is then used as the length of the array for the dependent parameter. For NumberValidatorAspectDependencies, the function is applied to the dependee value and used to calculate the value of the chosen validator aspect. In the NumberVisualDependency class, if the function when applied to the dependee value returns a value greater than 0 the dependent is displayed. Otherwise, the dependent is hidden.

The algorithm for expressing dependencies in the GUI is as follows:

1. A parameter's value is changed by the end-user.
2. The Treemodel queries the associated dependency sheet to see whether or not the parameter that changed has any dependents.
3. If the parameter does have dependents, the Treemodel requests a list of all the dependencies in which the changed parameter is a dependee.
4. For each dependency, the evaluate function is called. The dependency makes any necessary changes to the dependent parameter and the Treemodel updates the Treeview with the new data.
5. If any dependents now have invalid values, focus is given to them and the end-user is requested to change their value to something more appropriate.

The order in which dependencies are evaluated is arbitrary. We have not tested what happens under the conditions of conflicting dependencies or circular dependencies yet. This is an area for further study.

5.3. Custom Functions. Normally, in Optika the end-user configures the ParameterList, hits submit, the GUI disappears, and the program continues with execution. However, an alternative to this work flow was desired. A persistent GUI was needed. The development team added the ability to specify a pointer to a function that would be executed whenever the end-

user hit submit. The function was required to have the signature *foo(Teuchos::RCP<const ParameterList> userParameters)*.

5.4. Various Niceties. Various niceties were added to the GUI as well. The ability to save and load ParameterLists was added. The Optika GUI class was expanded to allow for customization of the window icon and use of Qt Style Sheets to style the GUI. Checks were also added to see if the end-user was trying to exit the GUI without saving. In such a case they would be warned and given the option to save their work. Tooltips were added so when ever the end-user hovered over a parameter, it's documentation string would be displayed. Also, the ability to search for a parameter was added.

6. Waiting For Copyright. All of the above features were completed around or shortly after the end of August 2009 and Optika was officially given its name. Optika was then submitted for copyright. It took Optika a little over six months to complete copyright. Since it was not yet copyrighted, it could not be included in the Trilinos 10 release in October 2009. During the time Optika spend in copyright limbo, little development on Optika was done. Most of development was cleaning up various pieces of code, adding examples, and adding documentation. Finally, in March 2010 Optika completed copyright and was ready to be included in Trilinos. It was released to the public with the Trilinos 10.2 release.

7. User Feedback. In the summer of 2010, Optika got it's first user. Dr. Laurie Frink began using Optika to create a GUI for Tramonto. There had been a previous attempt to build a GUI for Tramonto, but it had been largely unsuccessful

Initial feedback was very positive. Dr. Frink was very impressed with the capabilities of Optika and the ease at which should could construct a GUI. However, she did have some small initial issues picking up Optika. But most of them arose from the fact she is a C programmer and Optika is C++ based. Her issues were always easily and quickly addressed. Some of her more involved questions even lead to the creation of some great examples.

For the most part Dr. Frink found Optika to be quite adequate for her purposes. However, she did have one rather major feature request: she needed the ability to specify multiple dependents and in some cases even multiple dependees. This was quite a task and required a large reworking of the Dependency part of the framework.

Adding support for multiple dependents was fairly trivial. Instead of specifying a single dependent to the constructor of a Dependency, a list of Parameters was now passed. If the developer only needed one dependent then he/she could just pass a list of length one. This simple list worked in the case of all the dependents having the same parent list. If they had different parent lists, then a more complex data structure which mapped parameters to parent lists would be used. Convenience constructors were also made for simple cases where just one dependent was needed. The algorithm used for evaluating dependencies changed very little with these modifications. The only addition needed was an extra loop for evaluating each dependent in a dependency for a given dependee.

Adding support for multiple dependents was much harder. There was actually only one specific use case where multiple dependents were needed or even appropriate for that matter. Dr. Frink needed the ability to test the condition of multiple parameters to determine whether or not a particular parameter should be displayed. So a new VisualDependency class called ConditionVisualDependency was created. ConditionVisualDependencies evaluated a condition object to determine the whether or not a set of dependents should be hidden or shown. The set of condition classes created are as follows (shown as a hierarchy of classes):

Condition : Parent class of all conditions.

ParameterCondition : examines the value of a particular parameter and evaluates to true or false accordingly. Types of ParameterConditions include:

BoolCondition: examines boolean parameters.

NumberCondition<T>: examines number parameters.

StringCondition: examines string parameters.

BinaryLogicalCondition: examines the value of two or more conditions passed to it and evaluates to true or false accordingly. Types of BinaryLogicalConditions include:

AndCondition: returns the equivalent of performing a logical AND on all conditions passed to it.

EqualsCondition: returns the equivalent of performing a logical EQUALS on all conditions passed to it.

OrCondition: returns the equivalent of performing a logical OR on all conditions passed to it.

NotCondition: examines the value of one condition passed to it and evaluates to the opposite of what ever that condition evaluates.

Through the recursive use of BinaryLogicalConditions the developer can now chain together an arbitrary amount of dependents.

ConditionVisualDependencies are the only dependencies which allow for multiple dependents. So while support was added for multiple dependents at the Dependency parent class level, ConditionVisualDependency is the only class which actually implements the functionality. In the case of multiple dependents the algorithm for evaluating dependencies didn't need to change at all.

At the time of this publication, the Optika team is still waiting to hear back from Dr. Frink as to whether or not these new features meet her needs.

8. Future Development. There are several main development goals for Optika in the near future.

- We would like to give users the option to completely write an Optika GUI (with dependencies and validators) solely in XML. Our hope is that since writing XML is simpler than writing C++ this will further ease the demands of the application developer. In order to achieve this goal XML serialization for all of the validator and dependency related classes must be developed. We have already started working on serialization this summer and currently XML serialization for validators is almost finished after which serialization for the dependency and dependency sheet class will begin.
- We would like to develop a stand-alone version of Optika. The development team believes that the potential audience for Optika is much larger than just the user base of Trilinos. However, creating a stand-alone version presents the problem of keeping source code consistent between the Optika that exists in Trilinos and the stand-alone version. This is an issue that we will need to make sure to address.
- We would like to create a single Optika based executable that acts as a generic ParameterList configurator. It would take in a ParameterList in XML format, allow the user to configure the ParameterList, and then either output the entire ParameterList again with the new settings or output a ParameterList only containing the parameters that were changed. We think this will be useful because it will enable end-users to utilize Optika without requiring the program their using to implement Optika support (just ParameterList support).
- An absolute and simpler way to uniquely identify dependees and dependents is needed. The current system right now is clunky and doesn't offer us guaranteed identification.
- Do a more thorough use case analysis. Initially, only a few people were consulted on

the potential requirements of such a generic GUI. A broader selection of potential users would help us determine how we can make Optika more useful in a much boarder context.

- We need to further investigate what happens when dependencies conflict or become circular. Right now the behavior of Optika under such conditions is unknown.

9. Conclusions. It is difficult to tell if Optika has met it's initial goals yet. As of the writing of this paper, Optika only has one user. Hopefully, by continuing to do further development and evangelization it's user base can grow. This will then allow us to see if we truly are meeting the needs of the scientific community. Based on early use of Optika by Dr. Frink we believe that Optika is indeed robust enough to meet most of the community's needs but we can't say for sure until we have more user testing.

