

## SUMMER PROCEEDINGS 2016

The Center for Computing Research at Sandia National  
Laboratories

### Editors:

James B. Carleton and Michael L. Parks  
Sandia National Laboratories

December 15, 2016



**Sandia  
National  
Laboratories**

SAND#: SAND2017-1294 R

Sandia National Laboratories is a multi-mission laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's 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](mailto: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](mailto:orders@ntis.fedworld.gov)  
Online ordering: <http://www.ntis.gov/ordering.htm>



## Preface

The Center for Computing Research (CCR) at Sandia National Laboratories organizes a summer student program each summer, in coordination with the Computer Science Research Institute (CSRI) and Cyber Engineering Research Institute (CERI).

CERI focuses on open, exploratory research in cyber security in partnership with academia, industry, and government, and provides collaborators an accessible portal to Sandia's cybersecurity experts and facilities. Moreover, CERI provides an environment for visionary, threat-informed research on national cyber challenges.

CSRI brings university faculty and students to Sandia National Laboratories for focused collaborative research on DOE computer and computational science problems. CSRI provides a mechanism by which university researchers learn about problems in computer and computational science at DOE Laboratories. Participants conduct leading-edge research, interact with scientists and engineers at the laboratories, and help transfer the results of their research to programs at the labs.

A key component of CCR programs over the last decade has been an active and productive summer program in which students from around the country conduct internships at Sandia. 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 the 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.

Starting in 2014, CERI and CSRI combined their efforts to form the CCR Summer Proceedings. Both CERI and CSRI encourage all summer participants and their mentors to contribute a technical article to the CCR Summer Proceedings. In many cases, the CCR Proceedings are the first opportunity that students have to write a research article. Not only do these proceedings serve to document the research conducted during the summer but, as part of the research training goals of Sandia, 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 2016 CCR Proceedings, research articles have been organized into the following broad technical focus areas—*computational mathematics, applications, and software and high performance computing*—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 and CERI in 2016. The success of the program hinged on the hard work of enthusiastic student collaborators and their dedicated Sandia technical staff mentors. 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 reports.

An important educational component of the summer program is the CCR Summer Seminar Series. We would like to thank the staff who spoke at the 2016 Series: Jon Berry (org. 1464), Joe Bishop (1554), Frances Chance (1462), Pat Crossno (1461), Karen Devine (1426), Kurt Ferreira (1423), Michael Frank (1425), Todd Griffith (6121), Jim Laros (1422), Josh Robbins (1444), and Andy Salinger (1442).

Finally, the CCR summer program would not be possible without the administrative support of Ashley Avallone, Jonathan Compton, Steven Garcia, Denise LaPorte, Amy Levan, Lorena Martinez, Celia Montoya, Sandra Portlock, Phyllis Rutka, Val Romero, and Bernadette Watts.

James B. Carleton

Michael L. Parks

December 15, 2016



## Table of Contents

### Preface

<i>J.B. Carleton and M.L. Parks</i> . . . . .	iii
---	-----

### Computational Mathematics

<i>J.B. Carleton and M.L. Parks</i> . . . . .	1
Parallel Hierarchical Solver for Elliptic Partial Differential Equations <i>C. Chen, S. Rajamanickam, E. G. Boman, and E. Darve</i> . . . . .	3
Scalable Three-Term Recurrence Preconditioned Conjugate Gradient Methods <i>P.R. Eller and M. Hoemmen</i> . . . . .	17
Numerical Methods for Orthogonal Spline Collocation Discretizations in PDE Con- strained Optimization <i>N.L. Fisher and E.C. Cyr</i> . . . . .	27
Increasing Concurrency in Two-Level Schwarz Preconditioners via Additive Vari- ants <i>M. Lupo Pasini, R.S. Tuminaro, and J. Hu</i> . . . . .	40
Hybrid Multigrid Methods for Nearly Structured Meshes <i>P.B. Ohm and R.S. Tuminaro</i> . . . . .	53
A Hybridized Discontinuous Galerkin Method for Resistive Incompressible Mag- netohydrodynamics <i>S.J. Shannon, J.N. Shadid, T. Bui-Thanh, and J. Lee</i> . . . . .	63
An Importance Sampling Approach to Risk Estimation <i>T.A. Takhtaganov, D.P. Kouri, and D. Ridzal</i> . . . . .	77
Information Theoretic Refinement of Seismic Arrival Time with Uncertainty Esti- mation <i>C. Vollmer, M. Chen, and D. Stracuzzi</i> . . . . .	97
Optimal Experimental Design Using A Consistent Bayesian Approach <i>S.N. Walsh, T.M. Wildey, and J.D. Jakeman</i> . . . . .	109
<b>Applications</b> <i>J.B. Carleton and M.L. Parks</i> . . . . .	127
Implementation of Enthalpy Model for Polythermal Glaciers <i>A. Barone and M. Perego</i> . . . . .	129
Using and Developing Novel Data Analytics for Synthetic Microstructures Gener- ated by SPPARKS <i>A.R. Castillo, J.A. Mitchell, and S.D. Bond</i> . . . . .	145
Density Functional Theory Applied to Transition Metal Elements and Binaries: Development, Application, and Results of the V-DM/17 Test Set <i>E.R. Decolvenaere and A.E. Mattsson</i> . . . . .	163
An Exact Solution for a Forced Burger's Equation <i>K.L. Fisher and A.C. Robinson</i> . . . . .	176
A Comparative Study of Simplified Models for Ion Crowding and Correlation Ef- fects in Electrolyte Solutions <i>B.D. Hong and M. Perego</i> . . . . .	191
Analysis of Phase Precession from Dual Components in Hippocampal Place Cells <i>K.R. Kalmbach and F.S. Chance</i> . . . . .	201
Constraining Thermal Oscillations in Molecular Dynamic Simulations <i>R. Keller, S. Silling, and Q. Du</i> . . . . .	216
Interface Based Conductivity: An FEM Approximation of Electrical Conductivity in Multi-Material Cells <i>L.T. Meredith, C.M. Siefert, and R.M.J. Kramer</i> . . . . .	229

Semi-Lagrangian RKPM and its Implementation in the Sierra/Solidmechanics Analysis Code	
<i>M. Pasetto and D.J. Littlewood</i>	241
Exploring the Use of the FOVIO Eyetracker to Measure Cognitive Load	
<i>G.A. Thompson, L.E. Matzen, and M.L. Armenta</i>	254
File Fragment Classification through Sparse Coding	
<i>F. Wang, T. Quach, and F. Rothganger</i>	263
<b>Software and High Performance Computing</b>	
<i>J.B. Carleton and M.L. Parks</i>	272
Accelerating DGM with Graphics Processing Units and Kokkos	
<i>P.D. Bello-Maldonado, C. Ober, and K. Belcourt</i>	274
Kokkos-Based Structured Grid Multigrid Solver	
<i>Z.A. Bookey, J.K. Holmen, and J.J. Hu</i>	284
Distributed and Task-Parallel Approach to Linear Algebra Based Graph Analytics Using HPX	
<i>D.C. Bourgeois and M.M. Wolf</i>	297
Automatic Tuning of Performance Portable SPMV Kernels in Kokkos	
<i>S.P. Garcia De Gonzalo, S.D. Hammond, and C.R. Trott</i>	306
Transitioning Green-Gauss Gradients to the Kokkos Framework	
<i>W.B. Held and A.M. Bradley</i>	320
A Generalized Vectorization Mechanism for VTK-m	
<i>A. Hota and K. Moreland</i>	326
Evaluation of Area of Optimization in Hydrodynamic Code with PerfMiner	
<i>T. Juedeman and J. Cook</i>	338
Trilinos Improvements: Ifpack2 Block Relaxation Performance and YAML Parameter Lists	
<i>B.M. Kelley, M.F. Hoemmen, and C.M. Siefert</i>	346
A C++ Equation Parser for Next-Generation Platforms	
<i>J.A. Perez and D.J. Littlewood</i>	354
Kokkos Task-DAG Thread-Team Collectives on GPUs and CPUs	
<i>J.D. Stevens and H.C. Edwards</i>	363

## Computational Mathematics

Computational mathematics is concerned with the design, analysis, and implementation of algorithms to solve mathematical problems. Articles in this section describe the discretization of partial differential equations and methods to solve equations, couple multiphysics systems of equations, and quantify uncertainty.

*Chen, Rajamanickam, Boman, and Darve* introduce a parallel, hierarchical solver for linear systems generated from the discretization of elliptic PDEs that is based on the LoRaSp hierarchical solver, which uses block Cholesky factorization and low-rank elimination. The parallel version makes use of the distance-2 connection property to partition the graph nodes based on their distances from processor boundaries and coloring to coordinate computation and communication for processor boundary nodes. They demonstrate that their algorithm achieves nearly linear scaling in solve time and memory usage for 3-D problems and constant iteration count when used as a preconditioner for Poisson problems, and good strong and weak scaling on multiple processors.

*Eller and Hoemmen* derive scalable versions of the preconditioned conjugate gradient method that use a three-term recurrence. By overlapping two iterations of matrix-vector multiplies and preconditioner applications with a single non-blocking allreduce, they address scalability problems associated with blocking allreduce. They give full derivations of their scalable methods and compare them with two existing preconditioned conjugate gradient algorithms. Their approach provides the potential to develop further pipelined, non-blocking methods.

*Fisher and Cyr* construct a discretization for PDE constrained optimization problems based on piecewise Hermite cubic orthogonal spline collocation in one and two dimensions. They also prove the optimality of a preconditioner for the MINRES iteration and confirm it computationally. They apply the method to the problems of distributed control of Poisson's equation and Burgers' equation.

*Lupo Pasini, Tuminaro, and Hu* investigate an additive multigrid method adapted to a domain decomposition setting with the aim of achieving convergence properties comparable with multiplicative approaches while maintaining a computational cost comparable with additive approaches. The approach has the potential to enhance concurrency and improve performance on new and future architectures. They carefully construct the smoothed interpolation operator and use subdomain coloring to compress the prolongator to significantly decrease communication cost and memory storage requirements. They demonstrate that the approach can improve performance if the problem size and number of subdomains are carefully chosen to achieve good load balancing.

*Ohm and Tuminaro* present a multigrid method for linear systems derived from PDEs discretized using meshes that have both structured and unstructured regions that ensures that the prolongation operator is consistent at the interfaces between structured and unstructured regions. They demonstrate significant reductions in the number of solver iterations needed in cases where the mesh contains a region of high aspect ratio elements.

*Shannon, Shadid, Bui-Thanh, and Lee* construct a new high-order implicit hybridized discontinuous Galerkin method for a linearized, resistive, incompressible magnetohydrodynamics system. This method is well-suited to this multiphysics, highly nonlinear PDE system spanning a very large range of length and time scales. They prove the well-posedness and stability of the scheme, and show numerical results for a Hartmann flow, a steady-state problem, and a Rayleigh flow, a time-dependent problem. In both cases, they show optimal convergence to the analytic solution for the velocity and magnetic field.

*Takhtaganov, Kouri, and Ridzal* propose an algorithm for estimating finite coherent risk functions of random quantities of interest that require expensive computation, such as solving PDEs. They aim to reduce computation time by sampling the input parameter space more efficiently than in standard Monte Carlo sampling by using risk-informed sampling. They use both kernel density estimation and acceptance-rejection methods to generate samples, and use both likelihood ratio and distribution matching approaches to adjust the probabilities of generated samples. They demonstrate the proposed algorithm's effectiveness for problems with a small number of random inputs.

*Vollmer, Chen, and Stracuzzi* estimate signal arrival times for seismic sensors. Their method provides a more refined estimate than the STA/LTA algorithm, which only produces an onset search window, and it also gives an estimate of the uncertainty in the arrival time. They use a Gaussian noise model, an auto-regressive time series model, and a Monte Carlo sampling scheme, which are applied to produce a full probability density function for the arrival time.

*Walsh, Wildey, and Jakeman* use a recently-developed consistent Bayesian approach for solving stochastic inverse problems to guide the acquisition of experimental data and create an optimal experimental design. They consider the case where experimental data is not yet available, and the order in which the measurable quantities should be prioritized is determined by maximizing the expected information gain. They also discuss causes of and methods for avoiding infeasible data. They show numerical examples and compare their consistent Bayesian approach to existing classical/statistical Bayesian approaches.

J.B. Carleton

M.L. Parks

December 15, 2016

## PARALLEL HIERARCHICAL SOLVER FOR ELLIPTIC PARTIAL DIFFERENTIAL EQUATIONS

CHAO CHEN<sup>\*</sup>, SIVASANKARAN RAJAMANICKAM<sup>†</sup>, ERIK G. BOMAN<sup>‡</sup>, AND ERIC DARVE<sup>§</sup>

**Abstract.** Solving sparse linear systems from the discretization of elliptic partial differential equations (PDEs) is an important building block in many engineering applications. Sparse direct solvers can solve general linear systems, but are usually slower and use much more memory than effective iterative solvers. To overcome these two disadvantages, a hierarchical solver (LoRaSp) based on  $\mathcal{H}^2$ -matrices was introduced in [22]. Here, we have developed a parallel version of the algorithm in LoRaSp to solve large sparse matrices on distributed memory machines. On a single processor, the factorization time of our parallel solver scales almost linearly with the problem size for three-dimensional problems, as opposed to the quadratic scalability of many existing sparse direct solvers. Moreover, our solver leads to almost constant numbers of iterations, when used as a preconditioner for Poisson problems. On more than one processor, our algorithm has significant speedups compared to sequential runs. With this parallel algorithm, we are able to solve large problems much faster than many existing packages as demonstrated by the numerical experiments.

**1. Introduction.** Solving elliptic PDEs is an important building block in many engineering applications. The common approach is to first discretize a PDE with local bases such as finite difference or finite element methods, and then solve the subsequent sparse linear system. Our goal is to develop a parallel algorithm that is robust for solving large three-dimensional (3D) elliptic problems.

The earliest record for solving simultaneous linear equations is in an ancient mathematical text *The Nine Chapters on the Mathematical Art*, a collection of math problems composed in China more than 2000 years ago [27]. The solution in that book was what we call Gaussian elimination today. While Gaussian elimination delivers exact solutions (up to rounding errors) for general linear systems, its worst case time complexity is cubic, and it uses much more memory to solve a sparse matrix than storing it because of fill-in, nonzero entries introduced during the elimination process. To solve sparse linear systems more efficiently, several sparse direct solvers based on Gaussian elimination have been developed [5, 7, 9, 20]. They organize computation efficiently (e.g., the multifrontal algorithm [10]), and leverage different elimination orderings, such as the nested dissection ordering [11] and the approximate minimum degree ordering [2]. For a survey of sparse direct solvers, refer to [8]. However, those state-of-the-art sparse direct solvers still have quadratic-time complexity in the factorization phase for solving three-dimensional elliptic PDEs. This quadratic factorization cost is prohibitive for practical large problems.

A completely different approach for solving sparse matrices is to use iterative methods [24]. Unlike sparse direct solvers, iterative methods are more memory efficient because they treat a sparse matrix as a black-box operator and do not modify the matrix itself. Moreover, the multigrid method [15, 29], an effective iterative method, is among the fastest solution techniques for solving a class of elliptic PDEs. For example, the geometric multigrid method has linear time complexity for solving Poisson’s equation on regular domains. However, the convergence of iterative methods is not guaranteed for solving general elliptic PDEs, and the iteration number may grow dramatically for matrices with large condition numbers. Furthermore, iterative methods are typically not as efficient as sparse direct solvers for solving multiple right-hand sides.

To reduce the memory usage and improve the time complexity of sparse direct solvers,

---

<sup>\*</sup>Stanford University, cchen10@stanford.edu,

<sup>†</sup>Sandia National Laboratories, srajama@sandia.gov,

<sup>‡</sup>Sandia National Laboratories, egboman@sandia.gov

<sup>§</sup>Stanford University, darve@stanford.edu

several hierarchical solvers [1, 3, 17, 28] based on the  $\mathcal{H}$ -matrix theory [14] have been developed, and some parallel algorithms have been introduced [12, 21, 23]. The key idea is data sparsity, or approximation of dense matrix blocks by structured matrices. We investigate parallelization of a closely-related hierarchical linear solver named LoRaSp [22]. LoRaSp is based on  $\mathcal{H}^2$ -matrices, not  $\mathcal{H}$ -matrices, and employs graph partitioning as opposed to the nested dissection ordering of traditional hierarchical solvers. Similar to traditional hierarchical solvers, LoRaSp exhibits almost linear time complexity and is more memory efficient than sparse direct solvers. We propose a new parallelization approach to LoRaSp that is based on data-decomposition, and as we will show that an important sparsity property can be leveraged with the parallel algorithm.

There are three main features of our parallel algorithm as follows. First, only local communication is needed. In other words, every processor only exchanges information with its neighbors. Second, the communication volume is small compared to the amount of computation. It will be shown later that the communication-to-computation ratio is roughly the surface-to-volume ratio of a cube. Third, the bulk of work in our parallel algorithm lies in dense linear algebra operations. Compared to iterative methods, our algorithm is more friendly to modern machine architectures and provides an opportunity for using modern many core processors such as the graphics processing unit (GPU) and Intel Xeon Phi [18].

The rest of this paper is organized as follows. Section 2 reviews the sequential algorithm in [22] as the foundation for the parallel algorithm. Section 3 discusses the details of our parallel algorithm. Section 4 shows numerical results and Section 5 summarizes this report.

**2. Sequential hierarchical solver.** We assume the sparse linear system to be solved is symmetric positive definite (SPD) in this paper. An SPD matrix can be represented by a graph, where each vertex represents a row/column and edges between vertices represent matrix nonzeros. LoRaSp is based on block Cholesky factorization of SPD matrices, so we present block Cholesky factorization to introduce notations and set the stage for LoRaSp.

**2.1. Block Cholesky factorization.** For an SPD linear system  $A$ , define  $V$  as the set of all unknowns. Given a clustering of the unknowns  $V = \cup_{i=1}^n \pi_i$ , block Cholesky factorization is equivalent to applying a sequence of linear operators  $\mathcal{G}_i$ , ( $i = 1, 2, \dots, n$ ) on  $A$ , where  $\mathcal{G}_i$  eliminates cluster  $\pi_i$ . A good clustering of the unknowns can be obtained via partitioning the corresponding graph of  $A$ , which can be done using any of the existing graph partitioning packages - METIS [19], Scotch [6] and Zoltan [4] to name a few. A good clustering has well-balanced numbers of unknowns in all clusters, and the number of edges between different clusters is small. For example, Figure 2.1(Left)<sup>1</sup> shows a 32-parts clustering of the vertices in a two-dimensional unstructured grid using Scotch, where every cluster is color-coded.

The nonzero pattern of an SPD sparse matrix during Cholesky factorization can be perfectly depicted by the graph notation [8], and the basic rule is that after an unknown is eliminated, its neighbors form a clique, representing fill-in in the original sparse matrix. Similarly, the nonzero pattern of a blocked sparse matrix<sup>2</sup> during block Cholesky factorization can be described using the same rule but on the *quotient graph*. With a sparse matrix partitioning, each node in the quotient graph stands for a cluster of vertices in the original graph and an edge exists between two nodes in the quotient graph if two clusters are adjacent in the partitioned graph. From now on, we always talk about the quotient graph corresponding to a matrix partitioning. From the perspective of the quotient graph, block Cholesky factorization traverses the quotient graph, and after a node is visited, its

<sup>1</sup>Figure taken from [22] with permission.

<sup>2</sup>A block is a nonzero block if it has at least one nonzero entry.

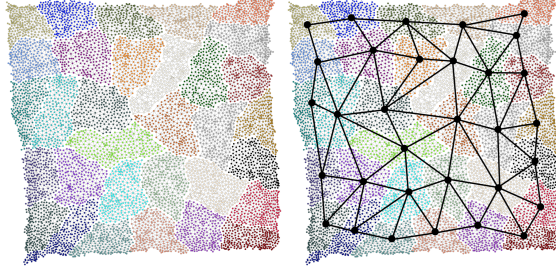


Fig. 2.1: (Left) A clustering of the vertices in a two-dimensional unstructured grid. All vertices have been partitioned into 32 clusters using Scotch, where every part is color-coded. The edges are not shown here for visualization purpose. (Right) The quotient graph corresponding to the partitioning of all vertices. Every node in the quotient graph stands for a cluster of vertices, and edges in the quotient graph represents the adjacency of two clusters. (Figure taken from [30] with permission.)

neighbors form a clique.

**2.2. Low-rank elimination.** As a preliminary to understand LoRaSp, we first introduce *low-rank elimination* (LRE). Suppose in a block SPD linear system,  $A_{sn}$  is full-rank and  $A_{sw} = U V^T$  is low-rank

$$A x = \begin{pmatrix} A_{ss} & A_{sn} & A_{sw} \\ A_{sn}^T & A_{nn} & A_{nw} \\ A_{sw}^T & A_{nw}^T & A_{ww} \end{pmatrix} \begin{pmatrix} x_s \\ x_n \\ x_w \end{pmatrix} = \begin{pmatrix} A_{ss} & A_{sn} & U V^T \\ A_{sn}^T & A_{nn} & A_{nw} \\ V U^T & A_{nw}^T & A_{ww} \end{pmatrix} \begin{pmatrix} x_s \\ x_n \\ x_w \end{pmatrix} = \begin{pmatrix} f_s \\ f_n \\ f_w \end{pmatrix}.$$

With auxiliary variables  $r_s = U^T x_s, b_s = V^T x_w$ , we define the extended sparsification operator  $\mathcal{E}_{sw}$  that maps  $A$  to a larger sparse matrix, and the original linear system becomes

$$\mathcal{E}_{sw}(A) \tilde{x} = \begin{pmatrix} A_{ss} & U & A_{sn} & & \\ U^T & & & & -I \\ A_{sn}^T & & A_{nn} & A_{nw} & \\ & & A_{nw}^T & A_{ww} & V \\ & -I & & V^T & \end{pmatrix} \begin{pmatrix} x_s \\ b_s \\ x_n \\ x_w \\ r_s \end{pmatrix} = \begin{pmatrix} f_s \\ 0 \\ f_n \\ f_w \\ 0 \end{pmatrix},$$

where  $I$  is the identity matrix of appropriate size.