REFERENCES

- [1] *Model/View Programming*. <http://doc.trolltech.com/4.6/model-view-programming.html>.
- [2] *Online Reference Documentation*. <http://doc.trolltech.com>.
- [3] *Optika: Trilinos/packages/optika*. <http://trilinos.sandia.gov/packages/docs/r10.4/packages/optika/doc/html/index.html>.
- [4] *Qt Widget Gallery*. <http://doc.trolltech.com/4.6/gallery.html>.
- [5] *Teuchos: The Trilinos Tools Package*. <http://trilinos.sandia.gov/packages/docs/r10.4/packages/teuchos/doc/html/index.html>.
- [6] *Tramonto Software*. <http://software.sandia.gov/DFTfluids>.
- [7] M. A. HEROUX, R. A. BARTLETT, V. E. HOWLE, R. J. HOEKSTRA, J. J. HU, T. G. KOLDA, R. B. LEHOUCQ, K. R. LONG, R. P. PAWLOWSKI, E. T. PHIPPS, A. G. SALINGER, H. K. THORNQUIST, R. S. TUMINARO, J. M. WILLENBRING, A. WILLIAMS, AND K. S. STANLEY, *An overview of the trilinos project*, ACM Trans. Math. Softw., 31 (2005), pp. 397–423.

Appendix

A. Nomenclature.

Dependency A relationship between two or more parameters in which the state or value of one set of parameters depends on the state or value of another.

Dependee The parameter upon which another parameter's state or value depends.

Dependent A parameter whose state or value is determined by another parameter.

Parameter An input needed for a program.

ParameterList A class containing a list of parameters and other parameter lists.

ParameterEntry A class containing a parameter located in a ParameterList

RCP Reference counted pointer. RCPs referred to in this document reference the RCP class located in the Teuchos [5] package of Trilinos.

Sublist A parameter list contained within another parameter list.

Widget A GUI element, usually used to obtain user input.

Validator An object used to ensure a particular parameter's value is valid.

EPETRA/AZTECOO AND RELATED TO TPETRA/STRATIMIKOS AND RELATED: A CONVERSION GUIDE

KELLY J. FERMOYLE[¶] AND MICHAEL A. HEROUX^{||}

Abstract. Tpetra gives users additional flexibility for using different data types and computing platforms for example allowing users to choose floats instead of doubles for the scalar type or use a long integer for practically limitless size of operators. The flexibility in platforms allows for use with graphics processing units or even hybrid systems that mix both systems. Tpetra was designed to be easily transitioned from experienced Epetra users or even new users. We describe some design differences between Tpetra and Epetra including the typedef keyword and the use of Teuchos utilities. This guide gives examples with side-by-side comparisons of common classes. The examples can be compiled to create a simple tri-diagonal solver in both Epetra and Tpetra. This guide begins with the most essential classes and continues to classes that use these base classes. We then explain the steps needed to wrap the objects in Thyra and create a Stratimikos solver.

1. Introduction. Templated code allows programmers much more flexibility to easily use data types that are most appropriate for their code. This is especially important in the design of library code that is used for many different applications. Templated design is a relatively new paradigm, and Trilinos is one of the first scientific libraries to support templated programming [4].

Tpetra allows for the use of new computing environments including graphics processors and the IBM cell processor [5, 6]. These new platforms can deliver incredible computing rates, but many existing code bases cannot easily be ported to harness this power. Tpetra is templated on the node type, which allows the programmer to specify that a Thrust GPU node will be used, for example. And the code would be no different than if a serial CPU were used since the node type is set automatically.

The rest of this guide proceeds as follows. Section 2 explains some of the design decisions in Tpetra as well as tips to manage code and understand the rest of the guide. Section 3 describes the differences between Epetra and Tpetra classes and includes examples of creating them and using important methods. Section 4 describes the steps used to wrap objects in Thyra solve linear systems using Stratimikos. Finally, section 5 concludes the guide.

Example 1 Header.hpp: The keyword ‘using’ allows us to avoid the namespace Teuchos before RCP. We also typedef the scalar and ordinal types.

```
#include "Teuchos_RCP.hpp"
2 // To avoid providing the namespace Teuchos with RCP or rcp
  using Teuchos::RCP;
4 using Teuchos::rcp;

6 // Typedef here makes it clear what the scalar and ordinal are
  // All scalars and ordinals can be changed to i.e. float in one line
8 typedef double Scalar;
  typedef int LO;
10 typedef int GO;
```

2. Background. This section combines some tips to make the code more manageable as well as some of the standardized features used in Tpetra. Typedefs are used to keep the code looking neat. Reference counted pointers are used for ease in memory management and

[¶]Penn State University Computer Science and Engineering Department, kjf198@cse.psu.edu

^{||}Sandia National Laboratories, maheroux@sandia.gov

avoiding memory leaks with automatic garbage collection. Finally, we discuss the design decisions that led to using Teuchos array objects instead of pointers to scalars.

Example 2 TpetraTest.cpp: The driver program for each of the examples in this guide. This shows the order that the guide proceeds. MPI_Init is only needed for Epetra. We use RCP for Tpetra objects which uses automatic garbage collection so they do not need to be deleted.

```

#include "Header.hpp"
2 #include "CommTest.hpp"
#include "MapTest.hpp"
4 #include "VectorTest.hpp"
#include "MatrixTest.hpp"
6 #include "SetMat.hpp"
#include "ImportTest.hpp"
8 #include "ThyraTest.hpp"
#include "SolveTest.hpp"
10 #include "ArrayTest.hpp"

12 int main(int argc, char* argv[]){
#include EPETRA_MPI
14 MPI_Init(&argc,&argv);
#include
16 CreateComm(); CreateMap(); CreateVec(); CreateMat(); CreateImp();

18 RandomizeVec(); UseImp(); SetMat(); WrapThyra(); Solve(); ArrayTest();
delete epetra_comm; delete epetra_map; delete epetra_vector;
20 delete epetra_lhs; delete epetra_multivector; delete epetra_matrix;
delete epetra_import; delete epetra_export; delete epetra_solver;
22 MPI_Finalize();
}
24

void SimpleConstructor(){
26 // This code shows the difference between using a pointer
// and the simple objects.
28 Epetra_Map *pointer = new Epetra_Map(NumGlobalElements, 0, *epetra_comm);
Epetra_Map simple(NumGlobalElements, IndexBase, *epetra_comm);
30 // The simple object does not use '*' and no call to new
simple.MaxMyGID(); // Uses '.' instead of '->'
32 delete pointer; // We do not delete simple because it is not a pointer
RCP<MAP> tpetra_m = rcp(new MAP(NumGlobalElements, 0, tpetra_comm));
34 // Tpetra constructors look more like epetra with pointers
// tpetra_m is deleted automatically when it goes out of scope
36 }

```

A few notes about the example program. We use header files to easily display each portion of code in Examples 3-11 separately. Normally, a program would not be setup as independent files but the function calls remain the same. Because of this, we need to use global pointers to the objects so that they are available to each method in different header files. Many users use the simple constructor rather than a call to new; Example 2 shows the use of the simple constructor on line 25. The use of pointers means calls to function will use `->function()` rather than `.function()`.

2.1. Define types to clean up code using typedef. Tpetra introduces a number of template parameters needed to define certain classes. Using all or most of them can quickly make the code unmanageable and hard to read. The use of **typedef** can make the code easier to write and also more readable to everyone else. This guide uses typedef for the scalar and

ordinal types throughout the code. Example 1 lists this simple use of typedef in lines 8-10. In addition, each of the basic types use typedef to hide the parameters and make the code more legible. An example of this can be found in Example 5 on line 4. This line uses typedef to shorten the declaration of the Map type to simply MAP, and uses the passed template parameters. Lines 7 and 13 show the use of this map typedef.

Example 3 ArrayTest.hpp: Tpetra uses Array, ArrayView, and ArrayRCP in place of raw pointers.

```

void ArrayTest(){
2   Teuchos::Array<Teuchos::Array<Scalar>> data(NumVectors);
   Teuchos::Array<Teuchos::ArrayView<Scalar>> dataView(NumVectors);
4
   // We need to allocate data with Teuchos::Array::resize()
6   for(size_t i=0; i < NumVectors; i++){
       data[i].resize(NumGlobalElements);
8       dataView[i] = data[i];
   }
10  // But we need to pass the function Teuchos::ArrayView
   tpetra_multivector->get2dCopy(dataView);
12
   // View methods use Teuchos::ArrayRCP for a persisting view
14   Teuchos::ArrayRCP<Teuchos::ArrayRCP<Scalar>> view
       = tpetra_multivector->get2dViewNonConst();
16 }

```

2.2. Teuchos features in Tpetra. Tpetra makes extensive use of reference counted pointers (**RCP**) as defined in the Teuchos namespace to pass objects. Example 5, line 13 shows the use of RCP for both the constructor of the Tpetra::Map and as the parameter for passing the Teuchos::Comm. In addition to passing objects for construction, RCPs are used in many of the get methods for the communicator or map or other objects as the pointer for the return type. For example, the Tpetra::Map function getComm() returns an RCP to the communicator. Example 1 includes the necessary header file and lines 3 and 4 tell the compiler that we are using RCP so we do not need to include the Teuchos namespace. Much more information, including examples on RCPs can be found in the beginner's guide [2].

Using C style pointers can be tedious and lead to memory errors, segmentation faults, or memory leaks. Tpetra was designed to avoid these problems as much as possible by using advanced features in Teuchos that hide much of the tricky coding needed for passing pointers. It uses Teuchos::ArrayView and Teuchos::ArrayRCP in much the same way that double* is used in Epetra. For users familiar with the standard template library in C++, these memory management classes should be easy to work with.

Example 3 shows the use of Teuchos::ArrayView and Teuchos::ArrayRCP with the Tpetra::MultiVector class. The code here is called after the multivectors are created, but shown now to demonstrate the use of ArrayView and ArrayRCP. This example first allocates space and gets a copy of the data. This data is not persistent, meaning that if the multivector values change, the copy will not reflect these changes. Next, the example gets a view of the data with get2dViewNonConst. This view is persistent, so changes will be reflected in the ArrayRCP. The example first creates two objects, the data variable is an array of arrays, and the copy variable is an array of array views. ArrayView is used as a lightweight wrapper for the array class. We create both objects because Tpetra needs an ArrayView, but the space is allocated with Array. There is an implicit conversion from Array to ArrayView, but it cannot

be converted from within the Array. The loop is used to allocate space in data, and then copy is used to point to the allocated space. Finally copy is passed as the argument to the method. Either variable can be used from this point on to access the data.

Next, `ArrayTest` shows the use of `Teuchos::ArrayRCP`. `ArrayRCP` is used for persisting views of data in Tpetra classes. Space does not need to be allocated for this type because the data it points to already exists.

2.3. LocalOrdinal vs. GlobalOrdinal vs. size_t. This section explains the difference between the ordinal types used in Tpetra. `LocalOrdinal` is used when referring to data that is on this processor. `GlobalOrdinal` is used for data that could be on any processor. For example, the return type for `Map::getGlobalElement` will be `GlobalOrdinal` because the element could be on any processor. This function returns the element defined in a global sense when given its local index. By comparison, `Map::getMaxLocalIndex` is `LocalOrdinal` because the largest element on this processor must be within the bounds of `LocalOrdinal`. The index of the largest local element also has a global equivalent that would be a `GlobalOrdinal` when used globally.

The cardinality of Tpetra objects are described by either `global_size_t` or `size_t` depending on if the cardinality is across all processor or just on the local processor. Again the difference between these type is whether or not the elements are on-processor or not. There is not a type `local_size_t`, it is just called `size_t`. An example in `CrsMatrix` is `global_size_t` is returned from `getGlobalNumRows` because they are on any processor and it is the cardinality, meaning the number of rows. The local size equivalent is `getNodeNumRows` which returns `size_t`.

When converting code from Epetra to Tpetra, it is often helpful to use the context of how a variable is used to determine if it is local or global. It might not be immediately obvious by a variable's name or simple usage if it should be global or local, but the context should make it apparent.

3. Epetra vs. Tpetra Basic Types. Here we discuss the important features that have changed during the development of Tpetra. Many of the types in Tpetra are very similar to Epetra, but users should be aware of what has changed. This guide will proceed from the most basic type, communicators in 3.1 to the more complicated types of matrices and import objects in 3.4 and 3.5, respectively. Example 2 shows the include statement for each of the following examples as well as the order of calls to functions.

3.1. Communicators. The first critical difference between Epetra and Tpetra is the way objects communicate. Epetra objects use an `Epetra_Comm` to learn their rank and the number of processors etc. Tpetra, on the other hand, does not use an `Epetra_Comm`, but a communicator from the Teuchos namespace. `Teuchos::Comm` is templated using an `Ordinal`, and in most cases will use `int`. Overall, it is used like the Epetra version and mostly used just for the construction of other objects.

Creating a `Teuchos::Comm` object is different than just declaring it either `Serial` or `MPI` in Epetra. The preferred method is to use the platform in Tpetra to create the communicator. A communicator could be instantiated using standard constructors and determining manually if MPI is installed, but the developers recommend using the related function to create the object.

Note that Example 4 creates the communicator in a reference counted pointer in line 24. The Epetra version requires the user to determine if the program is run with MPI installed using the compiler definitions as seen in lines 17-21, and this is not needed in Tpetra. Similarly, we no longer need to initialize MPI because this is taken care of in the background when appropriate.

3.2. Maps. Creating maps in Tpetra is similar to Epetra. The primary differences are that Tpetra uses RCP as seen in the communicator and that `Tpetra::Map` uses three template

Example 4 CommTest.hpp: Tpetra::DefaultPlatform determines if MPI is installed or not, so we do not need to use compiler flags.

```

1  #include "Epetra_Comm.h"
2  #ifdef EPETRA_MPI
3  #include "mpi.h"
4  #include "Epetra_MpiComm.h"
5  #else
6  #include "Epetra_SerialComm.h"
7  #endif
8  #include "Teuchos_Comm.hpp"
9  #include "Tpetra_DefaultPlatform.hpp"
10
11 typedef Teuchos::Comm<int> COMM;
12
13 const Epetra_Comm* epetra_comm;
14 RCP<const COMM> tpetra_comm;
15
16 void CreateComm(){
17     #ifdef EPETRA_MPI
18         epetra_comm = new Epetra_MpiComm(MPLCOMM_WORLD);
19     #else
20         epetra_comm = new Epetra_SerialComm();
21     #endif
22
23     // This determines if MPI is installed or not
24     tpetra_comm = Tpetra::DefaultPlatform::getDefaultPlatform().getComm();
25 }