Instead of eliminating  $x_s$  from  $A$  directly, we can eliminate  $x_s$  and  $b_s$  from  $\mathcal{E}_{sw}(A)$  and obtain

$$\mathcal{G}_{x_s} \mathcal{E}_{sw}(A) = \begin{pmatrix} I & & & & \\ & U^T A_{ss}^{-1} U & U^T A_{ss}^{-1} A_{sn} & & -I \\ & U A_{ss}^{-1} A_{sn}^T & A_{nn} & A_{nw} & \\ & & A_{nw}^T & A_{ww} & V \\ & -I & & V^T & \end{pmatrix},$$

$$\mathcal{W}_{sw}(A) \equiv \mathcal{G}_{b_s} \mathcal{G}_{x_s} \mathcal{E}_{sw}(A) = \begin{pmatrix} I & & & \\ & I & & \\ & & S_{nn} & A_{nw} & S_{nr_s} \\ & & A_{nw}^T & A_{ww} & V \\ & & S_{nr_s}^T & V^T & S_{r_s r_s} \end{pmatrix},$$

where  $I$ 's are identity matrices of appropriate sizes,  $S_{r_s r_s} = -(U^T A_{ss}^{-1} U)^{-1}$ ,  $S_{nr_s} = U A_{ss}^{-1} A_{sn}^T \cdot (U^T A_{ss}^{-1} U)^{-1}$  and  $S_{nn} = A_{nn} - U A_{ss}^{-1} A_{sn}^T \cdot (U^T A_{ss}^{-1} U)^{-1} \cdot U^T A_{ss}^{-1} A_{sn}$  are the Schur complements. Suppose  $x_s, x_w$  have sizes  $N_s, N_w$  and  $b_s, r_s$  have sizes  $r \ll N_s, N_w$ . Dropping lower-order term involving  $r$ , the saving is  $O(N_s \cdot N_w^2) - O(N_w \cdot N_s^2)$ , where the first term corresponds to forming the Schur complement  $S_{ww} = A_{ww} - A_{sw}^T A_{ss}^{-1} A_{sw}$  in  $A$  and second term corresponds to computing the low-rank factorization  $A_{sw} = U V^T$  with the singular value decomposition (SVD). When  $N_w \gg N_s$ , the saving will be significant.

With the graph notation, low-rank elimination is shown in Figure 2.2, where Figure 2.2(a) shows the graph of  $A$ ; Figure 2.2(b) shows the graph of  $\mathcal{E}_s(A)$ ; and Figure 2.2(c) shows the graph of  $\mathcal{W}_{sw}(A)$ .

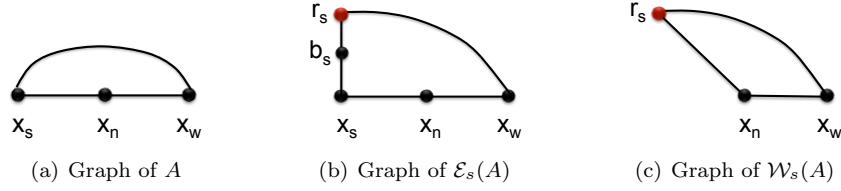


Fig. 2.2: Graph notation for low-rank elimination. (a) The original matrix  $A$  is a  $3 \times 3$  block SPD matrix, so its graph has three clusters where all nodes are connected. The edge between  $X_s$  and  $X_w$  corresponds to a low-rank matrix block. (b) The extended sparsification operator  $\mathcal{E}_s$  is applied on  $A$ , which introduces a black node  $b_s$  and a red node  $r_s$  into the graph, and the connection between  $X_s$  and  $X_w$  now go through  $b_s$  and  $r_s$ . (c) Node  $X_s$  and node  $b_s$  have been eliminated, introducing a new edge between  $r_s$  and  $X_n$ .

In general, a block SPD matrix may have multiple low-rank sub-blocks and multiple full-rank sub-blocks corresponding to a cluster of vertices. What we do then is compress all low-rank sub-blocks together, forming the extended matrix and eliminating corresponding nodes in the graph. This sub-routine is used in LoRaSp, and it is summarized in Algorithm 2.1.

**2.3. Hierarchical solver algorithm.** Block Cholesky factorization is usually not efficient for sparse SPD matrices because of potentially too much fill-in. An important observation in [22] is that some fill-in is data sparse, i.e., some matrix off-diagonal blocks are low-rank. These low-rank blocks are named well-separated interactions as defined in [22]. To control the amount of fill-in, LoRaSp uses LRE to maintain sparsity. To be more specific, before applying the operator  $\mathcal{G}_s$  to eliminate a cluster  $\pi_s$ , LoRaSp classifies all nonzero blocks into near/full-rank interactions and well-separated/low-rank interactions, and then uses LRE. From the perspective of the quotient graph, LRE redirects well-separated edges through the parent red node. With a parent red node, much less fill-in will be introduced after cluster  $\pi_s$  is eliminated.

After all nodes in the quotient graph are eliminated in LoRaSp, what remains is a graph consisting of all parent red nodes, where every node represents a much smaller cluster than in the original quotient graph. Then LoRaSp uses a coarsening step to merge all pairs of red nodes together and repeats the same algorithm on the coarse graph. This recursive algorithm is summarized in Algorithm 2.2. For a detailed explanation of the algorithm and analysis of the hierarchical solver, we refer to [22].



**Algorithm 2.1** Low-rank elimination sub-routine

---

**function** LRE(block SPD matrix  $A$ , vertex clustering  $\Pi = \cup_{i=1}^n \pi_i$ , index  $s$ )  
 find all low-rank matrix blocks  $A_{\pi_s, \pi_{t1}}, A_{\pi_s, \pi_{t2}}, \dots, A_{\pi_s, \pi_{tn}}$   
 compute the low-rank approximation of the concatenated matrix

$$U V^T = [A_{\pi_s, \pi_{t1}}, A_{\pi_s, \pi_{t2}}, \dots, A_{\pi_s, \pi_{tn}}]$$

let  $w = \cup_{i=1}^n t_i$   
**return**  $\mathcal{W}_{s,w}(A)$

**function** LRE(block SPD matrix  $A$ , vertex clustering  $\Pi = \cup_{i=1}^n \pi_i$ )  
**for**  $i = 1, 2, \dots, n$  **do**  
 $A \leftarrow \text{LRE}(A, \Pi, i)$   
**return**  $A$

**function** LRE(block SPD matrix  $A$ , vertex clustering  $\Pi = \cup_{i=1}^n \pi_i$ , vertex clustering subset  $\cup_{i=1}^m \pi_{si}$ )  
**for**  $i = 1, 2, \dots, m$  **do**  
 $A \leftarrow \text{LRE}(A, \Pi, si)$   
**return**  $A$

---

**Algorithm 2.2** Hierarchical solver algorithm

- 
- 1: Partition the graph of  $A$  and get vertex clusters  $\Pi = \cup_{i=1}^n \pi_i$
  - 2: LRE( $A, \Pi$ )
  - 3: create the coarse graph  $A_c$  from  $A$  and recurse on  $A_c$
- 

**3. Parallel hierarchical solver.** As we see in Section 2, LoRaSp repeatedly carries out two operations  $\mathcal{G}$  and  $\mathcal{E}$  on a sparse matrix. With the graph notation, the first operation  $\mathcal{G}$  introduces new edges, and the other operation  $\mathcal{E}$  creates a parent red node and redirects all well-separated edges through that node. With these two operators, LoRaSp has an important sparsity structure that we call *distance-2 connection* property. It means that in the graph, no newly introduced edge will exist between two nodes farther than distance two from each other, where distance is measured as the length of the shortest path in the original quotient graph. This property is the key to our parallel algorithm, and we give a sketch of the proof in the appendix.

**3.1. Parallel algorithm.** To present the parallel algorithm, we first introduce some definitions and notations. We focus on the quotient graph  $G = (V, E)$  corresponding to a given vertex partitioning  $V = \cup_{i=1}^n \pi_i$  of the SPD matrix  $A$ . The distance between two nodes in the quotient graph is defined as the length of the shortest path between them. We define the *distance-2 neighbors*  $\mathcal{N}_2(\cdot)$  of a node  $S$  as all neighbors of  $S$  and the neighbors of its neighbors, i.e.,

$$\mathcal{N}_2(S) = \mathcal{N}(S) \cup \mathcal{N}(\mathcal{N}(S)),$$

where  $\mathcal{N}(\cdot)$  is the set of all neighbors. For our parallel algorithm, we always assume a processor  $P$  owns a piece of the whole quotient graph  $G_P$  such that  $G = \cup_P G_P$  and the sub-graph owned by different processors do not overlap. We define the distance between a node  $S$  in the quotient graph and a processor  $P$  as the minimum distance between  $S$  and

any node in  $G_P$ . We also define the *distance-2 neighbors*  $\mathcal{N}_2(\cdot)$  of a processor  $P$  as processors (excluding  $P$  itself) owning at least one distance-2 neighbor of any node in  $G_P$ , i.e.,

$$\mathcal{N}_2(P) = \{Q \mid \exists S \in G_P : G_Q \cap \mathcal{N}_2(S) \neq \emptyset\} / \{P\}.$$

For every processor, we define *d1 nodes* as the boundary nodes, i.e., they are distance one from at least one other processor. With the same logic, we define *d2 nodes* as those that are distance two from at least one other processor. The rest of nodes on every processor are defined as *d3 nodes*, whose distance to any other processor is greater or equal to three. A two-processor example showing d1 nodes, d2 nodes and d3 nodes is given in Figure 3.1.

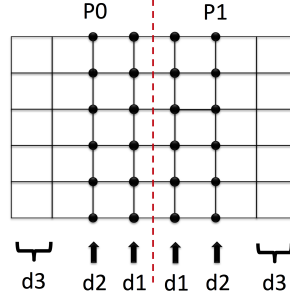


Fig. 3.1: A two-processor example showing d1 nodes, d2 nodes and d3 nodes on each processor. The structured grid represents the quotient graph, which is evenly divided into two pieces owned by two processors  $P_0$  and  $P_1$  respectively. Boundary nodes on each processor are defined as d1 nodes, and d2 nodes are one layer away from the other processor. The rest of the interior nodes on every processor are defined as d3 nodes, which is not shown explicitly.

With the definition of d1 nodes, d2 nodes and d3 nodes, we point out three important observations as follows. First, d1 nodes are coupled between different processors obviously. So d1 nodes should be processed in some order to avoid conflicts between different processors, and communication is needed for updating edges across different processors. Second, d2 nodes are partially coupled between different processors. A d2 node on one processor may be connected to a d1 node on another processor because of fill-in. But a d2 node on one processor will never connect to a d2 node on a different processor thanks to the distance-2 connection property of LoRaSp. So d2 nodes can be processed in parallel, and communication is needed for updating across edges. Third, d3 nodes are completely decoupled between different processors because of the distance-2 connection property, and hence they can be processed by all processors in parallel. With these observations, we have the parallel algorithm shown in Algorithm 3.1<sup>3</sup>. The algorithm for d1 nodes is shown in Algorithm 3.2.

Before diving into the details of a good ordering for d1 elimination, two advantages of our parallel algorithm are already clear from Algorithm 3.1. One is that every processor exchanges information only with its distance-2 neighbors. So communication in our parallel algorithm is always local. The other advantage is that the amount of communication is small compared to the amount of computation. The reason is that the communication volume is roughly proportional to the number of d1 nodes, while the amount of computation is proportional to the total number of d1 nodes, d2 nodes and d3 nodes. For 3D problems,

<sup>3</sup>all parallel pseudocode is written in *single program multiple data (SPMD)* style.

**Algorithm 3.1** Parallel algorithm

---

**Require:** my processor id  $P$ , local sub-graph  $G_P$  (local vertex clusters  $\Pi_P$ ), and local matrix  $A_P$  that has a subset of rows and columns corresponding to  $\Pi_P$  in the original matrix  $A$ .

- 1: Classify all clusters  $\Pi_P$  into d1 clusters, d2 clusters and d3 clusters.
  - 2: process d1 clusters with Algorithm 3.2
  - 3: LRE( $A_P$ ,  $\Pi_P$ , d2 clusters)
  - 4: communicate with  $\mathcal{N}_2(P)$  processors and update blocks in  $A_P$  corresponding to across edges in  $G_P$
  - 5: LRE( $A_P$ ,  $\Pi_P$ , d3 clusters)
  - 6: create the coarse graph  $A_{P_c}$  from  $A_P$  and recurse on  $A_{P_c}$
- 

this communication-to-computation ratio is approximately the surface-to-volume ratio of a cube, i.e., the amount of communication is one order less than computation in our parallel algorithm.

Another important observation useful in practice is that the communication for d1 nodes and the computation for d3 nodes are independent. Therefore, communication for d1 nodes can be overlapped with the computation for d3 nodes. Specifically, to hide the latency of sending messages for d1 nodes, every processor can work on its d3 nodes in the mean time while waiting for messages from other processors.

**3.2. Coloring of d1 nodes.** To coordinate all processors in eliminating d1 nodes in parallel, we assign colors to all d1 nodes so that all processors can process d1 nodes with the same color at a time. To avoid any conflict, every pair of d1 nodes within distance two must have different colors if they belong to different processors, i.e.,

$$\begin{aligned} \forall S, T \in V \text{ s.t. } T \in \mathcal{N}_2(S) \text{ and } S, T \text{ belong to different processors} \\ \text{Color}(S) \neq \text{Color}(T) \end{aligned} \quad (3.1)$$

Given such a coloring result, we have the algorithm for processing d1 nodes as shown in Algorithm 3.2.

**Algorithm 3.2** Process d1 nodes

---

**Require:** Coloring of all d1 nodes such that Condition 3.1 is satisfied, and inputs from Algorithm 3.1

- 1: **for all** colors  $i$  **do**
  - 2:   LRE( $A_P$ ,  $\Pi_P$ , d1 clusters with color  $i$ )
  - 3:   communicate with  $\mathcal{N}_2(P)$  processors and update blocks in  $A_P$  corresponding to across edges in  $G_P$
- 

Next we discuss two different coloring schemes. The first one, which we call processor coloring, is to assign colors to all processors such that every pair of distance-2 neighbors have different colors. Another perspective to view processor coloring is that it assigns one color to all of a processor's d1 nodes that is not the same color used for the d1 nodes of any distance-2 neighbor. This may lead to load imbalance because at every iteration in Algorithm 3.2 only the processors with a specific color are busy and others are idle. For a structured two-dimensional (2D) grid, the number of colors needed is  $3^2 = 9$ . For a structured three-dimensional (3D) grid, the number of colors needed is  $3^3 = 27$ . Therefore,

at every iteration, only a portion of  $1/9$  and  $1/27$  of all processors are working for in 2D and 3D respectively.

The other coloring scheme that extracts more parallelism is to color all d1 nodes directly. We call this scheme node coloring. Ideally, the node coloring scheme leads to a well-balanced number of d1 nodes with every color on all processors. If so, the load is perfectly balanced on all processors. For a structured 2D grid, the number of colors needed is 4 when four corners of a square have different colors. For a structured 3D grid, the number of colors needed is 8 when eight corners of a cube have different colors. Therefore, the node coloring scheme can lead to a smaller number of colors and much better load-balancing.

To compare processor coloring and node coloring, we give a simple example here. In Figure 3.2, the graph is divided into four pieces and owned by four processors. With processor coloring, all processors have different colors, and all d1 nodes on one processor have the same color as shown in Figure 3.2(a). Therefore, at every iteration in Algorithm 3.2, only one processor is working and the other three are idle. In contrast, every processor has all four different colors with node coloring as shown in Figure 3.2(b), so all processors have work to do at every iteration.

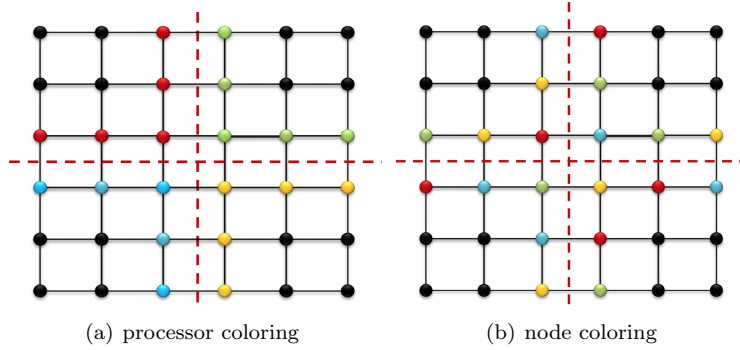


Fig. 3.2: Different coloring schemes for a four-processors example: processor coloring (left) assigns the same color to all d1 nodes on a processor, while node coloring (right) assigns the same color to d1 nodes on all four processors. As a result, only one processor is working at every iteration with the processor coloring scheme, whereas four processors are working at every iteration with the node coloring scheme.

**4. Numerical results.** We first present sequential results on linear factorization time and almost constant iteration numbers. Test matrices are generated from Poisson problems on a cube  $\Omega = (0, 1)^3$  with random Dirichlet boundary conditions, and the discretization scheme uses a second order finite difference stencil. All experiments are done on the Vesper machine (vesper@sandia.gov) at Sandia, which has 32 cores and about 132 gigabytes (GB) of memory.

We call our parallel solver ParH2 (Parallel solver using  $\mathcal{H}^2$ -matrices), which is implemented with the *message passing interface* (MPI) [13]. For sequential results, we run ParH2 with one MPI rank.

**4.1. Sequential results.** In the first experiment, we want compare the performance of ParH2 with LoRaSp by fixing the low-rank truncation error  $\epsilon$  and only keep the singular

values satisfying the following criteria for low-rank approximations

$$\sigma_i/\sigma_0 \geq \epsilon,$$

where  $\sigma_0$  is the largest singular value. The factorization times for LoRaSp and ParH2 are plotted in Figure 4.1. As shown in the figure, timing results for LoRaSp and ParH2 with the same truncation error are very close to each other, as are the other results such as memory footprint and iteration numbers. Therefore, we only present results from ParH2 hereafter.

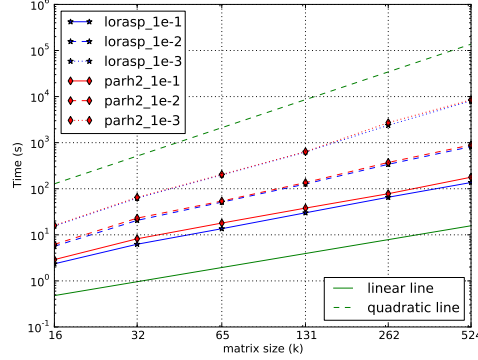


Fig. 4.1: Factorization time of ParH2 and LoRaSp for different truncation errors:  $\epsilon = 10^{-1}, 10^{-2}, 10^{-3}$ . Timing results for LoRaSp and ParH2 almost overlap with each other. The factorization time with fixed  $\epsilon$  grows towards quadratic when  $\epsilon$  decreases.

Figure 4.1 also shows that the factorization time is nearly linear for  $\epsilon = 10^{-1}$ , but it scales closer and closer to quadratic when  $\epsilon = 10^{-2}$  and  $\epsilon = 10^{-3}$ . When  $\epsilon$  is fixed, the rank of well-separated interactions grows at coarser and coarser graphs, so the factorization time may grow faster than linearly when  $\epsilon$  is small. This is shown in [22].

Because of using low-rank approximations in the algorithm, a single solve with ParH2 is not as accurate as standard sparse direct solvers, so we use it as a preconditioner for the conjugate gradient method (CG) [16] and show the iteration numbers in Table 4.1.

Table 4.1: Truncation error/ $\epsilon$ , matrix size and iteration number ( $\sim$ out of memory)

$\epsilon \setminus$ matrix	16k	32k	65k	131k	262k	524k	1m	2m
$10^{-1}$	8	9	11	14	16	20	23	28
$10^{-2}$	3	3	3	4	4	4	4	5
$10^{-3}$	1	2	2	2	2	2	$\sim$	$\sim$

Although setting  $\epsilon = 10^{-1}$  leads to linear factorization time, the corresponding iteration number grows as shown in Table 4.1. For  $\epsilon = 10^{-2}$  and  $\epsilon = 10^{-3}$ , the iteration number is almost constant but the factorization time grows faster than linearly. Therefore, fixing  $\epsilon$  is not a good criteria for ParH2.

To have almost constant number of iterations,  $\epsilon = 10^{-1}$  is apparently not accurate enough, so we focus on adding an extra constraint along with  $\epsilon = 10^{-3}$  to get nearly linear time factorization and almost constant iteration numbers. The factorization time for fixing

$\epsilon = 10^{-3}$  does not scale linearly because of increasing low-rank approximation ranks at coarser graphs, so in the next experiment, we force an upper bound  $\mathcal{B}$  on the rank and only keep at most  $\mathcal{B}$  singular values in all low-rank approximations. This will definitely force the factorization time to scale linearly as proved in [22].

As shown in Figure 4.2, the factorization time is linear for different rank upper bounds:  $\mathcal{B} = 8, 16, 32$ . When the rank is bounded, the approximation at coarser graphs will be less and less accurate, which in turn leads to growing number of iterations, as shown in the first part of Table 4.2. To preserve linear scalability of factorization time and almost constant iteration number, we allow the upper bound to grow by a small factor of 1.1 at every level (grow by 10% at every level). Figure 4.2 shows that the factorization time increases by roughly a constant factor, and the second part of Table 4.2 shows that the iteration number is almost constant for  $\mathcal{B} = 16, 32$ .

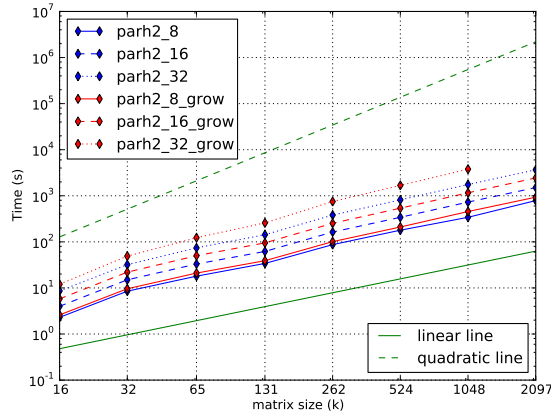


Fig. 4.2: (Red) linear factorization time for different bounds on the rank, and (Blue) linear factorization time for growing bounds by a factor of 1.1 at coarser graphs. Three different upper bounds are shown here:  $\mathcal{B} = 8, 16, 32$ .