```

Example 5 MapTest.hpp: Use the typedef to simplify the constructor and it is easy to create the map.

```

1  #include "Epetra_Map.h"
2  #include "Tpetra_Map.hpp"
3
4  typedef Tpetra::Map<LO, GO> MAP;
5
6  Epetra_Map* epetra_map;
7  RCP<MAP> tpetra_map;
8  GO NumGlobalElements = 10;
9  LO IndexBase = 0;
10
11 void CreateMap(){
12     epetra_map = new Epetra_Map(NumGlobalElements, IndexBase, *epetra_comm);
13     tpetra_map = rcp(new MAP(NumGlobalElements, IndexBase, tpetra_comm));
14 }

```

parameters. The first is the local ordinal, the whole number system used for indices on the local processor. The second is the global ordinal, which is similar except used to index elements on any processor. By default the global ordinal is set to the same type. Finally, Tpetra::Map uses a node parameter, which gives the ability to use different computing platforms. In most cases, users will allow this parameter to take the default value that is set by Kokkos::DefaultNode. The default value is determined based on which platform is installed with Trilinos. Example 5 shows the construction of the maps with integer ordinals and the

default node platform chosen.

The design of Tpetra::Map is different than Epetra because the BlockMap class was not carried over. Most of the methods of Tpetra::Map are used like they are in Epetra. The Epetra function NumGlobalElements() for example is getGlobalNumElements() in Tpetra. The noteworthy differences are that Tpetra uses keywords like get and set to specify if it is retrieving or setting parameters and also that the first letter of methods are not capitalized in Tpetra as a standard. Another difference is that the parameter types stem from the parameter types that were set on creation. So, the return type of global element counts is global_size_t and similarly the maximum global element would return type GlobalOrdinal. This difference will often not be noticeable to users, but does allow for greater flexibility and makes more obvious when a return type refers to the global or local space.

Example 6 VectorTest.hpp: The constructor and methods of the vector type are similar to Epetra.

```

1 #include "Epetra_Vector.h"
2 #include "Epetra_MultiVector.h"
3 #include "Tpetra_Vector.hpp"
4 #include "Tpetra_MultiVector.hpp"

6 typedef Tpetra::Vector<Scalar, LO, GO> VEC;
7 typedef Tpetra::MultiVector<Scalar, LO, GO> MV;
8
9 Epetra_Vector* epetra_vector;
10 Epetra_MultiVector* epetra_multivector;
11 RCP<VEC> tpetra_vector;
12 RCP<MV> tpetra_multivector;
13 Epetra_Vector* epetra_lhs;
14 RCP<VEC> tpetra_lhs;
15 size_t NumVectors = 1;
16
17 void CreateVec(){
18     epetra_vector = new Epetra_Vector(*epetra_map);
19     epetra_multivector = new Epetra_MultiVector(*epetra_map, NumVectors);
20     epetra_lhs = new Epetra_Vector(*epetra_map);
21
22     tpetra_vector = rcp(new VEC(tpetra_map));
23     tpetra_multivector = rcp(new MV(tpetra_map, NumVectors));
24     tpetra_lhs = rcp(new VEC(tpetra_map));
25 }
26
27 void RandomizeVec(){
28     epetra_vector->Random();
29     // simple_vector.Random();
30     epetra_multivector->Random();
31     tpetra_vector->randomize();
32     tpetra_multivector->randomize();
33 }

```

3.3. Vectors and MultiVectors. Tpetra::Vector introduces one more template parameter that needs to be set upon creation. It uses a scalar template for the storage of each value within the vector or multivector. This allows the user to specify that float would be used instead of double for example. The float type may be very useful in GPU programming because they operate much faster with the 32-bit type compared to the 64-bit type. Besides this

extra template, creating a `Tpetra::Vector` or `Tpetra::MultiVector` follows from the pattern of the communicator and map. Example 6 creates the vector with zeroed-out values and the map defined earlier. It also uses local and global ordinals of integer type and the scalar is the double precision type. The function `RandomizeVec()` shows a simple function call that sets all values to random numbers. The only difference between Epetra and Tpetra are the case of the first letter and the conjugation of the verb.

Another difference between Epetra and Tpetra is the use of array utilities in Teuchos in place of pointers. Section 2.2 discusses this change in more detail. For the `Vector` and `MultiVector` classes this means that instead of setting values using a double-pointer, we can use a `Teuchos::ArrayView` of any templated type.

Example 7 `MatrixTest.hpp`: There is very little difference in creating a `CrsMatrix` in Epetra and Tpetra.

```

1  #include "Epetra_CrsMatrix.h"
   #include "Tpetra_CrsMatrix.hpp"
3
   typedef Tpetra::CrsMatrix<Scalar, LO, GO> MAT;
5  Epetra_CrsMatrix* epetra_matrix;
   RCP<MAT> tpetra_matrix;
7  size_t NumEntriesPerRow = 3;

9  void CreateMat(){
   epetra_matrix = new Epetra_CrsMatrix
11    (Copy, *epetra_map, *epetra_map, NumEntriesPerRow);
   tpetra_matrix = rcp(new MAT(tpetra_map, tpetra_map, NumEntriesPerRow));
13 }

```

3.4. Matrices. Matrices in Tpetra are most commonly used with a compressed row storage matrix format, this storage format is implemented in `Tpetra::CrsMatrix`. Like the Epetra version, the Tpetra matrix uses the pure virtual interface `RowMatrix` to specify methods needed for the matrix. `RowMatrix` inherits `Operator` just like Epetra which allows it to be used as the linear operator in linear systems. Example 7 creates the Epetra and Tpetra `CrsMatrix` objects. The `Epetra_DataAccess` enumerated type was not carried over to the Tpetra version; it is assumed that the data will be copy mode. This design decision is the result of the need for compatibility with CUDA GPU programming. GPU programming requires that the data be copied over to the GPU memory, so a view of it is not possible in this case. Otherwise, they are created almost identically with the domain and range maps (RCP in Tpetra) and the number of entries per row.

Example 8 fills the matrix in the method `SetMat()`. The main difference between Epetra and Tpetra when filling a matrix is the use of `Teuchos::ArrayView` instead of the raw pointer. The `Teuchos::Array cols` is like the integer pointer `Indices`. Note that the `Array` does not call `new`, and then does not need to be deleted. The other change is the order of parameters to insert the matrix values. The indices now come before the values and size of the vector is not passed.

The `SetMat()` function also shows how the maps can be used to get the number of elements and go from local to global indices. The `Epetra_BlockMap GID()` function no longer has a shorthand, and instead uses `getGlobalElement()`. An example of this is lines 10-11, which use equivalent commands for Tpetra and the commented version of Epetra.

Example 8 SetMat.hpp: This method shows how to fill the matrix and a few map methods. Use the Teuchos::Array class instead of raw pointers and we no longer need to delete it.

```

1  void SetMat(){
    int* Indices = new int[NumEntriesPerRow];
3   double* Values = new double[NumEntriesPerRow];

5   Teuchos::Array<LO> cols(NumEntriesPerRow);
    Teuchos::Array<Scalar> vals(NumEntriesPerRow);
7   for(GO i = 0; i < tpetra_map->getNodeNumElements(); i++){
    //for(int i = 0; i < eptetra_map->NumMyElements(); i++){
9     cols.clear(); cols.resize(3); vals.clear(); vals.resize(3);
    if(tpetra_map->getGlobalElement(i) == 0){
11    //if(epetra_map->GID(i) == 0){
        Indices[0] = i; Indices[1] = i+1;
13        Values[0] = 4; Values[1] = -1;
        eptetra_matrix->InsertMyValues(i, NumEntriesPerRow-1, Values, Indices);

15        cols[0] = i; cols[1] = i+1;
17        vals[0] = 4; vals[1] = -1;
        tpetra_matrix->insertLocalValues(i, cols, vals);
19    } else if(tpetra_map->getGlobalElement(i) == NumGlobalElements){
        Indices[0] = i-1; Indices[1] = i;
21        Values[0] = -1; Values[1] = 4;
        eptetra_matrix->InsertMyValues(i, NumEntriesPerRow-1, Values, Indices);

23        cols[0] = i-1; cols[1] = i;
25        vals[0] = -1; vals[1] = 4;
        tpetra_matrix->insertLocalValues(i, cols, vals);
27    } else {
        Indices[0] = i-1; Indices[1] = i; Indices[2] = i+1;
29        Values[0] = -1; Values[1] = 4; Values[2] = -1;
        eptetra_matrix->InsertMyValues(i, NumEntriesPerRow, Values, Indices);

31        cols[0] = i-1; cols[1] = i; cols[2] = i+1;
33        vals[0] = -1; vals[1] = 4; vals[2] = -1;
        tpetra_matrix->insertLocalValues(i, cols, vals);
35    }
    }
37    eptetra_matrix->FillComplete(); tpetra_matrix->fillComplete();
    delete[] Indices; delete[] Values;
39 }

```

3.5. Import and Export. The Import and Export classes are used to move data between this processor and other processors. Its important function is to translate between the source and target maps. Importing and exporting can be done to any object that inherits from the DistObject base class. Example 9 shows the creation of Import and Export objects in both Epetra and Tpetra. It also includes a function that shows how the primary function is called from any object that inherits from DistObject.

We use the multivector to show that Import() in Epetra has become doImport() in Tpetra. As the code says, this call is useless because the source and target maps in the Import object are the same. The important thing to note is that the enumerated type for CombineMode has changed. Previously the options were: Add, Zero, Insert, InsertAdd, Average, and AbsMax. In Tpetra the only options are: ADD, INSERT, and REPLACE. Additionally, the namespace is required as seen in the example. When creating the Import or Export objects, the Tpetra

Example 9 ImportTest.hpp: Creating import and export objects is almost identical in Tpetra and Epetra.

```

1  #include "Epetra_Import.h"
   #include "Epetra_Export.h"
3  #include "Tpetra_Import.hpp"
   #include "Tpetra_Export.hpp"
5
   typedef Tpetra::Import<LO, GO> IMP;
7  typedef Tpetra::Export<LO, GO> EXP;

9  Epetra_Import* epetra_import;
   Epetra_Export* epetra_export;
11 RCP<IMP> tpetra_import;
   RCP<EXP> tpetra_export;
13
   void CreateImp(){
15     epetra_import = new Epetra_Import(*epetra_map , *epetra_map);
       epetra_export = new Epetra_Export(*epetra_map , *epetra_map);
17
       tpetra_import = rcp(new IMP(tpetra_map , tpetra_map));
19     tpetra_export = rcp(new EXP(tpetra_map , tpetra_map));
   }
21
   void UseImp(){
23     // This code doesn't do anything except show the function call
       epetra_multivector->Import(*epetra_vector , *epetra_import , Insert);
25
       tpetra_multivector->doImport
27     (*tpetra_vector , *tpetra_import , Tpetra::INSERT);
   }

```

version takes the parameters in the order: SourceMap, TargetMap, which is the opposite of the Epetra version.

4. Solving linear systems with Tpetra. We created the vectors and matrices so that we can solve linear systems. Stratimikos gives the ability to easily customize the solver and preconditioner in many ways. Stratimikos is a new package to unify the various solvers and preconditioners within the Trilinos framework. It does all the creation of the solver and preconditioner in the background based on the parameters specified in a simple XML file. This section describes the process of wrapping objects in Thyra and the steps to use the Stratimikos solver.

4.1. Wrap the Tpetra objects in Thyra. The Stratimikos solver described in Section 4.2 allows the user to easily create a Thyra solver with specific parameters chosen. In order to use this effectively, we need to wrap the Tpetra objects so Thyra can operate on them as if they were any Operator of VectorSpace objects. Thyra recommends using the create[Object] functions as seen in Example 10 to simplify the calls. The file Thyra_Tpetra_ThyraWrappers.hpp provides these functions to easily wrap the Tpetra objects. This code creates a Thyra::VectorSpace, Thyra::Vector, and Thyra::LinearOp to be used as map, vector, and operator respectively. Each of these is necessary to solve the linear system in Stratimikos. For more information about the Thyra package, see the guide about its operators and vectors [1].

Example 10 ThyraTest.hpp: Wrapping the Tpetra objects in Thyra allows the use of generic algorithms to use with the solver.

```

1 #include "Thyra_TpetraLinearOp.hpp"
2 #include "Thyra_TpetraVector.hpp"
  #include "Thyra_VectorSpaceBase.hpp"
4 #include "Thyra_TpetraThyraWrappers.hpp"

6 typedef Kokkos::DefaultNode::DefaultNodeType Node;
  typedef Thyra::VectorSpaceBase<Scalar> ThyraVS;
8 typedef Thyra::VectorBase<Scalar> ThyraVEC;
  typedef Thyra::LinearOpBase<Scalar> ThyraOP;
10
11 RCP<const ThyraVS> thyra_vs;
12 RCP<ThyraVEC> thyra_rhs;
  RCP<ThyraVEC> thyra_lhs;
14 RCP<ThyraOP> thyra_op;

16 void WrapThyra(){
  // Wrap the map as a vector space
18 thyra_vs = Thyra::createVectorSpace<Scalar, LO, GO, Node>
  (tpetra_map);
20 // Wrap the vectors for Thyra
  thyra_lhs = Thyra::createVector(tpetra_lhs, thyra_vs);
22 thyra_rhs = Thyra::createVector(tpetra_vector, thyra_vs);
  // Wrap the matrix as a linear operator
24 thyra_op = Thyra::createLinearOp<Scalar, LO, GO, Node>
  (tpetra_matrix, thyra_vs, thyra_vs);
26 }

```

4.2. The Stratimikos solver. The use of the AztecOO solver with the Epetra matrices and vectors will look familiar to experienced Epetra users. We show the most basic call to the AztecOO solver that creates it with default parameters and no preconditioner. With no options, we have a basic solver that will only work with simple linear systems. A real code would set options with `SetAztecOption()` to make the solver more elaborate and be able to solve any complicated system.

The best way to create a Stratimikos solver is using the default builder. This method takes the solver parameters as an XML file. The only parameter that should be passed is the name of the XML parameter file. We can pass the solver builder a number of different solvers and preconditioners, for example Belos, AztecOO, or Amesos and Ifpack for the preconditioner. All the options for the XML file and format can be found in the Trilinos documentation [3].

Example 11 lists the steps used to get the solver from Stratimikos and call the `solve()` method. Line 23 creates the Stratimikos default solver from the parameters chosen in the XML file. The next two lines are needed to force the solver builder to read the parameters that we passed to it. Then we get a `Thyra::LinearOpWithSolveFactory` from the solver builder. This is next given the `Thyra::LinearOp` that was wrapped in Example 10. Finally, a simple call to `solve()` passes the right and left hand side Thyra vectors. This will return a `SolveStatus` that can be used to check for convergence and has a print method that can be used to easily verify the result of the call to solve.