Table 4.2: Rank, matrix size and iteration number. The first half corresponds to fixed upper bounds of rank and the second half corresponds to growing upper bounds at 10%. (\*growing upper bound; ~out of memory)

rank \ matrix	16k	32k	65k	131k	262k	524k	1m	2m
8	15	13	15	22	21	26	32	38
16	10	8	9	15	12	15	17	21
32	5	4	4	9	6	8	9	10
8*	13	10	12	18	15	19	22	26
16*	7	5	5	9	6	8	8	10
32*	2	2	3	4	4	4	4	~

**4.2. Parallel results.** Here we present numerical results by running ParH2 on more than one processor with  $\epsilon = 10^{-3}$  and  $\mathcal{B} = 16$  or 32. All parallel runs have almost the same

( $\pm 1$ ) iteration numbers as corresponding sequential runs, but the factorization time, solve time and memory footprint have been distributed to different processes. We focus on the scalability of the factorization time in the following discussion.

First we compare ParH2 to other existing state-of-the-art sparse direct solvers, including SuperLU [9] and PARDISO-MKL [20, 25, 26]. Figure 4.3 shows timing results of SuperLU, PARDISO-MKL and the parallel factorization time of ParH2 ( $\mathcal{B} = 16$ ). In the figure, SuperLU and PARDISO-MKL clearly exhibit quadratic time complexity, whereas ParH2 has linear time complexity. The line of ParH2 with 1 MPI rank crosses the line of PARDISO-MKL at matrix size 262k, whereas the line of ParH2 with 4 MPI rank crosses the line of PARDISO-MKL at matrix size 65k. Some lines do not extend to all matrix sizes either because the running time is too long (SuperLU) or because memory requirements exhausted the available resources (ParH2 with 32 MPI ranks). Details of ParH2 parallel efficiency are studied in the a later experiment.

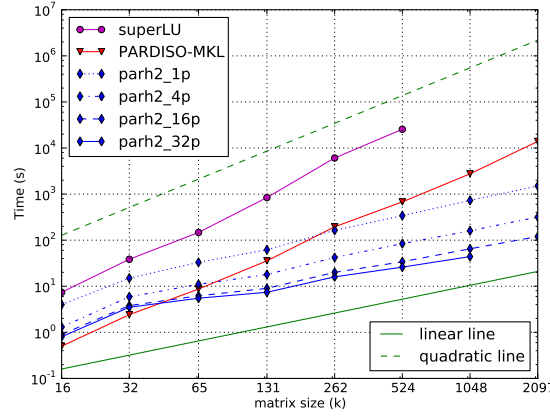


Fig. 4.3: Timing results for SuperLU, PARDISO-MKL and ParH2 with different MPI ranks (1, 4, 16 and 32). ParH2 is configured with  $\epsilon = 10^{-3}$  and  $\mathcal{B} = 16$ .

Before showing details of parallel efficiency, we present an experiment showing the effectiveness of overlapping d1-communication and d3-computation. Figure 4.4 illustrates a speed-up of about two times on 32 processors with the overlapping.

The last experiment shows the parallel efficiency of ParH2 in Figure 4.5 with configuration  $\epsilon = 10^{-3}$  and  $\mathcal{B} = 8$ . Figure 4.5(a) shows the efficiency corresponding to strong scaling, i.e., the matrix size is fixed while the number of processors increases. The speedup and parallel strong efficiency are computed with the following standard formulas:

$$\text{speedup} = \frac{\text{time}(1, N)}{\text{time}(np, N)} \quad \text{s-efficiency} = \frac{\text{speedup}}{np}$$

where  $N$  is the problem size,  $np$  is the number of processors,  $\text{time}(np, N)$  corresponds to the parallel running time with  $np$  processors for solving a matrix of size  $N$ . From Figure 4.5(a), the efficiency is around 80%, i.e., the speedup is about 25 at 32 processors for four million unknowns. Figure 4.5(b) shows the efficiency corresponding to weak scaling, i.e., the matrix size increases proportionally to the number of processors. The parallel weak efficiency is

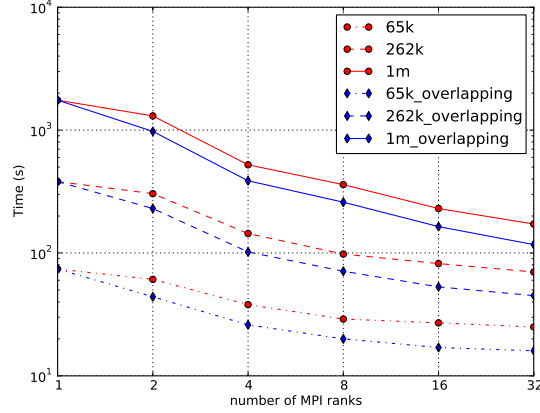
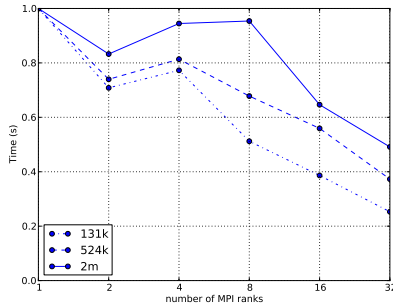


Fig. 4.4: Timing differences between non-overlapping and overlapping of d1-communication and d3-computation. The speedup is about two times on 32 processors ( $\epsilon = 10^{-3}$  and  $\mathcal{B} = 32$ ).

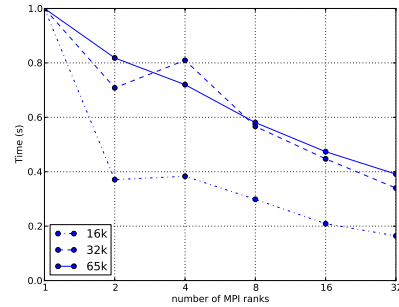
computed with the following formula:

$$\text{w-efficiency} = \frac{\text{time}(1, N)}{\text{time}(np, N \times np)}.$$

From Figure 4.5(b), the efficiency is around 75% at 32 processors for 131 thousand unknowns per processor.



(a) Strong scaling: the matrix size is fixed and the number of processes increases



(b) Weak scaling: the matrix size increases proportional to the number of processes

Fig. 4.5: Parallel efficiency of ParH2 with configuration  $\epsilon = 10^{-3}$  and  $\mathcal{B} = 8$ . The results for  $\mathcal{B} = 16, 32$  are very similar.

**5. Conclusions.** The pursuit to solve larger and larger matrices will never end. Here, we have shown the attractive features of our parallel algorithm: the factorization time, the solve time and the memory usage all scale nearly linearly. When our solver is used as a



preconditioner for an iterative solver, the iteration number stays almost constant. This nearly optimal scaling leads to faster running times on large matrices which beats many existing sparse direct solvers.

To solve really large problems that cannot be stored on a single machine, distributed memory parallel computing has to be used. Our solver achieves good speed-ups on more than one processor. The parallel algorithm requires exchanging only boundary information between neighbor processors, which is an order of magnitude less than the amount of computation to be done. It uses a lot of dense linear algebra, giving opportunity for parallelism using modern many core architectures.

**Appendix: proof of distance-2 property.** We use mathematical induction. Define the distance between two nodes as the length of the shortest path in the original graph. At the leaf level, initially edges only exist between nodes of distance-1 apart obviously. Suppose a new edge  $E'$  is introduced between two nodes of distance-3 during the elimination process. There must be another edge  $E$  between two nodes of distance-2 before  $E'$ . However, this is impossible because such an edge  $E$  will be compressed before the elimination happens. By induction, suppose distance-2 connection property holds at one level, and we want to show it also holds when the next level is constructed. Suppose an edge exists between two nodes of distance-3 when the next level is formed. Consider the children of those two nodes. There must be two of them of distance-3 at the previous level - a contradiction. Therefore the property holds at all levels.

#### REFERENCES

- [1] P. AMESTOY, C. ASHCRAFT, O. BOITEAU, A. BUTTARI, J.-Y. L'EXCELLENT, AND C. WEISBECKER, Improving Multifrontal Methods by Means of Block Low-rank Representations, *SIAM Journal on Scientific Computing*, 37 (2015), pp. A1451–A1474.
- [2] P. R. AMESTOY, T. A. DAVIS, AND I. S. DUFF, An Approximate Minimum Degree Ordering Algorithm, *SIAM Journal on Matrix Analysis and Applications*, 17 (1996), pp. 886–905.
- [3] A. AMINFAR, S. AMBIKASARAN, AND E. DARVE, A Fast Block Low-rank Dense Solver with Applications to Finite-element Matrices, *Journal of Computational Physics*, 304 (2016), pp. 170–188.
- [4] E. G. BOMAN, U. V. CATALYUREK, C. CHEVALIER, AND K. D. DEVINE, The Zoltan and Isorropia Parallel Toolkits for Combinatorial Scientific Computing: Partitioning, Ordering, and Coloring, *Scientific Programming*, 20 (2012), pp. 129–150.
- [5] Y. CHEN, T. A. DAVIS, W. W. HAGER, AND S. RAJAMANICKAM, Algorithm 887: CHOLMOD, Supernodal Sparse Cholesky Factorization and Update/Downdate, *ACM Transactions on Mathematical Software (TOMS)*, 35 (2008), p. 22.
- [6] C. CHEVALIER AND F. PELLEGRINI, PT-Scotch: A Tool for Efficient Parallel Graph Ordering, *Parallel computing*, 34 (2008), pp. 318–331.
- [7] T. A. DAVIS, Algorithm 832: UMFPACK V4. 3—an Unsymmetric-pattern Multifrontal Method, *ACM Transactions on Mathematical Software (TOMS)*, 30 (2004), pp. 196–199.
- [8] T. A. DAVIS, S. RAJAMANICKAM, AND W. M. SID-LAKHDAR, A Survey of Direct Methods for Sparse Linear Systems, *Acta Numerica*, 25 (2016), pp. 383–566.
- [9] J. W. DEMMEL, S. C. EISENSTAT, J. R. GILBERT, X. S. LI, AND J. W. H. LIU, A Supernodal Approach to Sparse Partial Pivoting, *SIAM J. Matrix Analysis and Applications*, 20 (1999), pp. 720–755.
- [10] I. S. DUFF, A. M. ERISMAN, AND J. K. REID, *Direct Methods for Sparse Matrices*, Clarendon press Oxford, 1986.
- [11] A. GEORGE, Nested Dissection of a Regular Finite Element Mesh, *SIAM Journal on Numerical Analysis*, 10 (1973), pp. 345–363.
- [12] P. GHYSELS, X. S. LI, F.-H. ROUET, S. WILLIAMS, AND A. NAPOV, An Efficient Multi-core Implementation of a Novel HSS-structured Multifrontal Solver Using Randomized Sampling, *arXiv preprint arXiv:1502.07405*, (2015).
- [13] W. GROPP, E. LUSK, N. DOSS, AND A. SKJELLUM, A High-performance, Portable Implementation of the MPI Message Passing Interface Standard, *Parallel computing*, 22 (1996), pp. 789–828.
- [14] W. HACKBUSCH, A Sparse Matrix Arithmetic Based on  $\mathcal{H}$ -matrices. Part i: Introduction to  $\mathcal{H}$ -matrices, *Computing*, 62 (1999), pp. 89–108.
- [15] ———, *Multi-grid Methods and Applications*, vol. 4, Springer Science & Business Media, 2013.

- [16] M. R. HESTENES AND E. STIEFEL, *Methods of Conjugate Gradients for Solving Linear Systems*, vol. 49, NBS, 1952.
- [17] K. L. HO AND L. YING, Hierarchical Interpolative Factorization for Elliptic Operators: Differential Equations, *Communications on Pure and Applied Mathematics*, (2015).
- [18] J. JEFFERS AND J. REINDERS, *Intel Xeon Phi Coprocessor High-performance Programming*, Newnes, 2013.
- [19] G. KARYPIS AND V. KUMAR, A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs, *SIAM Journal on scientific Computing*, 20 (1998), pp. 359–392.
- [20] A. KUZMIN, M. LUISIER, AND O. SCHENK, Fast Methods for Computing Selected Elements of the Greens Function in Massively Parallel Nanoelectronic Device Simulations, in *Euro-Par 2013 Parallel Processing*, F. Wolf, B. Mohr, and D. Mey, eds., vol. 8097 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2013, pp. 533–544.
- [21] Y. LI AND L. YING, Distributed-memory Hierarchical Interpolative Factorization, *arXiv preprint arXiv:1607.00346*, (2016).
- [22] H. POURANSARI, P. COULIER, AND E. DARVE, Fast Hierarchical Solvers for Sparse Matrices, *arXiv preprint arXiv:1510.07363*, (2015).
- [23] F.-H. ROUET, X. S. LI, P. GHYSELS, AND A. NAPOV, A Distributed-memory Package for Dense Hierarchically Semi-Separable Matrix Computations Using Randomization, *arXiv preprint arXiv:1503.05464*, (2015).
- [24] Y. SAAD, *Iterative Methods for Sparse Linear Systems*, Siam, 2003.
- [25] O. SCHENK, M. BOLLHÖFER, AND R. A. RÖMER, On Large-Scale Diagonalization Techniques for the Anderson Model of Localization, *SIAM Rev.*, 50 (2008), pp. 91–112.
- [26] O. SCHENK, A. WCHTER, AND M. HAGEMANN, Matching-based Preprocessing Algorithms to the Solution of Saddle-point Problems in Large-scale Nonconvex Interior-point Optimization, *Computational Optimization and Applications*, 36 (2007), pp. 321–341.
- [27] K. SHEN, J. N. CROSSLEY, A. W.-C. LUN, AND H. LIU, *The Nine Chapters on the Mathematical Art: Companion and Commentary*, Oxford University Press on Demand, 1999.
- [28] J. XIA, S. CHANDRASEKARAN, M. GU, AND X. S. LI, Superfast Multifrontal Method for Large Structured Linear Systems of Equations, *SIAM Journal on Matrix Analysis and Applications*, 31 (2009), pp. 1382–1411.
- [29] J. XU, Iterative Methods by Space Decomposition and Subspace Correction, *SIAM review*, 34 (1992), pp. 581–613.
- [30] K. YANG, H. POURANSARI, AND E. DARVE, Sparse Hierarchical Solvers with Guaranteed Convergence, *arXiv preprint arXiv:1611.03189*, (2016).

## SCALABLE THREE-TERM RECURRENCE PRECONDITIONED CONJUGATE GRADIENT METHODS

PAUL R. ELLER <sup>§</sup> AND MARK HOEMMEN <sup>¶</sup>

**Abstract.** The preconditioned conjugate gradient method (PCG) is a popular method for solving linear systems at scale. PCG requires frequent blocking allreduce collective operations that can limit performance at scale. We investigate PCG variations designed to reduce communication costs by decreasing the number of allreduces and by overlapping communication with computation using a non-blocking allreduce. This paper investigates three-term recurrence variations of PCG. This three-term recurrence variation allows us to develop PIPE2CG, a PCG method that overlaps two iterations of matrix-vector multiplies and preconditioner applications with a single non-blocking allreduce. PIPE2CG provides a more scalable method that performs better as communication costs increase and work per core decreases. This approach provides the potential to develop further pipelined non-blocking PCG methods.

**1. Introduction.** To achieve the best performance on extreme-scale systems, we must develop more scalable methods. Blocking collective operations are a barrier to scalability due to their increase in cost when scaling to higher core counts and due to requiring synchronization of all cores. Synchronization forces all processes to wait for the slowest process, which becomes particularly harmful on supercomputers due to the many sources of noise that cause performance variation across cores. Methods that require frequent collective operations and synchronization may run efficiently enough on some modern systems, but they will struggle to maintain good performance as supercomputers continue to grow.

The preconditioned conjugate gradient method (PCG) [4, 6] is a popular iterative method for solving sparse symmetric positive definite (SPD) systems of linear equations, however the allreduce used once or twice per iteration for many of the standard formulations of PCG is a barrier to scalability. Therefore in order to obtain improved performance at scale, we need to minimize the allreduce cost and avoid synchronization to produce methods capable of scaling effectively on large-scale systems.

We can rearrange PCG to reduce communication latency by combining multiple allreduces into a single allreduce and to overlap communication and computation using non-blocking allreduces. Using the standard two-term recurrence PCG method to derive a PCG method that overlaps multiple iterations of PCG with a single non-blocking allreduce seems like a promising approach to produce more scalable methods.

Unfortunately this approach requires an additional matrix-vector multiply and preconditioner application each iteration to produce an accurate solution. Using suggestions from [5], we instead use a three-term recurrence variation of PCG (PCG3) based on the algorithm from [6]. This method avoids dependencies that limit our ability to rearrange the standard formulation, allowing us to develop a two-step pipelined, non-blocking PCG method, and providing the potential to develop further pipelined methods.

We previously investigated scalable two-term recurrence variations of PCG and the scalable three-term recurrence PIPE2CG method in [1]. That paper presented some methods, discussed implementation details, and showed detailed performance results and analysis, but had limited space to show and discuss the derivation of PIPE2CG. This paper addresses this issue by providing a more detailed description of how to derive PIPE2CG, including presenting a three-term recurrence variation of PIPECG as an intermediate step when deriving PIPE2CG.

---

<sup>§</sup>University of Illinois at Urbana-Champaign Department of Computer Science, eller3@illinois.edu

<sup>¶</sup>Sandia National Laboratories, mhoemme@sandia.gov

**Algorithm 2.1** Preconditioned Conjugate Gradient method

---

**Input:**  $A$ :  $n \times n$  SPD matrix,  $x_0$ : initial guess,  $b$ : right-hand side,  $M$ : prec  
**Output:**  $x_j$ : approximate solution

$r_0 \leftarrow b - Ax_0 \quad z_0 \leftarrow Mr_0$

$\gamma_0 \leftarrow (z_0, r_0) \quad norm_0 \leftarrow \sqrt{(z_0, z_0)} \quad \triangleright 2 \text{ Merged Operations}$   
Allreduce on  $\gamma_0, norm_0 \quad \triangleright \text{Blocking Allreduce}$

**for**  $j=1,2,\dots$ ,until convergence **do**  
  **if**  $j > 1$  **then**  $\beta_j \leftarrow \gamma_{j-1}/\gamma_{j-2}$   
  **else**  $\beta_j \leftarrow 0.0$   
   $p_j \leftarrow z_{j-1} + \beta p_{j-1}$   
   $w_j \leftarrow Ap_j$   
   $\delta_j \leftarrow (p_j, w_j)$   
  Allreduce on  $\delta_j \quad \triangleright \text{Blocking Allreduce}$   
   $\alpha_j \leftarrow \gamma_{j-1}/\delta_j$   
   $x_j \leftarrow x_{j-1} + \alpha_j p_j$   
   $r_j \leftarrow r_{j-1} - \alpha_j w_j$   
   $z_j \leftarrow Mr_j$   
   $\gamma_j \leftarrow (z_j, r_j) \quad norm_j \leftarrow \sqrt{(z_j, z_j)} \quad \triangleright 2 \text{ Merged Operations}$   
  Allreduce on  $\gamma_j, norm_j \quad \triangleright \text{Blocking Allreduce}$

---

**2. PCG.** The preconditioned conjugate gradient method solves a symmetric positive definite (SPD) linear system  $Ax = b$  where  $A$  is an SPD  $n \times n$  matrix,  $x$  is an  $n$ -vector to solve for, and  $b$  is a right hand side  $n$ -vector. This method is used iteratively and often converges in far fewer than  $n$  iterations. In practice we need a preconditioner to accelerate convergence by solving the system  $M^{-1}Ax = M^{-1}b$ , where  $M$  is an  $n \times n$  matrix.

**Algorithm 3.1** Three-term Recurrence PCG method

---

**Input:**  $A$ :  $n \times n$  SPD matrix,  $x_0$ : initial guess,  $b$ : right-hand side,  $M$ : prec  
**Output:**  $x_j$ : approximate solution

$r_0 \leftarrow b - Ax_0 \quad z_0 \leftarrow Mr_0 \quad w_0 \leftarrow Az_0$

$\delta_0 \leftarrow (z_0, w_0) \quad \beta_0 \leftarrow (z_0, r_0) \quad norm_0 \leftarrow \sqrt{(z_0, z_0)} \quad \triangleright 3 \text{ Merged Operations}$   
Allreduce on  $\delta_0, \beta_0, norm_0 \quad \triangleright \text{Blocking Allreduce}$

**for**  $j=1,2,\dots$ ,until convergence **do**  
   $\gamma_j \leftarrow \beta_{j-1}/\delta_{j-1}$   
  **if**  $j > 1$  **then**  $\rho_j \leftarrow 1/(1 - (\gamma_j/\gamma_{j-1})(\beta_{j-1}/\beta_{j-2})(1/\rho_{j-1}))$   
  **else**  $\rho_j \leftarrow 1$   
   $x_j \leftarrow \rho_j(x_{j-1} + \gamma_j z_{j-1}) + (1 - \rho_j)x_{j-2}$   
   $r_j \leftarrow \rho_j(r_{j-1} - \gamma_j w_{j-1}) + (1 - \rho_j)r_{j-2}$   
   $z_j \leftarrow Mr_j \quad w_j \leftarrow Az_j$   
   $\delta_j \leftarrow (z_j, w_j) \quad \beta_j \leftarrow (z_j, r_j) \quad norm_j \leftarrow \sqrt{(z_j, z_j)} \quad \triangleright 3 \text{ Merged Operations}$   
  Allreduce on  $\delta_j, \beta_j, norm_j \quad \triangleright \text{Blocking Allreduce}$

---

Method	Vectors	VecOps	Allreduces	Overlap	Init Costs
PCG	6	6	2 Blocking	None	1 PC
PCG3	9	7	1 Blocking	None	1 PC, 1 Matvec
PIPECG3	15	11	1 Non-blocking	Matvec, PC	2 PC, 2 Matvec
PIPE2CG*	27	24	1 Non-blocking	2 Matvec, 2 PC	3 PC, 3 Matvec

Table 3.1: Differences between each PCG method. Assumes cost of three-term vector operation is double cost of two-term vector operation. \*Note that PIPE2CG computes two iterations of PCG each iteration.