5. Conclusions. Tpetra is a powerful new package in Trilinos that gives users the ability to use multi-precision algorithms and different computing platforms. Tpetra makes extensive use of Teuchos utilities to help in memory management. RCP is used for garbage collection

Example 11 `SolveTest.hpp`: The parameters are passed to the solver with the XML file, then it is easy to call solve on our solver.

```

1 #include "Aztec00.h"
2 #include "Stratimikos_DefaultLinearSolverBuilder.hpp"
3 #include "Thyra_LinearOpWithSolveFactoryBase.hpp"
4 #include "Thyra_LinearOpWithSolveBase.hpp"
5 #include "Thyra_LinearOpWithSolveFactoryHelpers.hpp"
6
7 using Stratimikos::DefaultLinearSolverBuilder;
8 using Teuchos::VerboseObjectBase;
9
10 RCP<DefaultLinearSolverBuilder> tpetra_solver;
11 RCP<Thyra::LinearOpWithSolveFactoryBase<Scalar>> lows_factory;
12 RCP<Thyra::LinearOpWithSolveBase<Scalar>> lows;
13 Thyra::SolveStatus<Scalar> status;
14
15 AztecOO* epetra_solver;
16 int MaxIters = 100;
17 double Tolerance = 1e-6;
18
19 void Solve(){
20     epetra_solver = new AztecOO(epetra_matrix, epetra_lhs, epetra_vector);
21     epetra_solver->Iterate(MaxIters, Tolerance);
22
23     tpetra_solver = rcp(new DefaultLinearSolverBuilder("Parameters.xml"));
24     RCP<Teuchos::FancyOStream> out = VerboseObjectBase::getDefaultOStream();
25     tpetra_solver->readParameters(out.get());
26     lows_factory = tpetra_solver->createLinearSolveStrategy("");
27     lows = Thyra::linearOpWithSolve<Scalar>(*lows_factory, thyra_op);
28     status = lows->solve(Thyra::NOTRANS, *thyra_rhs, thyra_lhs.ptr());
29 }

```

and Array utilities wrap simple pointers to help avoid memory leaks. Tpetra uses the types `LocalOrdinal`, `GlobalOrdinal`, `size_t`, and `global_size_t` to distinguish between types of ordinals that can be passed. The basic types in Tpetra are all quite similar to those in Epetra. Using the `typedef` keyword, we can hide much of the templating and make the code easier to read.

Thyra and Stratimikos give users the ability to solve linear systems abstractly from any type of matrix and vectors. We wrap our Tpetra objects in Thyra and then can easily create the solver from an XML file. Using the solver is simple after it is created. Work will continue on Tpetra and Stratimikos to give users even more flexibility and options for matrix types and solver options.

REFERENCES

- [1] R. BARTLETT, *Thyra Linear Operators and Vectors*.
- [2] ———, *Teuchos::RefCountPtr beginner's guide: an introduction to the Trilinos smart reference-counted pointer class for (almost) automatic dynamic memory management in C++*, tech. rep., Sandia National Laboratories, 2004.
- [3] R. BARTLETT AND M. HEROUX, *Stratimikos::DefaultLinearSolverBuilder documentation*. http://trilinos.sandia.gov/packages/docs/dev/packages/stratimikos/doc/html/classStratimikos_1_1DefaultLinearSolverBuilder.html, 2009.
- [4] M. HEROUX, R. BARTLETT, V. H. R. HOEKSTRA, J. HU, T. KOLDA, R. LEHOUCQ, K. LONG, R. PAWLOWSKI, E. PHIPPS, A. SALINGER, H. THORNQUIST, R. TUMINARO, J. WILLENBRING, AND A. WILLIAMS, *An Overview of Trilinos*,

- Tech. Rep. SAND2003-2927, Sandia National Laboratories, 2003.
- [5] C. NVIDIA, *Programming Guide*, Version 0.8, 1, p. 53.
 - [6] D. PHAM, S. ASANO, M. BOLLIGER, M. DAY, H. HOFSTEE, C. JOHNS, J. KAHLE, A. KAMEYAMA, J. KEATY, Y. MASUBUCHI, ET AL., *The design and implementation of a first-generation CELL processor*, in 2005 IEEE International Solid-State Circuits Conference, 2005. Digest of Technical Papers. ISSCC, 2005, pp. 184–592.

TESTING ENGINEERING SOFTWARE: DEVELOPMENT AND TESTING OF THE CUBIT PROGRAM

STUDENT BRYAN J. HARDY* AND MENTOR BRETT W. CLARK†

Abstract. Testing is essential to the software development process. As active development is extended in a program and increasing amounts of code and algorithms combine, the probability that something may be missed also increases. Thus regular testing, both automated and human, must be done. This testing ensures that new changes in one area are accounted for and work properly, and don't negatively affect code in another area. The Python interface is a newer portion of Cubit that was found in need of testing. This work found a lot of capability that could be employed, and a lot of documentation adjustments needed to make that capability useful. This paper discusses the testing of both the outward user commands entered into Cubit and some of the internal data retrieval and modification functions accessed by Python. It looks at the effort to ensure Cubit functionality, make the Python functionality more apparent, and automate a regular testing of the Python interface. Increasing the scope of testing yields more complete code coverage and increased utility for the end user. Current and future user metrics are also discussed.

1. Introduction. Cubit is a finite element preprocessor for automated mesh generation. It has many features and algorithms for producing 2D and 3D meshes for both triangular/tetrahedral elements and quadrilateral/hexahedral elements. It "is designed to provide the user with a powerful toolkit of meshing algorithms that require varying degrees of input to produce a complete finite element model ... The overall goal of the CUBIT project is to reduce the time it takes a person to generate an analysis model." [2] Cubit is used to generate finite element meshes for geometric models that either came from a CAD modeler or which were generated within Cubit itself. Boundary conditions can also be defined and exported to fully prepare the model for analysis outside of Cubit.

Cubit is a command driven program. A GUI has also been implemented in the code to increase functionality, but most of the GUI functions are simply relaying commands to the command line. Many different commands and functions have been developed to accomplish the ends of Cubit, and as they are developed, tests are created and executed to ensure that the desired functionality works as intended. The current version of Cubit is 12.1. Due to the many years of development and continued work on the Cubit project, there is a lot of code to be tested and maintained to ensure robustness in performance.

Through this paper, we will describe the current testing setup and approaches. We will also delve more specifically into the testing of Cubit's Python interface. Future testing direction and methods will also be discussed.

2. General Cubit Testing. As Cubit has been developed, a substantial test suite has been created along with it. The tests consist of journal files (files of Cubit commands) that can be run in Cubit. Thus, by setting up the tests appropriately, certain parts of the code can be checked to ensure that they are working as desired. Cubit reports errors that can be used to determine passage or failure of a test. With this test suite in place, developers can access and run tests on their new code to see if it changes the outcome for other parts of the code. The entirety of the test suite (which consists of about 185 tests, each with several cases) is run each night, consistently checking the code as it is developed each day.

Many tests arise in association with new features and new code developed, but there are also many tests that are implemented in response to finding bugs in the program. A bug is found by manual testing, a fix is implemented, and then a test is created to make sure that the bug does not surface again.