**3. Three-term Recurrence PCG.** Algorithm 2.1 is the most commonly used algorithm for PCG; however, there are some other variations that are equivalent in exact arithmetic. The three-term recurrence variation of PCG [6] uses a three-term recurrence of the form  $r_j \leftarrow \rho_j(r_{j-1} - \gamma_j w_{j-1}) + (1 - \rho_j)r_{j-2}$  instead of the two-term recurrence of the form  $r_j \leftarrow r_{j-1} - \alpha_j w_j$ . Using the other properties of the conjugate gradient method allows us to produce Algorithm 3.1.

The standard PCG algorithm has a cyclic dependency between  $p_j$  and  $r_j$ , which we are able to avoid with the three-term recurrence variation that directly computes  $x_j$  and  $r_j$  without the need for  $p_j$ . We can then rearrange this method to create PIPE2CG without needing additional matrix-vector multiplies or preconditioner applications. This approach requires some additional vector storage compared to the two-term recurrence method, but otherwise has similar costs. However using three-term recurrences has been shown to produce less accurate residuals than two-term recurrences, resulting in more limited accuracy in some cases [3].

We can rearrange PCG3 to decrease the number of allreduces or overlap communication and computation using recurrence relations to change the vector used in key kernels, allowing us to rearrange the order of the kernels. This allows us to derive methods that are equivalent to PCG3 in exact arithmetic but have more scalable properties. The number of matrix-vector multiplies and preconditioner applications each iteration remains constant, but additional vector multiply-adds and initialization costs are introduced. For the best performance, we want to use the method that effectively minimizes the communication from the allreduce while introducing as little overhead as possible. This approach produces three variations of PCG3 that we investigate. Table 3.1 shows an overview of key differences between these methods.

**4. PIPECG3.** We can derive a three-term recurrence variation of PIPECG by starting with PCG3 and using a similar process as for the two-term recurrence variation of PIPECG shown in [2]. We can derive this method by substituting

$$r_j \leftarrow \rho_j(r_{j-1} - \gamma_{j-1}w_{j-1}) + (1 - \rho_j)r_{j-2} \quad \text{into} \quad z_j \leftarrow Mr_j$$

and simplifying

$$\begin{aligned} z_j &\leftarrow M(\rho_j(r_{j-1} - \gamma_{j-1}w_{j-1}) + (1 - \rho_j)r_{j-2}) \\ z_j &\leftarrow \rho_j(Mr_{j-1} - \gamma_{j-1}Mw_{j-1}) + ((1 - \rho_j)Mr_{j-2}) \\ p_{j-1} &\leftarrow Mw_{j-1} \text{ and } z_j \leftarrow \rho_j(z_{j-1} - \gamma_{j-1}p_{j-1}) + (1 - \rho_j)z_{j-2} \end{aligned}$$

to get new equations for  $p_j$  and  $z_j$ . We can then move  $p_{j-1}$  to the previous iteration after

**Algorithm 4.1** Three-term Recurrence Pipelined PCG method

---

**Input:**  $A$ :  $n \times n$  SPD matrix,  $x_0$ : initial guess,  $b$ : right-hand side,  $M$ : prec  
**Output:**  $x_j$ : approximate solution

$r_0 \leftarrow b - Ax_0 \quad z_0 \leftarrow Mr_0 \quad w_0 \leftarrow Az_0$

$\delta_0 \leftarrow (z_0, w_0) \quad \beta_0 \leftarrow (z_0, r_0) \quad norm_0 \leftarrow \sqrt{(z_0, z_0)} \quad \triangleright 3 \text{ Merged Operations}$

MPI\_Iallreduce on  $\delta_0, \beta_0, norm_0 \quad \triangleright \text{Non-blocking Allreduce}$

$p_0 \leftarrow Mw_0 \quad q_0 \leftarrow Ap_0 \quad \triangleright \text{Overlap Comm and Comp}$

**for**  $j=1,2,\dots$ , until convergence **do**

$\gamma_{j-1} \leftarrow \beta_{j-1}/\delta_{j-1}$

**if**  $j > 1$  **then**  $\rho_j \leftarrow 1/(1 - (\gamma_{j-1}/\gamma_{j-2})(\beta_{j-1}/\beta_{j-2})(1/\rho_{j-1}))$

**else**  $\rho_j \leftarrow 1$

$x_j \leftarrow \rho_j(x_{j-1} + \gamma_{j-1}z_{j-1}) + (1 - \rho_j)x_{j-2}$

$r_j \leftarrow \rho_j(r_{j-1} - \gamma_{j-1}w_{j-1}) + (1 - \rho_j)r_{j-2}$

$z_j \leftarrow \rho_j(z_{j-1} - \gamma_{j-1}p_{j-1}) + (1 - \rho_j)z_{j-2} \quad \triangleright 7 \text{ Merged Operations}$

$w_j \leftarrow \rho_j(w_{j-1} - \gamma_{j-1}q_{j-1}) + (1 - \rho_j)w_{j-2}$

$\delta_j \leftarrow (z_j, w_j) \quad \beta_j \leftarrow (z_j, r_j) \quad norm_j \leftarrow \sqrt{(z_j, z_j)}$

MPI\_Iallreduce on  $\delta_j, \beta_j, norm_j \quad \triangleright \text{Non-blocking Allreduce}$

$p_j \leftarrow Mw_j \quad q_j \leftarrow Ap_j \quad \triangleright \text{Overlap Comm and Comp}$

---

$w_j$  is computed. We can repeat this same process with  $w_j \leftarrow Az_j$  and our new equation for  $z_j$  to get

$$w_j \leftarrow \rho_j(w_{j-1} - \gamma_{j-1}q_{j-1}) + (1 - \rho_j)w_{j-2} \quad \text{and} \quad q_{j-1} \leftarrow Ap_{j-1}.$$

We can then move  $q_{j-1}$  to the previous iteration after  $p_j$  is computed. Since the dot products do not use  $p_j$  and  $q_j$ , we can use a non-blocking allreduce and overlap it with the matrix-vector multiply and preconditioner, giving us the PIPECG3 method shown in Algorithm 4.1.

**5. Pipelined 2-Iteration PCG.** We can rearrange PIPECG3 to get a 2-step pipelined, non-blocking PCG method (PIPE2CG). The PIPE2CG algorithm computes two iterations of PCG at once. As a result, we overlap a single allreduce with two matrix-vector multiplies and two preconditioner applications each iteration. This potentially allows us to hide the allreduce cost when the work per core becomes too small for PIPECG or PIPECG3 to overlap effectively. However, the initialization step continues to grow, requiring three matrix-vector multiplies and three preconditioner applications. Therefore we need to ensure that the solver will require enough iterations to overcome the increased initialization cost and obtain a speedup. We can likely produce further pipelined methods that may produce further speedups in some cases. For PIPE2CG performance results, see [1].

We can derive this method by first combining two iterations of PIPECG3 into a single iteration to get Algorithm 5.1. We can then move the first matrix-vector multiply and preconditioner application into the previous iteration immediately after the second matrix-vector multiply and preconditioner application. We do this by substituting

$$w_j \leftarrow \rho_j(w_{j-1} - \gamma_{j-1}q_{j-1}) + (1 - \rho_j)w_{j-2} \text{ into } p_j \leftarrow Mw_j,$$

**Algorithm 5.1** 2-Iteration Three-term Recurrence Pipelined PCG method**Input:**  $A$ :  $n \times n$  SPD matrix,  $x_0$ : initial guess,  $b$ : right-hand side,  $M$ : prec**Output:**  $x_j$ : approximate solution

$$r_0 \leftarrow b - Ax_0 \quad z_0 \leftarrow Mr_0 \quad w_0 \leftarrow Az_0$$

$$\delta_0 \leftarrow (z_0, w_0) \quad \beta_0 \leftarrow (z_0, r_0) \quad norm_0 \leftarrow \sqrt{(z_0, z_0)}$$

MPI\_Allreduce on  $\delta_0, \beta_0, norm_0$ 

$$p_0 \leftarrow Mw_0 \quad q_0 \leftarrow Ap_0$$

**for**  $j=1,3,\dots$ ,until convergence **do**

$$\gamma_{j-1} \leftarrow \beta_{j-1}/\delta_{j-1}$$

$$\textbf{if } j > 1 \textbf{ then } \rho_j \leftarrow 1/(1 - (\gamma_{j-1}/\gamma_{j-2})(\beta_{j-1}/\beta_{j-2})(1/\rho_{j-1}))$$

$$\textbf{else } \rho_j \leftarrow 1$$

$$x_j \leftarrow \rho_j(x_{j-1} + \gamma_{j-1}z_{j-1}) + (1 - \rho_j)x_{j-2}$$

$$r_j \leftarrow \rho_j(r_{j-1} - \gamma_{j-1}w_{j-1}) + (1 - \rho_j)r_{j-2}$$

$$z_j \leftarrow \rho_j(z_{j-1} - \gamma_{j-1}p_{j-1}) + (1 - \rho_j)z_{j-2}$$

$$w_j \leftarrow \rho_j(w_{j-1} - \gamma_{j-1}q_{j-1}) + (1 - \rho_j)w_{j-2}$$

$$\delta_j \leftarrow (z_j, w_j) \quad \beta_j \leftarrow (z_j, r_j) \quad norm_j \leftarrow \sqrt{(z_j, z_j)}$$

MPI\_Allreduce on  $\delta_j, \beta_j, norm_j$ 

$$p_j \leftarrow Mw_j \quad q_j \leftarrow Ap_j$$

$$\gamma_j \leftarrow \beta_j/\delta_j \quad \rho_{j+1} \leftarrow 1/(1 - (\gamma_j/\gamma_{j-1})(\beta_j/\beta_{j-1})(1/\rho_j))$$

$$x_{j+1} \leftarrow \rho_{j+1}(x_j + \gamma_j z_j) + (1 - \rho_{j+1})x_{j-1}$$

$$r_{j+1} \leftarrow \rho_{j+1}(r_j - \gamma_j w_j) + (1 - \rho_{j+1})r_{j-1}$$

$$z_{j+1} \leftarrow \rho_{j+1}(z_j - \gamma_j p_j) + (1 - \rho_{j+1})z_{j-1}$$

$$w_{j+1} \leftarrow \rho_{j+1}(w_j - \gamma_j q_j) + (1 - \rho_{j+1})w_{j-1}$$

$$\delta_{j+1} \leftarrow (z_{j+1}, w_{j+1}) \quad \beta_{j+1} \leftarrow (z_{j+1}, r_{j+1}) \quad norm_{j+1} \leftarrow \sqrt{(z_{j+1}, z_{j+1})}$$

MPI\_Allreduce on  $\delta_{j+1}, \beta_{j+1}, norm_{j+1}$ 

$$p_{j+1} \leftarrow Mw_{j+1} \quad q_{j+1} \leftarrow Ap_{j+1}$$

and simplifying to get

$$c_{j-1} \leftarrow Mq_{j-1} \text{ and } p_j \leftarrow \rho_j(p_{j-1} - \gamma_{j-1}c_{j-1}) + (1 - \rho_j)p_{j-2}.$$

Repeating the same process with  $q_j \leftarrow Ap_j$  and our new equation for  $p_j$  gives us

$$d_{j-1} \leftarrow Ac_{j-1} \text{ and } q_j \leftarrow \rho_j(q_{j-1} - \gamma_{j-1}d_{j-1}) + (1 - \rho_j)q_{j-2}.$$

Moving these new equations for  $c_{j-1}$  and  $d_{j-1}$  into the previous iteration gives us Algorithm 5.2, which has two iterations of matrix-vector multiplies and preconditioner applications gathered together.

Next we need to move the dot products for the first  $\delta_j$  and  $\beta_j$  into the previous iteration. We do this by substituting the full equations for  $z_j$ ,  $w_j$ , and  $r_j$  into the dot products, giving us a larger set of dot products that rely on vectors from the previous iteration. This gives us

$$\delta_j \leftarrow (\rho_j(z_{j-1} - \gamma_{j-1}p_{j-1}) + (1 - \rho_j)z_{j-2}, \rho_j(w_{j-1} - \gamma_{j-1}q_{j-1}) + (1 - \rho_j)w_{j-2}) \text{ and } \beta_j \leftarrow (\rho_j(z_{j-1} - \gamma_{j-1}p_{j-1}) + (1 - \rho_j)z_{j-2}, \rho_j(r_{j-1} - \gamma_{j-1}w_{j-1}) + (1 - \rho_j)r_{j-2}).$$

**Algorithm 5.2** Pipelined 2-iteration PCG - Gather matrix kernels

---


$$\begin{aligned}
r_0 &\leftarrow b - Ax_0 & z_0 &\leftarrow Mr_0 & w_0 &\leftarrow Az_0 \\
\delta_0 &\leftarrow (z_0, w_0) & \beta_0 &\leftarrow (z_0, r_0) & norm_0 &\leftarrow \sqrt{(z_0, z_0)}
\end{aligned}$$

$$\begin{aligned}
&\text{MPI\_Allreduce on } \delta_0, \beta_0, norm_0 \\
p_0 &\leftarrow Mw_0 & q_0 &\leftarrow Ap_0 & c_0 &\leftarrow Mq_0 & d_0 &\leftarrow Ac_0
\end{aligned}$$

**for**  $j=1,3,\dots$ ,until convergence **do**

$$\begin{aligned}
&\gamma_{j-1} \leftarrow \beta_{j-1}/\delta_{j-1} \\
&\text{if } j > 1 \text{ then } \rho_j \leftarrow 1/(1 - (\gamma_{j-1}/\gamma_{j-2})(\beta_{j-1}/\beta_{j-2})(1/\rho_{j-1})) \\
&\text{else } \rho_j \leftarrow 1 \\
&x_j \leftarrow \rho_j(x_{j-1} + \gamma_{j-1}z_{j-1}) + (1 - \rho_j)x_{j-2} \\
&r_j \leftarrow \rho_j(r_{j-1} - \gamma_{j-1}w_{j-1}) + (1 - \rho_j)r_{j-2} \\
&z_j \leftarrow \rho_j(z_{j-1} - \gamma_{j-1}p_{j-1}) + (1 - \rho_j)z_{j-2} \\
&w_j \leftarrow \rho_j(w_{j-1} - \gamma_{j-1}q_{j-1}) + (1 - \rho_j)w_{j-2} \\
&p_j \leftarrow \rho_j(p_{j-1} - \gamma_{j-1}c_{j-1}) + (1 - \rho_j)p_{j-2} \\
&q_j \leftarrow \rho_j(q_{j-1} - \gamma_{j-1}d_{j-1}) + (1 - \rho_j)q_{j-2} \\
&\delta_j \leftarrow (z_j, w_j) & \beta_j &\leftarrow (z_j, r_j) \\
&\text{Allreduce on } \delta_j, \beta_j \\
&\gamma_j \leftarrow \beta_j/\delta_j & \rho_{j+1} &\leftarrow 1/(1 - (\gamma_j/\gamma_{j-1})(\beta_j/\beta_{j-1})(1/\rho_j)) \\
&x_{j+1} \leftarrow \rho_{j+1}(x_j + \gamma_j z_j) + (1 - \rho_{j+1})x_{j-1} \\
&r_{j+1} \leftarrow \rho_{j+1}(r_j - \gamma_j w_j) + (1 - \rho_{j+1})r_{j-1} \\
&z_{j+1} \leftarrow \rho_{j+1}(z_j - \gamma_j p_j) + (1 - \rho_{j+1})z_{j-1} \\
&w_{j+1} \leftarrow \rho_{j+1}(w_j - \gamma_j q_j) + (1 - \rho_{j+1})w_{j-1} \\
&\delta_{j+1} \leftarrow (z_{j+1}, w_{j+1}) & \beta_{j+1} &\leftarrow (z_{j+1}, r_{j+1}) & norm_{j+1} &\leftarrow \sqrt{(z_{j+1}, z_{j+1})} \\
&\text{MPI\_Allreduce on } \delta_{j+1}, \beta_{j+1}, norm_j \\
p_{j+1} &\leftarrow Mw_{j+1} & q_{j+1} &\leftarrow Ap_{j+1} & c_{j+1} &\leftarrow Mq_{j+1} & d_{j+1} &\leftarrow Ac_{j+1}
\end{aligned}$$


---

We can simplify these equations, move the multipliers into  $\phi$ , and move the dot products to individual equations. This gives us simpler equations for  $\delta$  and  $\beta$  and allows us to compute the dot products in the previous iteration.

We can use the vector equalities  $q = Ap$ ,  $p = Mw$ ,  $w = Az$ , and  $z = Mr$  to change the vectors included in a dot product, allowing us to reduce the number of dot products to 10 unique dot products. Note that these transformations assume both a symmetric matrix and preconditioner.

Transformations for  $\delta$ :

$$\begin{aligned}
(z_j, w_j) &= (Mr_j, w_j) = (r_j, Mw_j) = (r_j, p_j) \\
(p_j, w_j) &= (Aq_j, w_j) = (q_j, Aw_j) = (z_j, q_j) \\
(z_j, w_{j-1}) &= (Aw_j, w_{j-1}) = (w_j, Aw_{j-1}) = (w_j, z_{j-1}) \\
(p_j, w_{j-1}) &= (Aq_j, w_{j-1}) = (q_j, Aw_{j-1}) = (q_j, z_{j-1})
\end{aligned}$$

Transformations for  $\beta$ :

$$\begin{aligned}
(z_{j-1}, r_j) &= (Mr_{j-1}, r_j) = (r_{j-1}, Mr_j) = (z_j, r_{j-1}) \\
(p_j, r_{j-1}) &= (Mw_j, r_{j-1}) = (w_j, Mr_{j-1}) = (z_{j-1}, w_j) = (z_j, w_{j-1})
\end{aligned}$$



Next we want to move these dot products into the previous iteration after the vector operations and before the matrix-vector multiplies and preconditioner applications. We are able to move most of the dot products to this point in the algorithm; however, there are a couple dot products that use  $p_j$  and  $q_j$ . Therefore we can only move these dot products to the point after we compute  $p_j$  and  $q_j$ , giving us Algorithm 5.3.

---

**Algorithm 5.3** Pipelined 2-iteration PCG - Move Dot Products

---

```

 $r_0 \leftarrow b - Ax_0 \quad z_0 \leftarrow Mr_0 \quad w_0 \leftarrow Az_0$ 
 $\delta_0 \leftarrow (z_0, w_0) \quad \beta_0 \leftarrow (z_0, r_0) \quad norm_0 \leftarrow \sqrt{(z_0, z_0)}$ 

MPI_Allreduce on  $\delta_0, \beta_0, norm_0$ 
 $p_0 \leftarrow Mw_0 \quad q_0 \leftarrow Ap_0 \quad c_0 \leftarrow Mq_0 \quad d_0 \leftarrow Ac_0$ 

for  $j=1,3,\dots$ , until convergence do
  if  $j > 1$  then
     $\rho_{j-1} \leftarrow 1/(1 - (\gamma_{j-1}\beta_{j-1})/(\gamma_{j-2}\beta_{j-2}\rho_{j-2})) \quad \gamma_{j-1} \leftarrow \beta_{j-2}/\delta_{j-2}$ 
     $\phi \leftarrow [\rho_{j-1}, -\rho_{j-1}\gamma_{j-1}, (1 - \rho_{j-1})]$ 
     $\delta_{j-1} \leftarrow \phi_0\phi_0\lambda_0 + 2\phi_0\phi_1\lambda_1 + 2\phi_0\phi_2\lambda_2 + \phi_1\phi_1\lambda_3 + 2\phi_1\phi_2\lambda_4 + \phi_2\phi_2\lambda_5$ 
     $\beta_{j-1} \leftarrow \phi_0\phi_0\lambda_6 + 2\phi_0\phi_1\lambda_0 + 2\phi_0\phi_2\lambda_7 + \phi_1\phi_1\lambda_1 + 2\phi_1\phi_2\lambda_2 + \phi_2\phi_2\lambda_8$ 
     $\gamma_j \leftarrow \beta_{j-1}/\delta_{j-1} \quad \rho_j \leftarrow 1/(1 - (\gamma_j\beta_{j-1})/(\gamma_{j-1}\beta_{j-2}\rho_{j-1}))$ 
  else
     $\rho_j \leftarrow 1 \quad \gamma_j \leftarrow \beta_{j-1}/\delta_{j-1}$ 
  if  $j > 1$  then
     $x_{j-1} \leftarrow \rho_{j-1}(x_{j-2} + \gamma_{j-1}z_{j-2}) + (1 - \rho_j)x_{j-3}$ 
     $r_{j-1} \leftarrow \rho_{j-1}(r_{j-2} - \gamma_{j-1}w_{j-2}) + (1 - \rho_j)r_{j-3}$ 
     $z_{j-1} \leftarrow \rho_{j-1}(z_{j-2} - \gamma_{j-1}p_{j-2}) + (1 - \rho_j)z_{j-3}$ 
     $w_{j-1} \leftarrow \rho_{j-1}(w_{j-2} - \gamma_{j-1}q_{j-2}) + (1 - \rho_j)w_{j-3}$ 
     $p_{j-1} \leftarrow \rho_{j-1}(p_{j-2} - \gamma_{j-1}c_{j-2}) + (1 - \rho_j)p_{j-3}$ 
     $q_{j-1} \leftarrow \rho_{j-1}(q_{j-2} - \gamma_{j-1}d_{j-2}) + (1 - \rho_j)q_{j-3}$ 
     $x_j \leftarrow \rho_j(x_{j-1} + \gamma_j z_{j-1}) + (1 - \rho_j)x_{j-2}$ 
     $r_j \leftarrow \rho_j(r_{j-1} - \gamma_j w_{j-1}) + (1 - \rho_j)r_{j-2}$ 
     $z_j \leftarrow \rho_j(z_{j-1} - \gamma_j p_{j-1}) + (1 - \rho_j)z_{j-2}$ 
     $w_j \leftarrow \rho_j(w_{j-1} - \gamma_j q_{j-1}) + (1 - \rho_j)w_{j-2}$ 
     $\lambda_0 \leftarrow (z_j, w_j) \quad \lambda_2 \leftarrow (z_j, w_{j-1}) \quad \lambda_5 \leftarrow (z_{j-1}, w_{j-1})$ 
     $\lambda_6 \leftarrow (z_j, r_j) \quad \lambda_7 \leftarrow (z_j, r_{j-1}) \quad \lambda_8 \leftarrow (z_{j-1}, r_{j-1}) \quad \lambda_9 \leftarrow (z_j, z_j)$ 
     $\delta_j \leftarrow \lambda_0 \quad \beta_j \leftarrow \lambda_6 \quad norm_j \leftarrow \sqrt{\lambda_9}$ 
     $p_j \leftarrow Mw_j \quad q_j \leftarrow Ap_j$ 
     $\lambda_1 \leftarrow (z_j, q_j) \quad \lambda_3 \leftarrow (p_j, q_j) \quad \lambda_4 \leftarrow (p_j, w_{j-1})$ 
    MPI_Allreduce on  $\lambda_0$  to  $\lambda_9$ 
     $c_j \leftarrow Mq_j \quad d_j \leftarrow Ac_j$ 

```

---

Therefore we must move  $p_j \leftarrow Mw_j$  and  $q_j \leftarrow Ap_j$  into the previous iteration so we can gather all of the dot products together. We substitute

$w_j \leftarrow \rho_j(w_{j-1} - \gamma_j q_{j-1}) + (1 - \rho_j)w_{j-2}$  into  $p_j \leftarrow Mw_j$  and  
 $p_j \leftarrow \rho_j(p_{j-1} - \gamma_j c_{j-1}) + (1 - \rho_j)p_{j-2}$  into  $q_j \leftarrow Ap_j$

to get

$$p_j \leftarrow \rho_j(p_{j-1} - \gamma_j c_{j-1}) + (1 - \rho_j)p_{j-2} \text{ and} \\ q_j \leftarrow \rho_j(p_{j-1} - \gamma_j d_{j-1}) + (1 - \rho_j)q_{j-2}.$$

Since these computations for  $p_j$  and  $q_j$  rely on  $c_{j-1}$  and  $d_{j-1}$ , we must insert computations for  $c_{j-1} = Mq_{j-1}$  and  $d_{j-1} = Ac_{j-1}$  prior to these vector operations. This gives us Algorithm 5.4.

---

**Algorithm 5.4** Pipelined 2-iteration PCG - Gather Dot Products

---

$$r_0 \leftarrow b - Ax_0 \quad z_0 \leftarrow Mr_0 \quad w_0 \leftarrow Az_0 \\ \delta_0 \leftarrow (z_0, w_0) \quad \beta_0 \leftarrow (z_0, r_0) \quad norm_0 \leftarrow \sqrt{(z_0, z_0)}$$

MPI\_Allreduce on  $\delta_0, \beta_0, norm_0$

$$p_0 \leftarrow Mw_0 \quad q_0 \leftarrow Ap_0 \quad c_0 \leftarrow Mq_0 \quad d_0 \leftarrow Ac_0$$

**for**  $j=1,3,\dots$ ,until convergence **do**

**if**  $j > 1$  **then**

$$\rho_{j-1} \leftarrow 1/(1 - (\gamma_{j-1}\beta_{j-1})/(\gamma_{j-2}\beta_{j-2}\rho_{j-2})) \quad \gamma_{j-1} \leftarrow \beta_{j-2}/\delta_{j-2} \\ \phi \leftarrow [\rho_{j-1}, -\rho_{j-1}\gamma_{j-1}, (1 - \rho_{j-1})] \\ \delta_{j-1} \leftarrow \phi_0\phi_0\lambda_0 + 2\phi_0\phi_1\lambda_1 + 2\phi_0\phi_2\lambda_2 + \phi_1\phi_1\lambda_3 + 2\phi_1\phi_2\lambda_4 + \phi_2\phi_2\lambda_5 \\ \beta_{j-1} \leftarrow \phi_0\phi_0\lambda_6 + 2\phi_0\phi_1\lambda_0 + 2\phi_0\phi_2\lambda_7 + \phi_1\phi_1\lambda_1 + 2\phi_1\phi_2\lambda_2 + \phi_2\phi_2\lambda_8 \\ \gamma_j \leftarrow \beta_{j-1}/\delta_{j-1} \quad \rho_j \leftarrow 1/(1 - (\gamma_j\beta_{j-1})/(\gamma_{j-1}\beta_{j-2}\rho_{j-1}))$$

**else**

$$\rho_j \leftarrow 1 \quad \gamma_j \leftarrow \beta_{j-1}/\delta_{j-1}$$

**if**  $j > 1$  **then**

$$x_{j-1} \leftarrow \rho_{j-1}(x_{j-2} + \gamma_{j-1}z_{j-2}) + (1 - \rho_{j-1})x_{j-3} \\ r_{j-1} \leftarrow \rho_{j-1}(r_{j-2} - \gamma_{j-1}w_{j-2}) + (1 - \rho_{j-1})r_{j-3} \\ z_{j-1} \leftarrow \rho_{j-1}(z_{j-2} - \gamma_{j-1}p_{j-2}) + (1 - \rho_{j-1})z_{j-3} \\ w_{j-1} \leftarrow \rho_{j-1}(w_{j-2} - \gamma_{j-1}q_{j-2}) + (1 - \rho_{j-1})w_{j-3} \\ p_{j-1} \leftarrow \rho_{j-1}(p_{j-2} - \gamma_{j-1}c_{j-2}) + (1 - \rho_{j-1})p_{j-3} \\ q_{j-1} \leftarrow \rho_{j-1}(q_{j-2} - \gamma_{j-1}d_{j-2}) + (1 - \rho_{j-1})q_{j-3} \\ c_{j-1} \leftarrow Mq_{j-1} \quad d_{j-1} \leftarrow Ac_{j-1}$$

$$x_j \leftarrow \rho_j(x_{j-1} + \gamma_j z_{j-1}) + (1 - \rho_j)x_{j-2} \\ r_j \leftarrow \rho_j(r_{j-1} - \gamma_j w_{j-1}) + (1 - \rho_j)r_{j-2} \\ z_j \leftarrow \rho_j(z_{j-1} - \gamma_j p_{j-1}) + (1 - \rho_j)z_{j-2} \\ w_j \leftarrow \rho_j(w_{j-1} - \gamma_j q_{j-1}) + (1 - \rho_j)w_{j-2} \\ p_j \leftarrow \rho_j(p_{j-1} - \gamma_j c_{j-1}) + (1 - \rho_j)p_{j-2} \\ q_j \leftarrow \rho_j(q_{j-1} - \gamma_j d_{j-1}) + (1 - \rho_j)q_{j-2}$$

$$\lambda_0 \leftarrow (z_j, w_j) \quad \lambda_1 \leftarrow (z_j, q_j) \quad \lambda_2 \leftarrow (z_j, w_{j-1}) \quad \lambda_3 \leftarrow (p_j, q_j) \\ \lambda_4 \leftarrow (p_j, w_{j-1}) \quad \lambda_5 \leftarrow (z_{j-1}, w_{j-1}) \quad \lambda_6 \leftarrow (z_j, r_j) \quad \lambda_7 \leftarrow (z_j, r_{j-1}) \\ \lambda_8 \leftarrow (z_{j-1}, r_{j-1}) \quad \lambda_9 \leftarrow (z_j, z_j) \\ \delta_j \leftarrow \lambda_0 \quad \beta_j \leftarrow \lambda_6 \quad norm_j \leftarrow \sqrt{\lambda_9}$$

MPI\_Allreduce on  $\lambda_0$  to  $\lambda_9$

$$c_j \leftarrow Mq_j \quad d_j \leftarrow Ac_j$$


---

The resulting algorithm allows us to overlap the non-blocking allreduce with one matrix-vector multiply and preconditioner application, but not both. Additionally we cannot

pipeline all of the vector operations. Therefore we need to move the first matrix-vector multiply and preconditioner to the previous iteration. We do this by substituting

$q_{j-1} \leftarrow \rho_j(q_{j-2} - \gamma_{j-1}d_{j-2}) + (1 - \rho_j)q_{j-3}$  into  $c_{j-1} \leftarrow Mq_{j-1}$  and the new equation for  $c_{j-1}$  into  $d_{j-1} \leftarrow Ac_{j-1}$

to get

$c_{j-1} \leftarrow \rho_j(c_{j-2} - \gamma_{j-1}g_{j-1}) + (1 - \rho_j)c_{j-3}$  and  $g_{j-2} \leftarrow Md_{j-2}$  and  $d_{j-1} \leftarrow \rho_j(d_{j-2} - \gamma_{j-1}h_{j-1}) + (1 - \rho_j)d_{j-3}$  and  $h_{j-2} \leftarrow Ag_{j-2}$ .

We can then move the new matrix-vector multiply and preconditioner application into the previous iteration after the other matrix-vector multiply and preconditioner application. Since none of the dot products rely on the results of these kernels, we can now fully overlap both matrix-vector multiplies and preconditioner applications with a single non-blocking allreduce. We move the vector operations into a separate routine to simplify the main routine. The final version of PIPE2CG is shown in Algorithms 5.5 and 5.6.

---

**Algorithm 5.5** PIPELINED 2-Iteration PCG

---

**Input:**  $A$ :  $n \times n$  SPD matrix,  $x_0$ : initial guess,  $b$ : right-hand side,  $M$ : prec

**Output:**  $x_j$ : approximate solution

$r_0 \leftarrow b - Ax_0$     $z_0 \leftarrow Mr_0$     $w_0 \leftarrow Az_0$

$\delta_0 \leftarrow (z_0, w_0)$     $\beta_0 \leftarrow (z_0, r_0)$     $norm_0 \leftarrow \sqrt{(z_0, z_0)}$  ▷ 3 Merged Operations

MPI\_Allreduce on  $\delta_0, \beta_0, norm_0$

$p_0 \leftarrow Mw_0$     $q_0 \leftarrow Ap_0$  ▷ Overlap Comm and Comp

$c_0 \leftarrow Mq_0$     $d_0 \leftarrow Ac_0$

**for**  $j=1,3,\dots$ ,until convergence **do**

**if**  $j > 1$  **then**

$\rho_{j-1} \leftarrow 1/(1 - (\gamma_{j-1}\beta_{j-1})/(\gamma_{j-2}\beta_{j-2}\rho_{j-2}))$     $\gamma_{j-1} \leftarrow \beta_{j-2}/\delta_{j-2}$

$\phi \leftarrow [\rho_{j-1}, -\rho_{j-1}\gamma_{j-1}, (1 - \rho_{j-1})]$

$\delta_{j-1} \leftarrow \phi_0\phi_0\lambda_0 + 2\phi_0\phi_1\lambda_1 + 2\phi_0\phi_2\lambda_2 + \phi_1\phi_1\lambda_3 + 2\phi_1\phi_2\lambda_4 + \phi_2\phi_2\lambda_5$

$\beta_{j-1} \leftarrow \phi_0\phi_0\lambda_6 + 2\phi_0\phi_1\lambda_0 + 2\phi_0\phi_2\lambda_7 + \phi_1\phi_1\lambda_1 + 2\phi_1\phi_2\lambda_2 + \phi_2\phi_2\lambda_8$

$\rho_j \leftarrow 1/(1 - (\gamma_j\beta_{j-1})/(\gamma_{j-1}\beta_{j-2}\rho_{j-1}))$     $\gamma_j \leftarrow \beta_{j-1}/\delta_{j-1}$

**else**

$\rho_j \leftarrow 1$     $\gamma_j \leftarrow \beta_{j-1}/\delta_{j-1}$

  VecPipelined\_PIPE2CG()

MPI\_Allreduce on  $\lambda_0$  to  $\lambda_9$

$c_j \leftarrow Mq_j$     $d_j \leftarrow Ac_j$

▷ Overlap Comm and Comp

$g_j \leftarrow Md_j$     $h_j \leftarrow Ag_j$

---

**Algorithm 5.6** VecPipelined\_PIPE2CG

---

```

 $\mu_j \leftarrow 1 - \rho_j$        $\mu_{j-1} \leftarrow 1 - \rho_{j-1}$ 
if  $j > 1$  then
   $x_{j-1} \leftarrow \rho_{j-1}(x_{j-2} + \gamma_{j-1}z_{j-2}) + \mu_{j-1}x_{j-3}$ 
   $r_{j-1} \leftarrow \rho_{j-1}(r_{j-2} - \gamma_{j-1}w_{j-2}) + \mu_{j-1}r_{j-3}$ 
   $z_{j-1} \leftarrow \rho_{j-1}(z_{j-2} - \gamma_{j-1}p_{j-2}) + \mu_{j-1}z_{j-3}$ 
   $w_{j-1} \leftarrow \rho_{j-1}(w_{j-2} - \gamma_{j-1}q_{j-2}) + \mu_{j-1}w_{j-3}$ 
   $p_{j-1} \leftarrow \rho_{j-1}(p_{j-2} - \gamma_{j-1}c_{j-2}) + \mu_{j-1}p_{j-3}$ 
   $q_{j-1} \leftarrow \rho_{j-1}(q_{j-2} - \gamma_{j-1}d_{j-2}) + \mu_{j-1}q_{j-3}$ 
   $c_{j-1} \leftarrow \rho_{j-1}(c_{j-2} - \gamma_{j-1}g_{j-2}) + \mu_{j-1}c_{j-3}$ 
   $d_{j-1} \leftarrow \rho_{j-1}(d_{j-2} - \gamma_{j-1}h_{j-2}) + \mu_{j-1}d_{j-3}$ 
   $x_j \leftarrow \rho_j(x_{j-1} + \gamma_j z_{j-1}) + \mu_j x_{j-2}$ 
   $r_j \leftarrow \rho_j(r_{j-1} - \gamma_j w_{j-1}) + \mu_j r_{j-2}$ 
   $z_j \leftarrow \rho_j(z_{j-1} - \gamma_j p_{j-1}) + \mu_j z_{j-2}$ 
   $w_j \leftarrow \rho_j(w_{j-1} - \gamma_j q_{j-1}) + \mu_j w_{j-2}$ 
   $p_j \leftarrow \rho_j(p_{j-1} - \gamma_j c_{j-1}) + \mu_j p_{j-2}$ 
   $q_j \leftarrow \rho_j(q_{j-1} - \gamma_j d_{j-1}) + \mu_j q_{j-2}$ 
   $\lambda_0 \leftarrow (z_j, w_j)$        $\lambda_1 \leftarrow (z_j, q_j)$        $\lambda_2 \leftarrow (z_j, w_{j-1})$        $\lambda_3 \leftarrow (p_j, q_j)$ 
   $\lambda_4 \leftarrow (p_j, w_{j-1})$        $\lambda_5 \leftarrow (z_{j-1}, w_{j-1})$        $\lambda_6 \leftarrow (z_j, r_j)$        $\lambda_7 \leftarrow (z_j, r_{j-1})$ 
   $\lambda_8 \leftarrow (z_{j-1}, r_{j-1})$        $\lambda_9 \leftarrow (z_j, z_j)$ 
   $\delta_j \leftarrow \lambda_0$        $\beta_j \leftarrow \lambda_6$        $norm_j \leftarrow \sqrt{\lambda_9}$ 

```

---

▷ 8 Merged Operations

▷ 16 Merged Operations

**6. Conclusions.** We have shown detailed derivations of the PIPECG3 and PIPE2CG methods. These variations of the preconditioned conjugate gradient method are based on the three-term recurrence variation of PCG and allow us to develop the PIPE2CG method that overlaps 2-iterations of matrix-vector multiplies and preconditioner applications with a single non-blocking allreduce. This produces a more scalable PCG method that performs better as communication costs increase and work per core decreases. This approach provides the potential to develop further pipelined non-blocking PCG methods.

## REFERENCES

- [1] P. ELLER AND W. GROPP, Scalable Non-blocking Preconditioned Conjugate Gradient Methods, in Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC '16, ACM, Nov. 2016.
- [2] P. GHYSELS AND W. VANROOSE, Hiding Global Synchronization Latency in the Preconditioned Conjugate Gradient Algorithm, *Parallel Comput.*, 40 (2014), pp. 224–238.
- [3] M. H. GUTKNECHT AND Z. STRAKOS, Accuracy of Two Three-term and Three Two-term Recurrences for Krylov Space Solvers, *SIAM Journal on Matrix Analysis and Applications*, 22 (2000), pp. 213–229.
- [4] M. HESTENES AND E. STIEFEL, Methods of Conjugate Gradients for Solving Linear Systems, *Journal of Research of the National Bureau of Standards*, 49 (1952).
- [5] M. F. HOEMMEN, *Communication-avoiding Krylov subspace methods*, PhD thesis, EECS Department, University of California, Berkeley, Apr 2010.
- [6] Y. SAAD, *Iterative Methods for Sparse Linear Systems*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd ed., 2003.

# NUMERICAL METHODS FOR ORTHOGONAL SPLINE COLLOCATION DISCRETIZATIONS IN PDE CONSTRAINED OPTIMIZATION

NICHOLAS L. FISHER\* AND ERIC C. CYR†

**Abstract.** A new discretization for PDE constrained optimization based on piecewise Hermite cubic orthogonal spline collocation (OSC) is introduced. An optimal preconditioner for the MINRES iteration is also introduced, and the optimality of the preconditioner is proven theoretically and confirmed computationally.

**1. Introduction.** In [9], the authors present a method for PDE constrained optimization using the Galerkin finite element method to discretize the problem. Additionally, they conjecture that the overall method should be compatible with other discretization techniques. In what follows, we demonstrate how piecewise Hermite cubic OSC can be applied to some simple problems arising from PDE constrained optimization. Importantly, the linear system resulting from the OSC discretization has a saddle-point structure. Thus, the discrete version of the PDE optimization problem is readily solved via an iterative Krylov subspace method. To that end, a suitable preconditioner for the minimum residual (MINRES) method [8] is found and shown to be optimal. However, a complete study of OSC for PDE constrained optimization would require an analysis similar to that done in the functional setting for finite element discretizations. This effort is left for future work.

The rest of the paper is outlined as follows. In Section 2, the piecewise Hermite cubic and bicubic OSC methods are introduced. Section 3 outlines the formulation of the OSC method for PDE constrained optimization and discusses the preconditioning of the MINRES iteration. Finally, numerical results are presented in Section 4, and concluding remarks are given in Section 5.

**2. Piecewise Hermite Cubic and Bicubic OSC.** Orthogonal spline collocation methods have been widely used to solve boundary value problems [3]. In this paper we focus on the use of piecewise Hermite cubic basis functions as the spectral properties of the resulting collocation systems are well understood [2, 11], and it leads to a numerical method that is rather straightforward to implement.

**2.1. Piecewise Hermite Cubic OSC for Two-Point BVP .** We introduce the piecewise Hermite cubic OSC method by way of example. Consider the two-point boundary value problem:

$$-p(x)u'' + r(x)u' + q(x)u = f(x), \quad u(0) = u(1) = 0. \quad (2.1)$$

Let  $\{x_i\}_{i=0}^N$ , with  $N$  a positive integer, be a uniform partition of  $[0, 1]$ , i.e.,

$$x_k = kh, \quad k = 0, \dots, N \quad (2.2)$$

where the step size is  $h = 1/N$ . Define  $\mathcal{M}^0$  to be the space of piecewise Hermite cubics with the breaks at the  $x_i$ , that is,

$$\mathcal{M}^0 = \{v \in C^1[0, 1] : v|_{[x_{i-1}, x_i]} \in P_3, i = 1, \dots, N, v(0) = v(1) = 0\},$$

where  $P_3$  is the set of polynomials of degree  $\leq 3$ .

---

\*Department of Applied Mathematics and Statistics, Colorado School of Mines, nfisher@mines.edu

†Sandia National Laboratories, eccyr@sandia.gov

Let  $\{\zeta_k\}_{k=1}^2$  be the nodes of the 2-point Gauss-Legendre quadrature for  $(-1, 1)$ , and let  $\{\xi_m\}_{m=1}^{2N}$  be the set of the collocation points in  $(0, 1)$  defined by

$$\xi_{2n+1} = x_n + h\zeta_1, \quad \xi_{2n+2} = x_n + h\zeta_2, \quad n = 0, \dots, N-1. \quad (2.3)$$

Then, the OSC problem corresponding to (2.1) consists in finding  $U \in \mathcal{M}^0$  such that

$$-p(\xi_n)U''(\xi_n) + r(\xi_n)U'(\xi_n) + q(\xi_n)U(\xi_n) = f(\xi_n), \quad n = 1, \dots, 2N. \quad (2.4)$$

If  $\{\phi_j\}_{j=1}^{2N}$  is a basis for  $\mathcal{M}^0$ , then we can write

$$U = \sum_{j=1}^{2N} U_j \phi_j.$$

Thus (2.4) amounts to solving a system of linear equations for the coefficients  $\{U_j\}_{j=1}^{2N}$ . That is, for

$$[\bar{A}]_{i,j} = (-\phi_j''(\xi_i))_{i,j=1}^{2N}, \quad [\bar{B}]_{i,j} = (\phi_j(\xi_i))_{i,j=1}^{2N}, \quad (2.5)$$

$$[\bar{C}]_{i,j} = (\phi_j'(\xi_i))_{i,j=1}^{2N}$$

we have

$$(D(p)\bar{A} + D(r)\bar{C} + D(q)\bar{B}) \mathbf{u} = \mathbf{f},$$

where,

$$D(g) = \text{diag}(g(\xi_1), g(\xi_2), \dots, g(\xi_{2N}))$$

with the coefficients of the piecewise Hermite cubic spline interpolant,

$$\mathbf{u} = [U(x_1), U'(x_1)/h, \dots, U(x_N), U'(x_N)/h]^T,$$

and

$$\mathbf{f} = [f(\xi_1), f(\xi_2), \dots, f(\xi_{2N})]^T.$$

**2.2. Piecewise Hermite Bicubic OSC for Poisson's Equation.** The Dirichlet boundary value problem for Poisson's equation on the unit square is given by

$$-\nabla^2 u = f(x, y), \quad (x, y) \in \Omega, \quad u(x, y) = g(x, y), \quad (x, y) \in \partial\Omega \quad (2.6)$$

where  $\Omega = (0, 1) \times (0, 1)$ .