*Brigham Young University, bhardy@byu.net

†Sandia National Laboratories, bwclark@sandia.gov

Bugs are found primarily by 3 means: by developers encountering them as they create new code, by intentionally testing the code and looking for bugs, or by users while running Cubit. User reports are extremely useful as they reflect problems encountered in real world application of Cubit. Unfortunately, a user finding the problem is the least desirable method, as that indicates the product was not fully prepared before it was released for use. Thus, efforts are made before hand to manually test Cubit and ensure stability in the code.

One testing method for finding bugs is a cold call type method. This entails a general use and abuse of Cubit in whatever way the user decides. This is not the most effective way to test, but it is a good idea to employ occasionally. It can cover a wide range of features and employ some of the lesser used (and thus less tested) features. However, this method can also lead to finding ridiculous bugs in the program, which are somewhat less useful. Abuse in testing is good to check robustness in the code, but there is a point at which it can become overkill and inefficient. Covering every possible case of how someone may use a feature inappropriately would be a long and not so fruitful process. Thus in testing efforts, one must separate the ridiculous from the useful, making a note of the ridiculous when found, but placing emphasis on finding the bugs in the more commonly trod paths. Bugs are not to be ignored and forgotten, no matter how ridiculous (as it may reveal some other more important underlying flaw), but priorities should be established to allocate time and resources to fix bugs.

Another method is a targeted type approach. This involves testing one particular area as thoroughly as possible, trying the full range of commands and cases, and the full range of inputs and values. The target can be selected either by user feedback (areas in which they have had problems) or by developer feedback (areas which they have tinkered in or developed recently). When the code to be tested is brand new code, there are potentially more unhandled cases and forgotten details that could be found and dealt with. When the code is more established, better possible functionality could be noticed and suggested as well as finding discrepancies in how the functions should work and what actually occurs.

One key element to testing is that when you encounter a bug (either yourself or as reported by someone else), you need to find a way to reproduce it simply and reliably. A list of one hundred commands that causes a crash or an error doesn't help a whole lot for the efforts of fixing it. Reducing the commands to the simplest possible set that will still produce the bug enables greater ease to find the problem within the code. Also, a report of a bug is not of much use when the bug can't be reproduced. Often times one may find a bug but not be able to remember how they got there. Thus we know that something is wrong, but have no way to fix it. Isolating the offending code enables fixes to be implemented in a more timely and efficient manner. Shaving off the time it takes to bullet proof the code leaves more time for further development and progression.

Since Cubit is driven by commands, the best way to reproduce bugs is by finding the smallest number of commands issued that show the result. Having a GUI included in the code changes things a little. Now, the program can be sending and receiving information without executing commands. In Cubit, there are panels in the GUI that provide a nice interface to create commands and then send them to the command line. There are other parts of the GUI that retrieve information or do other functions. A GUI is built to interact with a human, which increases the difficulty of automating testing. Thus, most GUI testing is manual. Cubit has many interactive GUI setups and display, including toolbars, a tree navigator, an entity property page, and the Graphics window. Thus with data being transferred and displayed in a variety of ways, there is always room for error that automated testing would not be able to distinguish. Testing the GUI could include haphazard button pushing (making sure the program can handle all the GUI events), but a more controlled look at each interface is

needed. This testing would need to be done with any GUI development, but perhaps not so often as the automated tests (again, efficiency).

3. Testing Cubit's Python Interface. The Python interface within Cubit is a unique portion of the program. Python is an object oriented scripting language. Cubit is a command driven program, but has a Python interface that can act as a driver for it. With Python enabled to interface with Cubit, we can now script various commands and use features such as loops, etc. to create more elegant and complicated command sets. It is also used to expose and employ methods from the underlying C++ code to the user. Its easy customization “provides a method of delivering code to specific users needing functionality without having to wait for an entire release cycle. It also provides a method of delivering code to users that is of interest only to that user and is not of benefit to the general user base.”[1] This makes Cubit much more versatile for many models, problems, and applications.

This interface was previously tested only to a degree. It is a relatively new component with extensive capabilities, so upon its implementation, the full range of rigorous testing was not completed. Its differences also prevent traditional tests from being added to the test suite readily (those tests consisting of Cubit commands, not Python code). A primary objective for this testing was to ensure the setup was made such that .py files (uncompiled Python files) could be used in an automated fashion and report back the result of the test.

The versatility of Python presents a challenge for testing and robustness. The more you can do, the more potential there is for problems to be unaccounted for. Thus some testing needs to be done to see if basic scripting ability is present (loops, defined methods, data structures, etc) and to what extent within the program. Cubit intends to have the full capability of a scripting language through Python. Methods can be defined that would help automate processes within Cubit. Combining with loops could automate even more, or such methods could be made available for the user to adjust and employ as needed. With the various capabilities referred to, “we have found the scripting interface to [be] very useful” [1] for providing the unique functionality it intends to expose.

Python scripting capability and interface must be tested within Cubit to ensure usability. The generic functionality testing has been done previously with minor nuances found. Python is fully functional within Cubit, and Cubit is fully functional as a module imported into Python. With Python appropriately set up to work with Cubit, the methods defined for the Cubit module also must have their functionality ensured. These methods are defined in C++ and work within the Cubit GUI, but have been exposed to the user and Python via SWIG. The majority of these methods are simply data retrieval methods. That is, they return information, but do not change the state of Cubit. Some methods actually do change the state of Cubit, either by changing settings, creating geometry, or passing in and executing Cubit commands.

The first approach needed was manual testing. We primarily needed to know what calls were available for use in Python, what they did (or were supposed to do), what parameters were required, and what would be returned. Documentation for this was generated from comments before each function, so we only had as much information on their workings as there are in the code comments. The comments were written by developers who knew the code, and consequently did not need as much explanation to figure out how to use the method. Users however, do not have such a knowledge base, and the initial state of the documentation for using Python in Cubit was sorely lacking in description and clarity.

Most of the functions are named specifically such that their purpose is fairly straightforward. Many have sufficient descriptions to specify further what it does. Many of the functions, however, did not explain what they were doing or what they were returning to the user, or did not say anything at all. All of the functions indicated the names of the parameters required and sometimes explained them, but none of them stated exactly what the expected

```

##This creates a vertex at your given point, and then
##returns the vertex existing in the model that is closest to it.
def closest_vertex(x,y,z):
    vertices = cubit.get_entities('vertex') ##a list of vertex ids in the model
    cubit.cmd('create vertex '+str(x)+' '+str(y)+' '+str(z))
    point_A = cubit.get_last_id('vertex')
    closest = vertices[0]
    for vert in vertices:
        if (cubit.get_distance_between(vert, point_A) < /
cubit.get_distance_between(closest, point_A)): closest = vert
    print closest ##could also use a return statement if that is more useful

##now that it's defined, it can be used at will
closest_vertex(1,2,3)
closest_vertex(89, -20, 43)

```

FIG. 3.1. An example Python script for Cubit

parameter was! Is the user supposed to enter an int? A string? A unique data structure? An enum? For many functions, it was easy enough to determine, but several were fairly ambiguous and non-conforming. This made the functions essentially useless except with a large amount of trial and error until you figured out what exactly the function was asking for and got your desired input correct for it. Hence although the Python interface was very useful, it was also not very usable. Clearing up how to use the various functions available improved its utility.

Testing began by going through all of the documented functions for Python in the Cubit module. It was soon discovered that several of the functions were not present in the module. Upon examining the code, it was found that these functions existed, but were not wrapped for use by Python. Due to how Python handles references and pointers, it made sense not to have these available for Python. Several more functions were found that likewise did not play well with Python, and so needed to be hidden from it.

A first pass of testing determined if the given functions were actually available for the user. Next, each function was examined to determine what parameters were needed and what information was returned to the user. This was used to clarify inputs and outputs in the documentation. Then, each function was examined to see whether or not it performed as advertised. In some cases, it performed correctly, but the documentation was inaccurate. In other cases, it did not do what it was intended to do. Often, the word and the deed matched appropriately. Thus in testing, care had to be taken to ensure accuracy, because a function that returned some value or object but didn't produce an error could still be wrong or broken.

Hence, initial functionality and existence was established for the Python interface. At that point, there was a base for which future testing and development could be established. Use cases for tests will be found by users who will now be more adept at utilizing these functions in their projects. Covering every use case for each function would be an incredible feat, and may even be redundant for some commands which have their inner workings tested by Cubit's test suite. Thus it is not pertinent to begin creating an extensive test suite for all capabilities. Rather the approach should be incremental as user problems are reported, and tests can then be developed to ensure the bugs are fixed and desired functionality in place.

After manual testing was completed, the comments in the code and help documentation were updated to reflect more accurately the capabilities that could be employed by Python. Now the work to be done was to create a way to automate some of this testing to ensure that the functionality would be maintained with the various changes and developments that occur. The traditional Cubit test suite uses Cubit errors and exits to determine whether a test

was successful. The Python tests were designed such that errors would be reported both with Python code errors and when the function is broken and returns the wrong value. This setup is indicative of the many ways in which bugs may be manifest, and accounts for it.

The Python tests could have a very broad scope, as indicated earlier. They were grouped in various ways, but generally with other functions that either provided similar information or dealt with similar objects in Cubit. For example, there is a test dealing with all the information retrieval commands related to non-exodus boundary conditions, and there is a test dealing with any commands that count the number of a particular element in the model. These tests are created only to indicate basic functionality of the methods and test for basic bugs. Much of the taxing of code robustness is covered by the general test suite. To reiterate the point, the full extent of what could be tested with the Python interface is extremely massive, so these tests are designed to ensure that the functionality continues to exist from Python and not to try the extreme limits of that functionality.

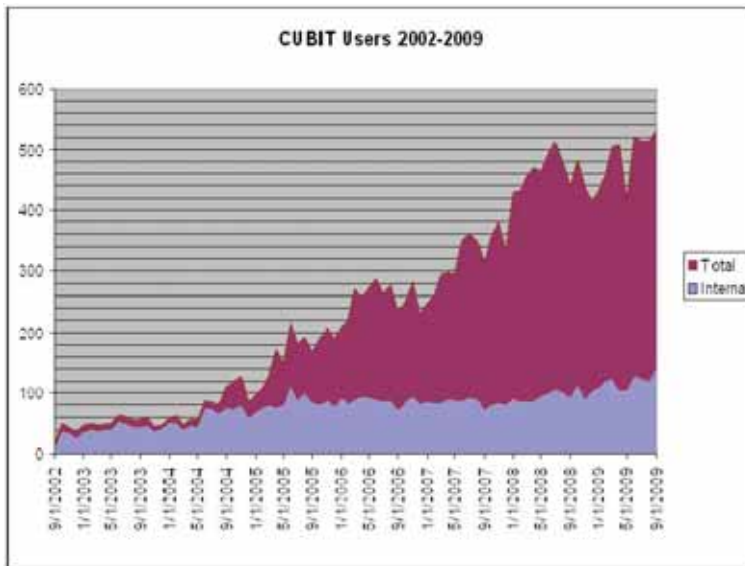
4. User Metrics. In order to make further and relevant improvements to the code, it is important to have information about how the program is being used and how it is not being used. User metrics will give developers a more keen insight into how to best meet the needs of users and what features or projects should have more time and money spent on them.

The first step was to record use and find out how many people regularly use Cubit, as seen in 4.1. This is a good general barometer for the current success of the program. The next indicator implemented was to record when Cubit abnormally exits operation. Many crashes in the program are difficult to reproduce because the chain of events that produced each one is not always recorded by commands issued. GUI usage is what throws things off, and the combination of commands and clicks is hard for the user to remember. Often, the user does something to get Cubit into an unstable state, and then some action will cause the crash. Developers are unable to reproduce, and thus unable to fix. Currently, the program reports when it crashed and an exit code, and the rest of the desired information is obtained from the user. This is general data to see what types of situations typically cause a crash, so we have a better heading on how to approach a solution. This effort will help improve robustness of the program, though may or may not improve functionality.

Moving forward, we are looking at what other metrics to gather and how to gather them. Cubit is a very large and complex program, and so there naturally are several ways to do this. One way to go about gathering the data is simply by the commands issued. Cubit is a command driven program, so this would report nearly everything a user does. That is also its drawback. It would return a large amount of information, and that return could slow down the speed of execution and would then require filtering to retrieve the useful, or most useful, data. Another problem that would be encountered here is possible spyware from retrieving proprietary information. Thus, the code would need to strip out all of the numbers and names from the parameters.

Another way to gather metrics is to use GUI events. This would limit our data sampling to just GUI version users, and then just the actual functions performed from the GUI. This would be valuable information for GUI development and usage, but would lack in many respects the breadth of coverage of functions (unless users do every single thing from the GUI) and depth in number of users (unless most users use the GUI version, which also would be valuable to know).

A third way still is to place several trigger spots in the code that would report when that portion of the code was used. These could correspond to common classes that are used by commands, or spots in the code that a set of commands pass through. A few different triggers for various operations could be targeted and implemented to focus and prioritize the metrics to analyze.

FIG. 4.1. *Cubit repeat users*

The first approach to gathering metrics would probably be the most useful in the end, as it would provide the most amount of information. It could also be used to check the command coverage of the test suite. However, the performance problem and implementation effort required suggests that this may not be the best approach for now. What seems to be the best approach for the time being is a combination of some of the GUI events from approach 2 and the triggers from approach 3. That way, we can tailor the code to report the most pertinent data for near term development and far term planning.

When we analyze these metrics, care must be taken. If we see a feature is not being used, it could be that it just isn't as useful a feature, and so less effort should be placed on it. But it could also be the case that the feature isn't used because it is not well publicized or understood, and so more effort should be placed on it. Obtaining these metrics will assist developers in knowing what questions to ask, and help them determine where to focus their efforts to help the users more and improve the code.

5. Conclusions. While development is ongoing, testing is ongoing. Whenever you are continually altering the code, there are chances for unexpected outcomes and mistakes, especially in very large and complex codes. Thus, testing must be maintained to ensure a robust and satisfactory program. Employing various techniques to test helps cover many aspects of the program. The work on testing the Python interface in Cubit exhibits this as a lot of functionality is present to be covered, it needs to be tested in different ways. The process was automated to ensure essential functionality without repeated manual testing. Obtaining user metrics will help direct where further development and testing efforts will be directed.

REFERENCES

- [1] K. MERKLEY, *The tool set for building claro - a component loading architecture*. Sandia Report, 2006.
- [2] R. MORRIS, *Cubit 12.1 User Documentation*, 2010.