In what follows we discretize with respect to a uniform partition of  $\Omega$  for brevity. To that end, let  $\{y_i\}_{i=0}^N$  be a uniform partition of  $[0, 1]$  in the same manner as (2.2). Consequently, if  $g(x, y) = 0$  then collocation equations for (2.6) with respect to a uniform discretization of  $\Omega$  are given by

$$-\nabla^2 U(\xi) = f(\xi), \quad \xi \in \mathcal{G} \quad (2.7)$$

for  $U \in \mathcal{M}^0 \otimes \mathcal{M}^0$  and where  $\mathcal{G} = \{(x, y) : x, y \in \{\xi_m\}_{m=1}^{2N}\}$ . Assuming  $\{\phi_j\}_{j=1}^{2N}$  is a basis for  $\mathcal{M}^0$ , we can write the piecewise Hermite bicubic orthogonal collocation approximation to the solution of (2.6) as

$$U(x, y) = \sum_{i=1}^{2N} \sum_{j=1}^{2N} U_{i,j} \phi_i(x) \phi_j(y).$$

Hence, the corresponding matrix-vector form of (2.7) is

$$A\mathbf{u} = \mathbf{f} \quad (2.8)$$

where, if  $\bar{A}$  and  $\bar{B}$  are defined by (2.5),

$$A = \bar{A} \otimes \bar{B} + \bar{B} \otimes \bar{A}, \quad (2.9)$$

for  $\otimes$  denoting the tensor product, and

$$\mathbf{u} = [U_{1,1}, \dots, U_{1,2N}, \dots, U_{2N,1}, \dots, U_{2N,2N}]^T,$$

where the  $U_{i,j}$  are the coefficients of the piecewise Hermite bicubic spline interpolant, and with

$$\mathbf{f} = [f_{1,1}, \dots, f_{1,2N}, \dots, f_{2N,1}, \dots, f_{2N,2N}]^T, \quad f_{i,j} = f(\xi_i, \xi_j).$$

The piecewise Hermite bicubic orthogonal spline collocation approximation to Poisson's equation is presented with homogeneous Dirichlet boundary conditions for simplicity; however, in [3], the authors describe two methods for handling more general boundary conditions. For example, one may compute the piecewise cubic interpolant of  $u$  at the boundary Gauss points. Then, these coefficients can be moved to the right-hand side of (2.8) and the problem reduces to the homogeneous case.

### 3. PDE Constrained Optimization.

**3.1. Distributed Control of Poisson's Equation.** Consider the distributed control problem:

$$\min_{u,f} \frac{1}{2} \|u - \hat{u}\|_{L^2(\Omega)}^2 + \beta \|f\|_{L^2(\Omega)}^2, \quad \Omega \subset \mathbb{R}^2 \quad (3.1)$$

subject to

$$-\nabla^2 u(x, y) = f(x, y), \quad (x, y) \in \Omega \quad (3.2)$$

with

$$u(x, y) = g(x, y), \quad (x, y) \in \partial\Omega. \quad (3.3)$$

The desired state  $\hat{u}$  is known, while  $u$  and  $f$ , satisfying the PDE, are allowed to vary in order to minimize the cost functional (3.1). The second term in (3.1) is the Tikhonov penalty term; it ensures the well posedness of the problem. The Tikhonov parameter  $\beta$  is often taken to be  $10^{-2}$  [9].

**3.1.1. Discretization by OSC.** Next, for  $U, f \in \mathcal{M}^0 \otimes \mathcal{M}^0$ , we consider the collocation equations for (3.1) - (3.3) with  $g = 0$  and  $\Omega = [0, 1] \times [0, 1]$ :

$$\min_{U,f} \frac{1}{2} \|U(\xi) - \hat{u}(\xi)\|_2^2 + \beta \|f(\xi)\|_2^2, \quad \xi \in \mathcal{G} \quad (3.4)$$

such that

$$-\nabla^2 U(\xi) = f(\xi), \quad \xi \in \mathcal{G}. \quad (3.5)$$

The discrete cost functional can now be written,

$$\min_{U, f} \frac{1}{2} \|U(\xi) - \widehat{u}(\xi)\|_2^2 + \beta \|f(\xi)\|_2^2 = \min_{\mathbf{u}, \mathbf{f}} \frac{1}{2} \mathbf{u} B^T B \mathbf{u}^T - \mathbf{u}^T B^T \mathbf{b} + \alpha + \beta \mathbf{f}^T \mathbf{f} \quad (3.6)$$

with  $\mathbf{u}$  and  $\mathbf{f}$  as defined in Section 2.2,  $\mathbf{b} = (\widehat{u}(\xi_1, \xi_1), \dots, \widehat{u}(\xi_{2N}, \xi_{2N}))^T$ , and where  $\alpha = \|\mathbf{b}\|_2^2$  and for  $\bar{B}$  given in (2.5),

$$B = \bar{B} \otimes \bar{B}. \quad (3.7)$$

Note that the constraint in (3.5) is equivalent to (2.8). Hence, (3.4) and (3.5) are equivalent to (3.6) and (2.8). The minimization problem may now be solved by considering the Lagrangian

$$\mathcal{L} := \frac{1}{2} \mathbf{u}^T B^T B \mathbf{u}^T - \mathbf{u}^T B^T \mathbf{b} + \alpha + \beta \mathbf{f}^T \mathbf{f} + \lambda^T (A \mathbf{u} - \mathbf{f}). \quad (3.8)$$

Using the stationarity conditions of  $\mathcal{L}$ , it can be shown that  $\mathbf{f}$ ,  $\mathbf{u}$ , and  $\lambda$  are defined by the linear system,

$$\underbrace{\begin{bmatrix} 2\beta I & 0 & -I \\ 0 & B^T B & A^T \\ -I & A & 0 \end{bmatrix}}_{\mathcal{A}} \begin{bmatrix} \mathbf{f} \\ \mathbf{u} \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ B^T \mathbf{b} \\ 0 \end{bmatrix}. \quad (3.9)$$

Importantly, this system has saddle-point structure. That is, (3.9) has the form

$$\begin{bmatrix} M & K^T \\ K & -L \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} = \begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} \quad (3.10)$$

where  $M = \begin{bmatrix} 2\beta I & 0 \\ 0 & B^T B \end{bmatrix}$ ,  $K = [-I \ A]$ ,  $L = 0$ .

In general, the system given in (3.9) is very large; however, the Hermite cubic OSC discretization generates sparse matrices of almost block diagonal form. Additionally, (3.9) is symmetric, thus the preconditioned MINRES method is applicable.

**3.1.2. Preconditioned MINRES.** Following [9] and [6], we will investigate the preconditioner of the form

$$\mathcal{P}_1 = \begin{bmatrix} 2\beta I & 0 & 0 \\ 0 & B^T B & 0 \\ 0 & 0 & \frac{1}{2\beta} I + A(B^T B)^{-1} A^T \end{bmatrix}. \quad (3.11)$$

This preconditioner can be approximated (for  $\beta$  not “too” small) by

$$\mathcal{P}_2 = \begin{bmatrix} 2\beta I & 0 & 0 \\ 0 & B^T B & 0 \\ 0 & 0 & A(B^T B)^{-1} A^T \end{bmatrix}. \quad (3.12)$$

**THEOREM 3.1.** *Let  $\lambda$  be an eigenvalue of  $\mathcal{P}_2^{-1} \mathcal{A}$  with  $\mathcal{P}_2$  defined by (3.12) and  $\mathcal{A}$  defined by (3.9). Then either  $\lambda = 1$ ,*

$$\frac{1}{2} (1 + \sqrt{1 + 4\sigma_1}) \leq \lambda \leq \frac{1}{2} (1 + \sqrt{1 + 4\sigma_m})$$



or,

$$\frac{1}{2} (1 - \sqrt{1 + 4\sigma_m}) \leq \lambda \leq \frac{1}{2} (1 - \sqrt{1 + 4\sigma_1})$$

where  $0 \leq \sigma_1 \leq \dots \leq \sigma_m$  are the eigenvalues of  $I + \frac{1}{2\beta} A^{-T} (B^T B) A^{-1}$ .

*Proof.* By similarity transformation, the eigenvalues of  $\mathcal{P}_2^{-1} \mathcal{A}$  are identical to the eigenvalues of  $\tilde{\mathcal{A}} := \mathcal{P}_2^{-1/2} \mathcal{A} \mathcal{P}_2^{-1/2}$ . By simple calculation, we have

$$\tilde{\mathcal{A}} = \begin{bmatrix} I & 0 & \tilde{K}_1 \\ 0 & I & \tilde{K}_2 \\ \tilde{K}_1 & \tilde{K}_2 & 0 \end{bmatrix}$$

where  $\tilde{K}_1 = \frac{1}{\sqrt{2\beta}} (A(B^T B)^{-1} A^T)^{-1/2}$ ,  $\tilde{K}_2 = (A(B^T B)^{-1} A^T)^{-1/2} A(B^T B)^{-1/2}$ .

The rest of the proof follows that of Proposition 2 in [9].  $\square$

LEMMA 3.2. For  $\bar{A}$  and  $\bar{B}$  given by (2.5), and for  $A$  and  $B$  defined by (2.9) and (3.7), respectively, we have the following:

1. The eigenvalues of the generalized eigenvalue problem

$$\bar{A}\mathbf{x} = \lambda \bar{B}\mathbf{x} \tag{3.13}$$

satisfy the inequality:

$$\pi^2 \leq \lambda \leq \frac{36}{h^2}.$$

2. The eigenvalues of the generalized eigenvalue problem

$$A\mathbf{x} = \lambda B\mathbf{x} \tag{3.14}$$

satisfy the inequality:

$$2\pi^2 \leq \lambda \leq \frac{72}{h^2}.$$

*Proof.* From Theorem 2.1 in [2], the eigenvalue problem (3.13) has  $2N$  distinct eigenvalues given by

$$\lambda_j^\pm = \frac{72 \sin^2 \left( \frac{j\pi}{2N} \right)}{h^2 \gamma_j^\pm}, \quad j \in \mathcal{Q}, \quad \lambda_0 = \frac{36}{h^2}, \quad \lambda_N = \frac{12}{h^2} \tag{3.15}$$

where

$$\gamma_j^\pm = 8 + \eta_j \mp \sqrt{43 + 40\eta_j - 2\eta_j^2}, \quad \eta_j = \cos \left( \frac{j\pi}{N} \right), \quad j \in \mathcal{Q} \tag{3.16}$$

with  $\mathcal{Q} = \{1, 2, \dots, N-1\}$ . Moreover, it is known [1] that the eigenvalues of (3.13) satisfy the inequality  $\pi^2 \leq \lambda \leq \frac{36}{h^2}$ .

To prove the second part of the lemma, consider that  $\bar{B}$  is invertible, hence,  $B$  is invertible. Thus,

$$\begin{aligned} A\mathbf{x} &= \lambda B\mathbf{x} \\ (\bar{A} \otimes \bar{B} + \bar{B} \otimes \bar{A})\mathbf{x} &= \lambda (\bar{B} \otimes \bar{B})\mathbf{x} \\ (\bar{B}^{-1} \bar{A} \otimes I + I \otimes \bar{B}^{-1} \bar{A})\mathbf{x} &= \lambda \mathbf{x} \end{aligned}$$

We know that if  $\lambda$  satisfies (3.13), then  $\lambda$  is an eigenvalue of the matrix  $\bar{B}^{-1}\bar{A}$ . Thus, application of some basic properties of the tensor product will complete the proof. Specifically, by Theorem 13.16 in [5], it follows that the  $4N^2$  eigenvalues of  $\bar{B}^{-1}\bar{A} \otimes I + I \otimes \bar{B}^{-1}\bar{A}$  are

$$\underbrace{2\lambda_1, \dots, 2\lambda_1}_{2N \text{ terms}}, \underbrace{2\lambda_2, \dots, 2\lambda_2}_{2N \text{ terms}}, \dots, \underbrace{2\lambda_{2N}, \dots, 2\lambda_{2N}}_{2N \text{ terms}} \quad (3.17)$$

where,  $\lambda_1, \dots, \lambda_{2N}$  are the eigenvalues of  $\bar{B}^{-1}\bar{A}$ . Thus, 3.17 and part (1) of the lemma yield the desired result.  $\square$

**COROLLARY 3.3.** *Let  $\lambda$  be an eigenvalue of  $\mathcal{P}_2^{-1}\mathcal{A}$ , with  $\mathcal{P}_2$  and  $\mathcal{A}$  defined in Theorem 3.1 and with  $A$  and  $B$  defined in Lemma 3.2. Then  $\lambda$  satisfies one of*

$$\lambda = 1,$$

$$\frac{1}{2} \left( 1 + \sqrt{5 + \frac{h^4}{2 \cdot 36^2 \beta}} \right) \leq \lambda \leq \frac{1}{2} \left( 1 + \sqrt{5 + \frac{1}{2\pi^4 \beta}} \right)$$

or,

$$\frac{1}{2} \left( 1 - \sqrt{5 + \frac{1}{2\pi^4 \beta}} \right) \leq \lambda \leq \frac{1}{2} \left( 1 - \sqrt{5 + \frac{h^4}{2 \cdot 36^2 \beta}} \right).$$

*Proof.* From Theorem 3.1, we know that the clustering of the eigenvalues of the preconditioned systems depends of finding the eigenvalues of the matrix  $T := I + \frac{1}{2\beta} A^{-T} (B^T B) A^{-1}$ . Thus, if  $\lambda$  is an eigenvalue of  $T$ , then

$$\begin{aligned} T\mathbf{x} &= \lambda\mathbf{x} \\ \left( \frac{1}{2\beta} A^{-T} (B^T B) A^{-1} + I \right) \mathbf{x} &= \lambda\mathbf{x} \\ (AB^{-1})^{-T} (AB^{-1})^{-1} \mathbf{x} &= 2\beta(\lambda - 1)\mathbf{x} \end{aligned}$$

Next, it follows from part (2) of Lemma 3.2 that if  $\gamma$  is an eigenvalue of the matrix  $(AB^{-1})^{-1}$  then  $\frac{h^2}{72} \leq \gamma \leq \frac{1}{2\pi^2}$ . Hence

$$\frac{h^4}{8 \cdot 36^2 \beta} + 1 \leq \lambda \leq \frac{1}{8\pi^2 \beta} + 1. \quad (3.18)$$

The desired result follows from (3.18) and Theorem 3.1.  $\square$

Note that the bounds of Corollary 3.3 depend on  $h$  in such a way that the eigenvalues of  $T$  remain bounded away from 0 as  $h \rightarrow 0$ . This suggests that the preconditioner  $P_2$  is optimal, i.e., its performance is independent of mesh size. A comparison of the theoretical bounds to the computed eigenvalues of  $\mathcal{P}_2^{-1}\mathcal{A}$  for two different values of  $\beta$  can be seen in Figures 3.1.(a) and 3.1.(b).

**3.2. Distributed Control of Burgers' Equation.** We consider the optimal control problem with the steady state Burgers' equation in one spatial dimension acting as the constraint. That is, for  $\Omega = [0, 1]$ , and  $\partial\Omega = \{0, 1\}$  we have

$$\min_{u, g} \frac{1}{2} \|u - \hat{u}\|_{L_2(\Omega)}^2 + \frac{\alpha}{2} \|g\|_{L_2(\Omega)}^2 \quad (3.19)$$

$$\text{s.t. } -\nu u_{xx} + uu_x = g \quad \text{in } \Omega \quad (3.20)$$

$$u = 0 \quad \text{on } \partial\Omega. \quad (3.21)$$

As before, the desired state  $\hat{u}$  is known,  $u$  is the state variable, and  $g$  is the control variable. Additionally,  $\nu$  is the viscosity and  $\alpha$  plays the role of the penalty parameter.

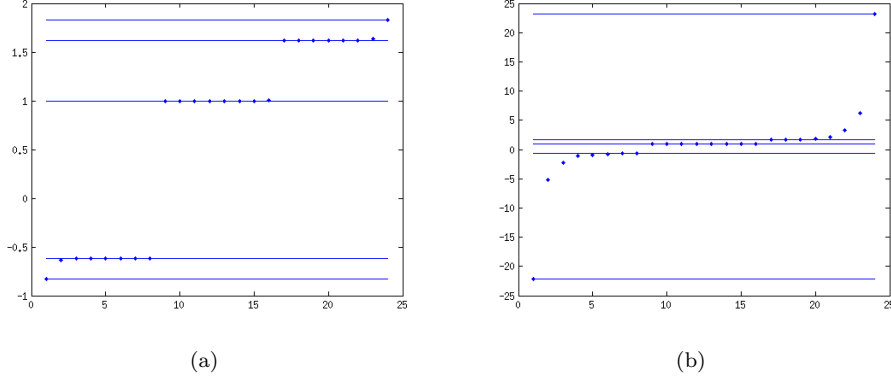


Fig. 3.1: Eigenvalues of  $\mathcal{P}_2^{-1}\mathcal{A}$  and the theoretical bounds given by Corollary 3.3 (lines are the bounds,  $\cdot$ 's are the eigenvalues).  $\beta = 10^{-2}$  and  $\beta = 10^{-5}$  in (a) and (b), respectively.

**3.2.1. A Local Newton-Lagrange Method.** Next, we consider the equality constrained optimization problem:

$$\begin{aligned} \min f(\mathbf{x}) \\ \text{s.t. } h(\mathbf{x}) = 0 \end{aligned} \quad (3.22)$$

where  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  and  $h : \mathbb{R}^n \rightarrow \mathbb{R}^m$  are sufficiently smooth functions. A standard approach [7] for solving non-linear optimization problems is to apply Newton's method to the Karush-Kuhn-Tucker (KKT) optimality conditions [7] for (3.22).

Recall that the Lagrangian function for (3.22) is  $\mathcal{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda^T h(\mathbf{x})$ . Then the first order KKT conditions for (3.22) can be written as

$$F(\mathbf{x}, \lambda) = \begin{bmatrix} \nabla \mathcal{L}(\mathbf{x}, \lambda) \\ h(\mathbf{x}) \end{bmatrix} = 0. \quad (3.23)$$

The non-linear system of equations (3.23) can now be solved by Newton's method. To that end, note that the Jacobian of (3.23) is given by

$$JF(\mathbf{x}, \lambda) = \begin{bmatrix} H\mathcal{L}(\mathbf{x}, \lambda) & Jh(\mathbf{x})^T \\ Jh(\mathbf{x}) & 0 \end{bmatrix} \quad (3.24)$$

where  $H$  and  $J$  denote the Hessian and Jacobian, respectively.

Hence, we arrive at the Newton-KKT system,

$$\begin{bmatrix} H\mathcal{L}(\mathbf{x}^k, \lambda^k) & Jh(\mathbf{x}^k)^T \\ Jh(\mathbf{x}^k) & 0 \end{bmatrix} \begin{bmatrix} \delta \mathbf{x} \\ \delta \lambda \end{bmatrix} = - \begin{bmatrix} \nabla \mathcal{L}(\mathbf{x}^k, \lambda^k) \\ h(\mathbf{x}^k) \end{bmatrix} \quad (3.25)$$

where the Newton step is given by,

$$\begin{aligned} \mathbf{x}^{k+1} &= \mathbf{x}^k + \delta \mathbf{x}, \\ \lambda^{k+1} &= \lambda^k + \delta \lambda. \end{aligned} \quad (3.26)$$

Thus, the Newton-Lagrange algorithm is: solve (3.25), update using (3.26) and repeat until convergence. It is worth noting that this method may not converge if the initial guess for  $\mathbf{x}$  is not "close enough" to the optimal solution  $\mathbf{x}^*$ .

**3.2.2. Problem Discretization.** We discretize the problem using piecewise Hermite cubic orthogonal spline collocation. To that end, consider the collocation equations of (3.20)-(3.21): find  $U \in M^0$  such that

$$-\nu U_{xx}(\xi) + U(\xi)U_x(\xi) = g(\xi), \quad \xi \in \mathcal{G}. \quad (3.27)$$

Assume that  $\{\phi_1, \dots, \phi_{2N}\}$  is a basis for  $\mathcal{M}^0$ , then we can write

$$U = \sum_{j=1}^{2N} U_j \phi_j.$$

Hence, for the matrices  $\bar{A}$ ,  $\bar{B}$ , and  $\bar{C}$  defined in (2.5), this yields the matrix-vector form of (3.27),

$$\nu \bar{A} \mathbf{u} + N(\mathbf{u}) = \mathbf{g} \quad (3.28)$$

where  $N(\mathbf{u}) = \bar{B} \mathbf{u} \circ \bar{C} \mathbf{u}$  and,  $\circ$ , denotes the element-wise product of two vectors. Thus we arrive at the discrete form of the Lagrangian,

$$\mathcal{L}(\mathbf{g}, \mathbf{u}, \lambda) = \frac{1}{2} \mathbf{u}^T \bar{B}^T \bar{B} \mathbf{u} - \mathbf{u}^T \bar{B}^T \mathbf{b} + \gamma + \frac{\alpha}{2} \mathbf{g}^T \mathbf{g} + \lambda^T (\nu \bar{A} \mathbf{u} + N(\mathbf{u}) - \mathbf{g}). \quad (3.29)$$

Now we can write (3.25) for the problem (3.19) - (3.21) with the discrete Lagrangian given by (3.29) and the discrete constraint given by (3.28). Specifically,

$$\nabla \mathcal{L}(\mathbf{g}, \mathbf{u}, \lambda) = \begin{bmatrix} \bar{B}^T \bar{B} \mathbf{u} - \bar{B}^T \mathbf{b} + \nu \bar{A}^T \lambda + N'(\mathbf{u})^T \lambda \\ \alpha \mathbf{g} - \lambda \end{bmatrix}, \quad (3.30)$$

$$h(\mathbf{g}, \mathbf{u}) = \nu \bar{A} \mathbf{u} + N(\mathbf{u}) - \mathbf{g}, \quad (3.31)$$

$$Jh(\mathbf{g}, \mathbf{u}) = \begin{bmatrix} -I & \nu \bar{A} + N'(\mathbf{u}) \end{bmatrix}, \quad (3.32)$$

with,

$$N'(\mathbf{u}) = \bar{B} \text{diag}(\bar{C} \mathbf{u}) + \text{diag}(\bar{B} \mathbf{u}) \bar{C}. \quad (3.33)$$

Next, the Hessian can be approximated by ignoring the contribution from the nonlinear term. Namely,

$$H\mathcal{L}(\mathbf{g}, \mathbf{u}, \lambda) \approx \begin{bmatrix} \alpha I & 0 \\ 0 & \bar{B}^T \bar{B} \end{bmatrix}. \quad (3.34)$$

This may affect the convergence of the overall algorithm, yet numerical experiments suggest that this approximation still yields good results.

**3.2.3. Preconditioned MINRES for the Newton-Lagrange Iteration.** The approximate Hessian Newton-Lagrange iteration is given by the linear system,

$$\underbrace{\begin{bmatrix} \alpha I & 0 & -I \\ 0 & \bar{B}^T \bar{B} & \tilde{A}(\mathbf{u}^k)^T \\ -I & \tilde{A}(\mathbf{u}^k) & 0 \end{bmatrix}}_{\mathcal{A}_B} \begin{bmatrix} \delta \mathbf{g} \\ \delta \mathbf{u} \\ \delta \lambda \end{bmatrix} = - \begin{bmatrix} \nabla \mathcal{L}(\mathbf{g}^k, \mathbf{u}^k, \lambda^k) \\ h(\mathbf{g}^k, \mathbf{u}^k) \end{bmatrix} \quad (3.35)$$

where  $\tilde{A}(\mathbf{u}^k) = \nu \bar{A} + N'(\mathbf{u}^k)$ . Clearly,  $\mathcal{A}_B$  is another saddle point system. Consequently, each step of the Newton-Lagrange iteration may be solved by preconditioned MINRES. The natural choice for a block diagonal preconditioner is the following,

$$\mathcal{P}_B = \begin{bmatrix} \alpha I & 0 & 0 \\ 0 & \bar{B}^T \bar{B} & 0 \\ 0 & 0 & \tilde{A}(\mathbf{u}^k)(\bar{B}^T \bar{B})^{-1} \tilde{A}(\mathbf{u}^k)^T \end{bmatrix}. \quad (3.36)$$

**4. Numerical Experiments.** We test our methods using some simple examples. First we seek to demonstrate the optimality of the preconditioning approach outlined in Section 3.1.2. Next, we show the results of the Newton-Lagrange method applied to the distributed control of Burgers' equation.

**4.1. Distributed Control of Poisson's Equation.** The first problem we address is modified from Example 10 in [9].

EXAMPLE 1. Let  $\Omega = [0, 1]^m$ , where  $m = 1, 2$ , and consider the problem

$$\min_{u, f} \frac{1}{2} \|u - \hat{u}\|_{L^2(\Omega)}^2 + \beta \|f\|_{L^2(\Omega)}^2, \quad (4.1)$$

subject to

$$-\nabla^2 u = f \quad \text{in } \Omega \quad (4.2)$$

with

$$u = 0 \quad \text{on } \partial\Omega. \quad (4.3)$$

where, in 1D,

$$\hat{u} = \exp(-64(x - 0.5)^2)$$

and, in 2D,

$$\hat{u} = \exp(-64((x - 0.5)^2 + (y - 0.5)^2)).$$

In Figures 4.1 and 4.2, the number of iterations for MINRES to converge is plotted against the stepsize  $h$  for different values of  $\beta$  in 1 and 2 dimensions, respectively. It is clear that, for larger values of  $\beta$ , the number of iterations for MINRES to converge remains constant as  $h$  decreases.

**4.2. Distributed Control of Burgers' Equation.** We follow [10], in our setup of the distributed control of Burgers' equation.

EXAMPLE 2. Let  $\Omega = [0, 1]$ , and consider

$$\min_{u, f} \frac{1}{2} \|u - \hat{u}\|_{L^2(\Omega)}^2 + \frac{\alpha}{2} \|g\|_{L^2(\Omega)}^2, \quad (4.4)$$

subject to

$$-\nu u_{xx} + uu_x = g \quad \text{in } \Omega \quad (4.5)$$

with

$$u = 0 \quad \text{on } \partial\Omega. \quad (4.6)$$

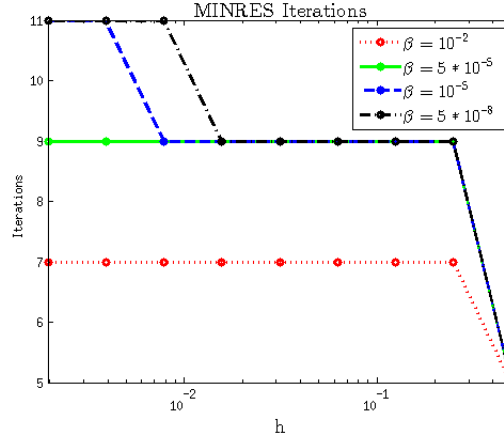


Fig. 4.1: Number of iterations for MINRES to solve Example 1 in 1D to a tolerance of  $10^{-6}$  for  $\beta = 10^{-2}$ ,  $\beta = 5 \times 10^{-5}$ ,  $\beta = 10^{-5}$  and  $\beta = 5 \times 10^{-8}$ .

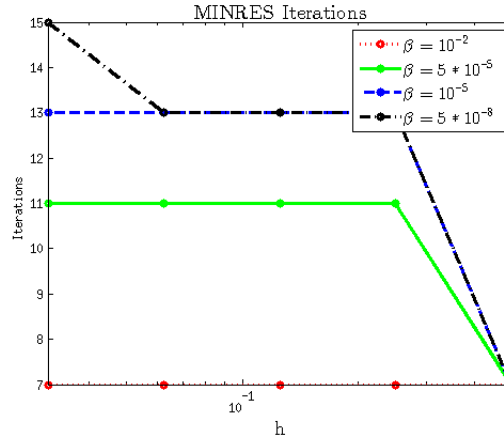


Fig. 4.2: Number of iterations for MINRES to solve Example 1 in 2D to a tolerance of  $10^{-6}$  for  $\beta = 10^{-2}$ ,  $\beta = 5 \times 10^{-5}$ ,  $\beta = 10^{-5}$  and  $\beta = 5 \times 10^{-8}$ .

where,

$$\hat{u} = \hat{u}(x) = \sin(2\pi x).$$

To demonstrate results for the distributed control of Burgers' equation, a uniform partition of  $\Omega$  with 100 subintervals is used to discretize Example 2. The Newton-Lagrange algorithm, is initialized with the vectors  $\lambda^0 = \mathbf{u}^0 = \mathbf{g}^0 = 0$ . The algorithm is terminated when both  $\|\nabla \mathcal{L}(\mathbf{u}, \mathbf{g}, \lambda)\|_2 < tol_{Newton}$  and  $\|h(\mathbf{u}, \mathbf{g})\|_2 < tol_{Newton}$ . For this example, we choose  $tol_{Newton} = 10^{-6}$ . In Figure 4.3, typical examples of the control and state solution are shown with  $\alpha = 10^{-5}$  and  $\nu = 10^{-2}$ .

Finally, we test how well MINRES performs across Newton iterations. Again, we choose

$\lambda^0 = \mathbf{u}^0 = \mathbf{g}^0 = 0$  and terminate the algorithm when the norm of both constraint and the gradient of the Lagrangian are less than  $tol_{Newton} = 10^{-4}$ . In addition, we set a tolerance for MINRES,  $tol_{MINRES} = 10^{-8}$ , and terminate the Newton method if the iteration count exceeds  $i_{max}^{Newton} = 25$ . Similarly, MINRES is terminated if the iteration count exceeds  $i_{max}^{MINRES} = 100$ . A uniform partition of  $\Omega$  with 64 subintervals is used, and the algorithm is tested for several different values of  $\alpha$  and  $\nu$ . Specifically, for each penalty parameter  $\alpha = 1, 0.01, 0.0001$  the algorithm is run with the viscosities  $\nu = 1, 0.1, 0.01, 0.001$ . The results of the numerical experiments are shown in Figure 4.4.

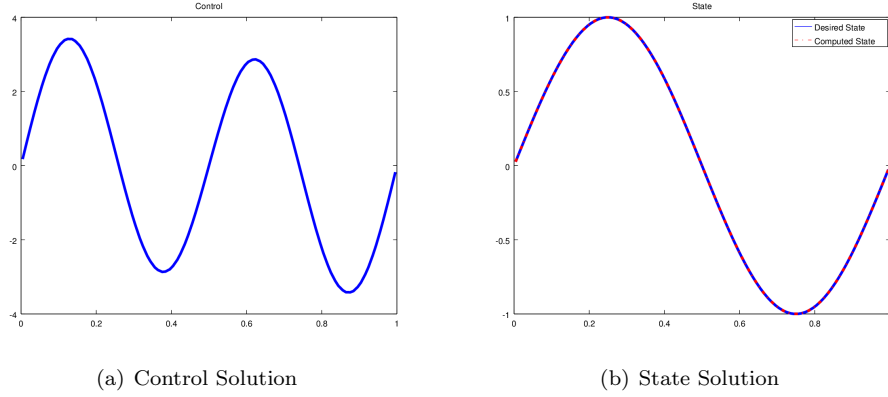


Fig. 4.3: Distributed control of Burgers' equation where  $\alpha = 10^{-5}$  and  $\nu = 10^{-2}$ .

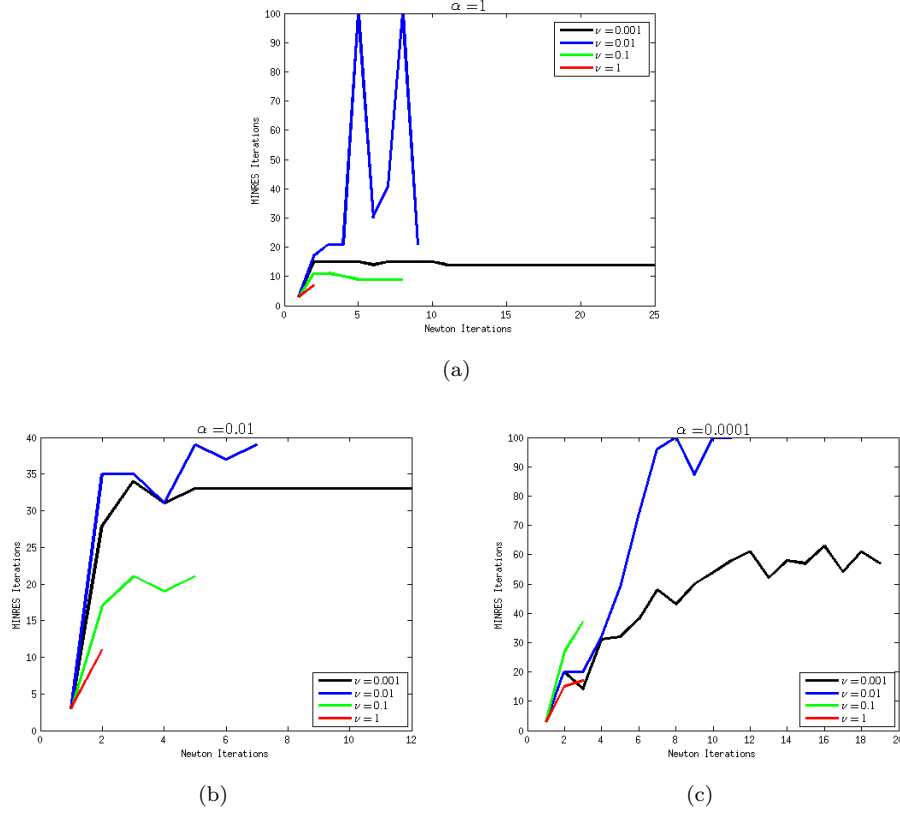


Fig. 4.4: Performance of MINRES across Newton iterations for the distributed control of Burgers' equation.

**5. Conclusions.** It has been shown how one can construct numerical methods for the solution of PDE constrained optimization problems based on piecewise Hermite cubic orthogonal spline collocation and Newton's method. In the linear case, the corresponding block diagonal preconditioner was shown to be optimal, both theoretically and computationally. Multigrid methods have been explored for collocation methods [4], thus it may be possible to accelerate the convergence of MINRES, and thus make the overall method more competitive. Additionally, spectral analysis of more general collocation systems has been conducted [11] indicating that the OSC method may be applicable to a broader class of PDE constraints while still maintaining the optimality of the preconditioner for the MINRES iteration. Finally, we note that, in this manuscript, the OSC discretization is applied without any functional-analytic considerations. Such considerations are left for future work.

#### REFERENCES

- [1] B. BIALECKI, An Alternating Direction Implicit Method for Orthogonal Spline Collocation Linear Systems, *Numer. Math.*, 59 (1991).
- [2] ———, Superconvergence of the Orthogonal Spline Collocation Solution of Poisson's Equation, *Num. Meth. for PDE's*, 15 (1999).
- [3] B. BIALECKI AND X. C. CAI, H1-Norm Error Bounds for Piecewise Hermite Bicubic Orthogonal Spline



- Collocation Schemes for Elliptic Boundary Value Problems, SIAM J. Numer. Anal., 31 (1994).
- [4] J. GARY, The Multigrid Iteration Applied to the Collocation Method, SIAM J. Numer. Anal., 18 (1981).
  - [5] A. LAUB, *Matrix Analysis for Scientists and Engineers*, SIAM, 2005.
  - [6] M. F. MURPHY, G. H. GOLUB, AND A. J. WATHEN, A Note on Preconditioning for Indefinite Linear Systems, SIAM J. Sci. Comput, 21, pp. 1969–1972.
  - [7] J. NOCEDAL AND S. WRIGHT, *Numerical Optimization*, Springer, 2006.
  - [8] C. C. PAIGE AND M. A. SAUNDERS, Solution of Sparse Indefinite Systems of Linear Equations, SIAM J. Numer. Anal., 12 (1975).
  - [9] T. REES, H. S. DOLLAR, AND A. J. WATHEN, Optimal Solvers for PDE Constrained Optimization, Tech. Rep. 08/10, Oxford University Computing Laboratory, 1999.
  - [10] D. RIDZAL, M. A. AGUILO, AND M. HEINKENSCHLOSS, Numerical Study of a Matrix-Free Trust-Region SQP Method for Equality Constrained Optimization, Tech. Rep. SAND2011-9346, Sandia National Laboratories, 2011.
  - [11] W. SUN, Spectral Analysis of Hermite Cubic Spline Collocation Systems, SIAM J. Numer. Anal., 36 (1999).

## INCREASING CONCURRENCY IN TWO-LEVEL SCHWARZ PRECONDITIONERS VIA ADDITIVE VARIANTS

MASSIMILIANO LUPO PASINI\*, RAYMOND S. TUMINARO<sup>†</sup>, AND JONATHAN HU<sup>‡</sup>

**Abstract.** Geometric multigrid (GMG) and domain decomposition (DD) are effective preconditioners for many partial differential equation applications. In this paper multiplicative and additive variants are formulated to increase concurrency on high performance computing architectures. Multiplicative methods are preferred from a convergence perspective, while additive variants can provide increased potential concurrency with the possible drawback of a deteriorated convergence rate.

In this paper we investigate a recently-formulated additive scheme that can obtain convergence comparable with multiplicative approaches at a computational cost comparable with additive approaches. Specifically, we consider an additive multigrid approach proposed by Vassilevski and Meier Yang in [20] that we adapt to a domain decomposition setting.

**1. Introduction.** Geometric multigrid and domain decomposition are highly successful linear solver algorithms for matrices that arise from elliptic partial differential equations (PDEs). Their success is based on the idea that simple iterative methods effectively reduce high frequency error components but have poor convergence rates on large wavelength (small frequency) components of the error, which then asymptotically dominate the overall convergence rate. More specifically, high frequency components of the error are reduced quickly using an inexpensive iterative approach on the fine mesh, whereas low frequency components are harder to dampen. Therefore, a way to accelerate the overall convergence is to reinterpret low frequency terms on a fine mesh as high frequency terms on a coarse mesh. To do this, a correction equation is projected to a coarse mesh after first applying a simple iterative technique. When the idea is applied recursively (to approximately solve the coarse mesh problem), a hierarchy of meshes of different resolutions results, and each grid has a version of the same PDE operator [12, 13, 16, 21].

The domain decomposition idea is to replace an original PDE problem defined on a large domain with multiple PDE problems defined on smaller regions [17, 18]. Different options to choose subdomains and boundary conditions of the surrogate problems are viable, but they must satisfy compatibility criteria with the original problem. Replacing a large problem with smaller ones is appealing as the smaller problems can be addressed concurrently. DD methods often include a coarse grid phase (similar to GMG) in addition to the local subdomain solves. In this way, DD methods can be derived so that convergence rates do not deteriorate significantly as the mesh is refined or the number of subdomains increases. A coarse solve is typically performed in a multiplicative fashion. That is, the subdomain solves are performed simultaneously, residuals are then computed followed by the coarse solve. The size of this coarse level problem is typically on the same order as the number of subdomains.

While multigrid and domain decomposition are now quite mature, new computing architectures give rise to new challenges, as communication complexity increases and data/hardware contention can impact heavily the performance of current techniques on future machines. In the context of multilevel solvers, coarse level processing can lead to parallel inefficiencies due to the sequential nature in which grids are processed and the fact that less concurrency is possible when processing coarse meshes. These issues motivate investigating a more efficient way to compute multigrid and domain decomposition preconditioners, aiming to preserve simultaneously convergence rates of multiplicative approaches and high

---

\*Emory University, massimiliano.lupo.pasini@emory.edu

<sup>†</sup>Sandia National Laboratories, rstumin@sandia.gov

<sup>‡</sup>Sandia National Laboratories, jhu@sandia.gov

concurrency of additive ones. If possible, this would open a path to exploit fully the potential offered by upcoming architectures. The new proposed DD approach is based on a technique already tested in the context of algebraic multigrid (AMG) in [20].

**2. Mathematical framework.** The problems analyzed in this paper are elliptic partial differential equations of the general form

$$\begin{cases} \mathcal{L}(u) = -\nabla \cdot (a(\mathbf{x})\nabla u) + \mathbf{b}(\mathbf{x}) \cdot \nabla u + c(\mathbf{x})u = f & \text{in } \Omega \\ u = g & \text{on } \partial\Omega \end{cases} \quad (2.1)$$

where  $\Omega \subset \mathbb{R}^d$  ( $d = 1, 2, 3$ ) is an open and convex set. The functions  $a(\mathbf{x})$ ,  $\mathbf{b}(\mathbf{x})$ ,  $c(\mathbf{x})$ ,  $f$  and  $g$  are given and one seeks to determine  $u$ .

Assuming the PDE problem is well posed [14] and that it is recast in finite dimension with a discretization technique [15] (such as Galerkin finite elements, finite differences or finite volumes), the discrete analogue to Eq. 2.1 is the linear system

$$A_h \mathbf{u}_h = \mathbf{f}_h, \quad (2.2)$$

where  $A_h \in \mathbb{R}^{n \times n}$  is the discretization of the differential operator  $\mathcal{L}$ ,  $\mathbf{u}_h$  is the discretization of the solution  $u$ ,  $\mathbf{f}_h \in \mathbb{R}^n$  is the discretization of the forcing term  $f$  and  $n$  is the number of degrees of freedom. The mesh spacing between adjacent grid points associated with Eq. 2.2 is denoted by  $h$ . We also introduce the discretization of the problem on a coarser grid with a mesh spacing  $H$ :

$$A_H \mathbf{u}_H = \mathbf{f}_H, \quad (2.3)$$

where  $A_H \in \mathbb{R}^{N \times N}$ ,  $\mathbf{u}_H \in \mathbb{R}^N$ ,  $\mathbf{f}_H \in \mathbb{R}^N$  and  $N$  is the number of degrees of freedom for the coarser mesh.

**3. Geometric multigrid setting.** Multigrid algorithms exploit the solution to the same problem associated with different grid refinements to effectively dampen different frequencies of the error [13, 21]. Restriction and interpolation operators are necessary to relate vectors associated with different mesh refinements. To this goal, let us define an interpolation operator  $I_H^h \in \mathbb{R}^{n \times N}$  and a restriction operator  $I_h^H \in \mathbb{R}^{N \times n}$ . In the framework of GMG, Eq. 2.2 and Eq. 2.3 often satisfy two variational properties:

$$\begin{aligned} A_H &= I_h^H A_h I_H^h & (\text{Galerkin condition}), \\ I_H^h &= k(I_h^H)^T & k \in \mathbb{R}. \end{aligned}$$

The idea behind GMG is to relax Eq. 2.2 (i.e., approximately solve Eq. 2.2 via a simple iterative scheme) and relax/solve Eq. 2.3 alternatively in a complementary fashion where high frequency errors are primarily reduced by relaxation of Eq. 2.2 while low frequency errors are damped by computations with Eq. 2.3. While these steps can be accomplished and combined in different ways (e.g., via recursion to “solve” the coarse equations), we consider the single two-level technique shown in Algorithm 3.1.

**Algorithm 3.1** Two-level multiplicative multigrid correction

---

Relax on  $A_h \mathbf{u}_h = \mathbf{f}_h$   
 Compute  $\mathbf{f}_H = I_h^H(\mathbf{f}_h - A_h \mathbf{u}_h)$   
 Solve  $A_H \mathbf{u}_H = \mathbf{f}_H$   
 Correct  $u_h = u_h + I_H^h u_H$

---

**4. Domain decomposition setting.** Domain decomposition preconditioning partitions a domain of interest into multiple regions to generate surrogate problems [15, 17, 18]. Approximated solutions to the surrogate problems are used in an iterative fashion until a matching criterion associated with the original problem is met.

There are multiple ways to partition the domain and to set up the surrogate problems. However, we omit these details (see [15, 17, 18] for further information) and restrict our attention to elliptic Dirichlet boundary value problems such as Eq. 2.1 with non-overlapping domain decomposition.

Consider a collection of subdomains  $\{\Omega_i\}_{i=1}^p$  partitioning  $\Omega$  such that

$$\Omega = \bigcup_{i=1}^p \Omega_i.$$

Such a partitioning of  $\Omega$  allows one to define a family of differential problems

$$\begin{cases} \mathcal{L}(u_i) = -\nabla \cdot (a(\mathbf{x}) \nabla u_i) + \mathbf{b}(\mathbf{x}) \cdot \nabla u_i + c(\mathbf{x}) u_i = f_i & \text{in } \Omega_i \\ u_i = g_i & \text{on } \partial\Omega_i, \end{cases} \quad i = 1, \dots, p. \quad (4.1)$$

Discretizing each with  $n_i$  degrees of freedom leads to the linear systems

$$A_i \mathbf{u}_i = \mathbf{f}_i, \quad i = 1, \dots, p. \quad (4.2)$$

Moreover, introducing binary restriction operators  $R_i$  to map mesh vertices in  $\Omega$  to the subset of mesh vertices in  $\Omega_i$ , we obtain

$$A_i = R_i A R_i^T, \quad i = 1, \dots, p.$$

In practice, the  $R_i \in \mathbb{R}^{n_i \times n}$  are rectangular matrices resulting from permuting a subset of rows of identity matrices. Defining

$$B_i = R_i^T (R_i A R_i^T)^{-1} R_i,$$

we can formulate an additive Schwarz fixed point method to solve Eq. 2.2 as follows:

$$\mathbf{u}^{\ell+1} = \mathbf{u}^\ell + \sum_{i=1}^p B_i (\mathbf{f} - A_h \mathbf{u}^\ell) \quad (4.3)$$

where  $\mathbf{u}^\ell$  is the approximated solution at the  $\ell$ th iteration of the fixed point scheme.

Equation 4.3 can be reinterpreted as a preconditioned Richardson scheme where the preconditioner acting on the residual  $\mathbf{f} - A \mathbf{u}^\ell$  is

$$B_{DD} = \sum_{i=1}^p B_i.$$

**5. Two-level Schwarz preconditioner.** The convergence rate of the additive Schwarz preconditioner can often be accelerated by introducing a coarse grid correction (similar to the GMG method) [15, 17]. Algorithm 5.1 and Algorithm 5.2 illustrate two possible multiplicative two-level methods. In Algorithm 5.1 the coarse correction is computed first, followed by the application of the  $B_{DD}$  operator. Algorithm 5.2 reverses the order of these two steps.

---

**Algorithm 5.1** Two-level Schwarz preconditioner 1

---

Input (vector to precondition):  $\mathbf{r}$

Compute  $\mathbf{v} = I_H^h A_H^{-1} I_h^H \mathbf{r}$

Compute  $\mathbf{v} = \mathbf{v} + B_{DD}(\mathbf{r} - A_h \mathbf{v})$

Output:  $\mathbf{v}$

---



---

**Algorithm 5.2** Two-level Schwarz preconditioner 2

---

Input (vector to precondition):  $\mathbf{r}$

Compute  $\mathbf{v} = B_{DD} \mathbf{r}$

Compute  $\mathbf{v} = \mathbf{v} + I_H^h A_H^{-1} I_h^H (\mathbf{r} - A_h \mathbf{v})$

Output:  $\mathbf{v}$

---

Mathematically, these can be written as

$$B = B_{DD} + (I - B_{DD} A_h) I_H^h A_H^{-1} I_h^H \quad (5.1)$$

and

$$B = B_{DD} + I_H^h A_H^{-1} I_h^H (I - A_h B_{DD}). \quad (5.2)$$

respectively. We choose to apply Eq. 5.1 for this paper.

As noted, the main concern for multiplicative schemes is the sequential nature of the two stages and the fact that the coarse problem has much less concurrent work. Although Algorithm 5.1 and similar multilevel formulations have already been investigated extensively, we aim to rewrite such a scheme through an unconventional formulation. The goal is to recast Algorithm 5.1 so that it formally appears as a purely additive scheme, as additive preconditioners have greater potential concurrency than multiplicative ones. By recasting Algorithm 5.1 in a completely equivalent formulation, the convergence rate should be identical to the multiplicative version.

To this goal, we introduce a smoothed prolongation operator mimicking the process shown by Vassilevski and Meier Yang in [20]:

$$\bar{I}_H^h = (I - B_{DD} A_h) I_H^h \quad (5.3)$$

which is essentially the highly successful *smoothed aggregation* prolongator introduced by Vanek in [19]. The construction of a smoothed prolongator usually starts by decomposing the set of grid nodes into small subsets to define a tentative often piecewise-constant interpolation operator (in the discrete sense). The tentative piecewise-constant prolongator is then smoothed to obtain the final prolongator.

Introducing the smoothed interpolator Eq. 5.3 allows us to interpret the Two-Level Schwarz preconditioner in Eq. 5.1 as

$$B = B_{DD} + \bar{I}_H^h A_H^{-1} I_h^H. \quad (5.4)$$

Note the additive nature of Eq. 5.4 and its similarity with the standard Two-Level Additive Schwarz preconditioner [17]

$$B_{\text{add}} = B_{DD} + I_H^h A_H^{-1} I_h^H. \quad (5.5)$$

The difference between Eq. 5.4 and Eq. 5.5 is that the two-level additive Schwarz preconditioner uses the standard  $I_H^h$ , whereas in Eq. 5.4 a smoothed interpolator  $\bar{I}_H^h$  is employed. The smoothing operator is the iteration matrix  $(I - B_{DD}A_h)$  of the DD fixed point iterative solver. The special interpolator  $\bar{I}_H^h$  allows one to recover the favorable convergence rate of a multiplicative method, though it does incur some expense to form  $\bar{I}_H^h$  and to apply  $\bar{I}_H^h$ , which contains more nonzeros than  $I_H^h$ .

**6. Approaches to apply the preconditioner.** Algorithm 6.1 presents high-level code to implement Eq. 5.4 that requires one to build explicitly the smoothed interpolation operator  $\bar{I}_H^h = (I - B_{DD}A_h)I_H^h$ . The hope is that the cost to build  $\bar{I}_H^h$ , which needs be done just once in the preconditioner setup phase, will be offset by a significant reduction of the run-time phase by performing  $A_H^{-1}$  concurrently with the  $B_{DD}$  subdomain solves. The run-time reduction could thus compensate and amortize the time spent to precompute the smoothed interpolator explicitly.

---

**Algorithm 6.1** Modified approach

---

Input (vector to precondition):  $\mathbf{r}$   
 Compute  $\mathbf{v1} = B_{DD}\mathbf{r}$  // concurrent  
 Compute  $\mathbf{v2} = \bar{I}_H^h A_H^{-1} I_h^H \mathbf{r}$  // concurrent  
 Compute  $\mathbf{v} = \mathbf{v1} + \mathbf{v2}$   
 Output:  $\mathbf{v}$

---

**6.1. Construction of the smoothed interpolation operator.** To accurately assess the potential of Algorithm 6.1, the time spent on the explicit construction of the smoothed interpolation operator  $\bar{I}_H^h$  should be minimized. Recalling Eq. 5.3, we notice that two matrix-matrix multiplications are necessary to compute  $B_{DD}A_h I_H^h$ . Both the size of each matrix and their sparsity patterns must be considered. Since  $I_H^h \in \mathbb{R}^{n \times N}$  and usually  $N \ll n$ , this matrix consists of a relatively small number of columns. This motivates computing  $A_h I_H^h$  first, as the result also contains only a few columns. For now we apply this matrix-matrix multiply naively, without specifically exploiting the column structure of  $I_H^h$ . Once  $A_h I_H^h$  is computed we premultiply this quantity by the domain decomposition preconditioner  $B_{DD}$ . In this case we exploit the sparsity pattern of  $B_{DD}$ ,  $A_h I_H^h$  and of the resulting matrix  $B_{DD}A_h I_H^h$ . Such a matrix-matrix multiply can be decomposed additively as follows:

$$B_{DD}A_h I_H^h = \sum_{i=1}^p [B_i(A_h I_H^h)]. \quad (6.1)$$

Clearly each term of the sum can be computed independently. Moreover, since the  $B_i$ 's are domain decomposition preconditioners associated with subdomains, each of them operates on a subset of the rows in  $A_h I_H^h$ . This allows one to decrease significantly the communication cost and memory storage to carry out each of the products  $B_i(A_h I_H^h)$ .

**6.2. Domain ordering (coloring).** To simplify the algorithms we consider structured Cartesian meshes with one aggregate per subdomain. Therefore there is a one-to-one correspondence between subdomains partitioning the fine mesh and nodes of the coarse mesh. This implies that the number of subdomains is equal to the number of columns in  $I_H^h$ .

The implementation described in Section 7 requires the application of  $B_{DD}$  to a matrix stored in a dense, rather than a sparse format. This is due to software limitations in Ifpack2 [5]. For efficiency in computing Eq. 6.1, we thus want to minimize the number of times the domain decomposition preconditioner BDD is applied to  $A_h I_H^h$  saved as a dense matrix. One possible option colors subdomains that can be processed independently with the same color as shown in Figure 6.1. In doing this, one must consider the stencil expansion each column undergoes due to the matrix-matrix multiply  $A_h I_H^h$  and the domain decomposition step  $B_i(A_h I_H^h)$ . One must ensure that subdomains (or equivalently matrix columns) with the same color are associated with non-overlapping columns of the final  $\bar{I}_H^h$ . Specifically, if two columns of  $B_i A_h I_H^h$  do not have any nonzero entries in the same rows, then they can belong to the same color and be computed simultaneously by “condensing” these columns into a single column. Repeating this idea for each color, which includes a set of columns that do not have nonzero entries in the same rows, is equivalent to building a condensed form of  $A_h I_H^h$ , denoted by  $C(A_h I_H^h)$ . The construction of  $C(A_h I_H^h)$  is executed by compressing directly  $I_H^h$ , followed by multiplying  $A_h$  and the compressed version of  $I_H^h$ . The number of columns in  $C(A_h I_H^h)$  is equal to the number of colors. One can then apply  $B_{DD}$  to  $C(A_h I_H^h)$ . As there are far fewer columns, there are far fewer applications of  $B_{DD}$ . The resulting  $B_{DD}C(A_h I_H^h)$  matrix can then be uncondensed to recover  $B_{DD}A_h I_H^h$ . Figure 6.1 provides a concrete example of this compression process.

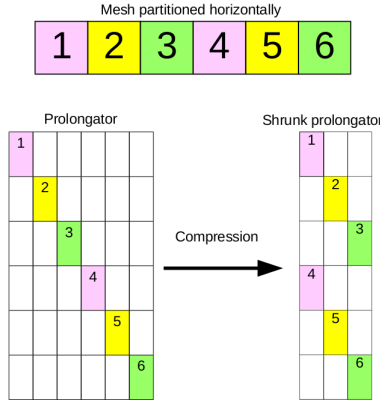


Fig. 6.1: Prolongator compression using subdomain coloring of a horizontally partitioned mesh.

Assuming no overlap among subdomains, and no overlap within the  $I_H^h$  columns, the minimal number of colors is  $3^d$ , where  $d$  is the spatial dimensional of the PDE problem. As illustrated in Figure 6.2, the fine mesh is partitioned into two-dimensional bricks. Each brick represents a subdomain, and all nodes on the fine mesh associated with a brick are aggregated to a single node on the coarse level. For this work, which is still at a proof-of-

concept stage, we assume the brick-shaped aggregates have the same size in each dimension. Subdomains with the same color are then assigned to the same column of the collapsed operator. In our specific context  $3^d$  colors are needed. However in more complex cases (e.g., overlapping subdomains), the same idea could be applied but more colors would be needed.

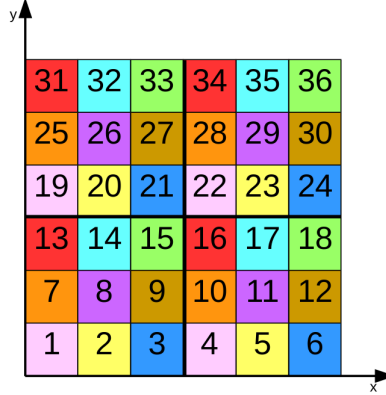


Fig. 6.2: Coloring scheme for two-dimensional structured Cartesian meshes partitioned with brick-shaped subdomains without overlapping.

**6.3. Memory storage and operation counts.** Algorithms 5.1 and 6.1, though mathematically equivalent, have different storage and computation characteristics. Table 6.1 reports the resources needed to apply the preconditioner. Every time a solution to a linear system is computed, we assume that such a solution is obtained by solving directly two linear systems involving LU factors.

Concerning memory storage, both approaches require saving the LU factors of the DD preconditioner, the DD restriction operators, and the LU factors of the stiffness problem on the coarse grid. The standard approach requires storing the regular restriction and prolongation operators, whereas the modified approach replaces the standard prolongation operator with the smoothed interpolator  $\bar{I}_H^h$ . Since the smoothed prolongation  $\bar{I}_H^h$  is the result of a matrix-matrix multiply, we expect fill in with respect to the sparsity of  $I_H^h$ . In our context we expect the number of nonzeros  $nnz(\bar{I}_H^h) \approx 9 \cdot nnz(I_H^h)$  in 2D when the discretization stencil corresponds to a nine point stencil. In particular, each column of  $I_H^h$  is nonzero only within one sub-domain while the nonzeros in each column of  $\bar{I}_H^h$  also extend over the eight nearest neighbor subdomains of a central subdomain. In 3D we expect  $nnz(\bar{I}_H^h) \approx 27 \cdot nnz(I_H^h)$ . Notice that the modified approach does not require that  $A_h$  be stored since the computation of the residual is not necessary, though typically an outer Krylov method would require this.

While requiring additional memory, the modified approach does not require a matrix-vector product with  $A_h$ , though the matrix-vector product with  $\bar{I}_H^h$  is likely more expensive than one with  $I_H^h$  needed for the standard approach.



	Standard approach	Modified approach
Memory	$\sum_{k=1}^p [nnz(L_k) + nnz(U_k)] +$ $\sum_{k=1}^p nnz(R_k) +$ $nnz(L_H) + nnz(U_H) +$ $nnz(I_h^H) + nnz(I_H^h) +$ $nnz(A_h)$	$\sum_{k=1}^p [nnz(L_k) + nnz(U_k)] +$ $\sum_{k=1}^p nnz(R_k) +$ $nnz(L_H) + nnz(U_H) +$ $nnz(I_h^H) + nnz(\bar{I}_H^h)$
Matrix - vector	$MV(I_h^h) + MV(I_h^H) +$ $2 \sum_{k=1}^p MV(R_k) + MV(A_h)$	$MV(\bar{I}_H^h) + MV(I_h^H) +$ $2 \sum_{k=1}^p MV(R_k)$
Direct solves	$\sum_{k=1}^p [DS(L_k) + DS(U_k)]$	$\sum_{k=1}^p [DS(L_k) + DS(U_k)]$

Table 6.1: Estimation of the memory storage and number of algebraic operations needed to apply the Two-Level Schwarz preconditioner according to different approaches. Matrices  $L_k$  and  $U_k$  are the LU factors of the matrix  $A_k$  discretizing the stiffness operator on a subdomain, and  $R_k$  is the binary projection operator associated with a subdomain.  $L_H$  and  $U_H$  are the LU factors of the coarse grid stiffness matrix  $A_H$ .

**7. Implementation.** The modified approach presented in Section 6 has been implemented in C++ using a Trilinos linear algebra framework [11] with an MPI environment for distributed memory parallelization [7, 9]. The main Trilinos packages include Tpetra [10], Galeri [4], Ifpack2 [5], MueLu [8] and Belos [1].

The package Tpetra is employed for the instantiation and handling of matrix and multivector structures, along with the construction of maps to partition data across the multi-process computational environment. Galeri has been extended to provide two-dimensional and three-dimensional advection-diffusion-reaction elliptic problems discretized with centered finite differences. Ifpack2 is necessary to build  $\bar{I}_H^h$  and to apply the overall domain decomposition cycle. MueLu is employed to set up the Two-Level Schwarz preconditioner used at each iteration of the iterative solver constructed with the Belos package.

Since the project goal is to recast the multiplicative Two-Level Schwarz preconditioner in an additive form to apply concurrently fine smoothing and the coarse correction, the V-cycle (see [13]) called `Iterate` in MueLu is rewritten. `Iterate` belongs to the templated class named `MueLu::Hierarchy`. Rewriting is necessary because the MueLu version employs recursion for the V-cycle, preventing different levels from being computed simultaneously. Also, reimplementing `Iterate` requires partitioning data and tasks among the processes in a different way from what is done conventionally, as illustrated in Figure 7.1. The idea behind our approach is to instantiate a total of  $P + 1$  processes with ID's from 0 to  $P$  at the fine level via a global MPI communicator. However, the data associated with the fine level are uniformly partitioned only among the processes with ID from 1 to  $P$ , leaving the master process with ID 0 free to take care of the coarse correction. Such a configuration allows one to operate concurrently fine level smoothing and coarse grid correction.

The data partitioning at the fine level is constructed manually via `Tpetra::Map` needed for `Tpetra::CrsMatrix` and `Tpetra::MultiVector` objects. Since the coarse level is

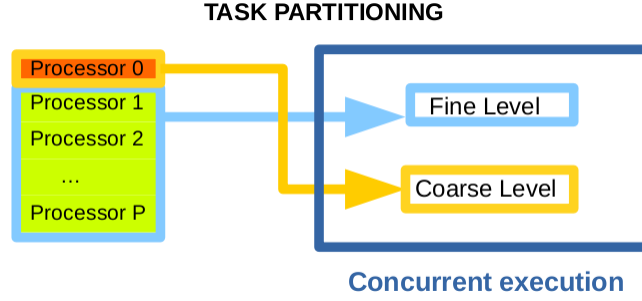


Fig. 7.1: Task partitioning for Two-Level Schwarz preconditioners.

managed just by one process, a repartitioning of the data is carried out to allow process 0 to be in charge of everything on that level. MueLu's repartitioning capabilities are used to assign the entire coarse level problem only to process 0.

A high level sketch of `MueLu::Hierarchy::Iterate` is reported here below.

```

template <...>
Return Type Hierarchy <...>::Iterate (...) {
    ...
    RCP<Level>    Fine = Levels_[0];
    RCP<Level>    Coarse;
    RCP<Operator> A = Fine->Get< RCP<Operator> >("A");

    // if coarse level
    if( Levels_.size()>1 ) {
        Coarse = Levels_[1];
        R = Coarse->Get< RCP<Operator> >("R");
        P = Coarse->Get< RCP<Operator> >("P");
        R->apply(B, *coarseRhs, Teuchos::NO_TRANS, one, zero);
        ...
        Ac = Coarse->Get< RCP< Operator > >("A");
        ...
    }

    // if fine level
    if (Fine->IsAvailable("PreSmoother")) {
        ...
        preSmoo->Apply(*fineX, B, zeroGuess); // apply BDD
        ...
    }
    ...

    // if coarse level
    if( Levels_.size()>1 ) {
        if (Coarse->IsAvailable("PreSmoother")) {
            // apply  $AH^{-1}$  in our context

```

```

preSmoo_coarse->Apply(*coarseX, *coarseRhs, ...);
...
}
...
P->apply(*coarseX, *coarseX_prolonged, ...);
}
// X_update sums components
X.update(1.0, *fineX, 1.0, *coarseX_prolonged, 0.0);
...
}

```

**8. Numerical results.** The performance of the standard and modified approaches are tested on the three-dimensional elliptic problem given by Eq. 2.1 with homogeneous Dirichlet boundary conditions. The domain is chosen to be  $\Omega = [0, 6] \times [0, 6] \times [0, 6]$ , and the equation coefficients are set to  $a(\mathbf{x}) = 1$ ,  $\mathbf{b}(\mathbf{x}) = [10, 10, 10]^T$ ,  $c(\mathbf{x}) = 0$  and  $f(\mathbf{x}) = 1$ . The discretization of the PDE is carried out via seven-point stencil centered finite differences. The actual size of the problem varies according to the number of nodes employed to discretize the domain.

Simulations are run on Edison, a National Energy Research Scientific Computing Center (NERSC) supercomputer [3]. This supercomputer is a Cray XC30 [2], endowed with 5,576 nodes. Each node has two sockets and each socket is populated with a 12-core Intel “Ivy Bridge” processor [6] at 2.4 GHz, for a total of 24 cores per node.

A Two-Level Schwarz preconditioner is employed, where the fine smoothing is computed with a domain decomposition approach. Each subdomain solve and the coarse solve are approximated via an incomplete LU factorization (ILUK) with a level of fill-in equal to 10 [16]. This large level of fill was chosen to mimic a standard sub-domain direct solver that was not available at the time of this project. The outer Krylov method is BiCGSTAB [16] as implemented in Belos. The Two-Level Schwarz is employed as right preconditioner and a threshold of  $10^{-10}$  is imposed on the relative residual as stopping criterion for BiCGSTAB. Simulations are run so that different subdomains are given to different cores without resorting to any threading. For each configuration, the number of allocated nodes is the minimal necessary to have enough cores available. This minimizes communication cost as 24 cores reside on the same node. All the results are averaged over 2 runs.

We first fix the problem size to 27,000,000 degrees of freedom employing a discretization with  $300 \times 300 \times 300$  nodes. Regarding the standard approach, we monitor the time spent to apply the restriction, the prolongation, the coarse solve, the smoothing at the fine level, the computation of the residual between fine and coarse level and the total time spent in the `MueLu::Hierarchy::Iterate`. Table 8.1 reports the maximal time spent across all the processes in seconds. For the modified approach, we additionally monitor the time spent to build  $\bar{I}_H^h$ , the time spent to apply  $\bar{I}_H^h$ , and the time to add the contributions of the two levels. In Table 8.2 the maximal time spent across all the processes is reported in seconds.

#domains	sub. size	$I_h^H$	$I_H^h$	$A_H^{-1}$	$B_{DD}$	residual	Iterate
216	$50 \times 50 \times 50$	0.31	0.2	0.03	27.2	0.9	28.5
1,728	$25 \times 25 \times 25$	0.06	0.02	0.23	3.39	0.1	7.29
3,375	$20 \times 20 \times 20$	0.03	0.01	0.49	1.64	0.52	14.8

Table 8.1: Timings in seconds for standard approach.

#dom.	sub. size	$I_h^H$	Set $\bar{T}_H^h$	$\bar{T}_H^h$	$A_H^{-1}$	$B_{DD}$	sum	Iterate
216	$50 \times 50 \times 50$	0.25	3.95	1.92	0.04	26.5	0.1	28.5
1,728	$25 \times 25 \times 25$	0.02	0.78	0.35	0.23	3.27	0.02	4.45
3,275	$20 \times 20 \times 20$	0.01	0.58	0.26	0.5	1.55	0.01	14.39

Table 8.2: Timings in seconds for modified approach. The column “Set  $\bar{T}_H^h$ ” shows the time spent in computing  $\bar{T}_H^h$ , whereas the column “ $\bar{T}_H^h$ ” shows the timings spent in applying the smoothed interpolator.

The comparison between Tables 8.1 and 8.2 shows that the additive approach is not faster using 216 or 3,375 subdomains. In the former case the size of the subdomains is too big relatively to the size of the coarse mesh. Therefore an excessive computational imbalance hinders any timing improvement. In the latter case the opposite situation occurs, since the number of processes involved is too high and the timings are dominated by communication overhead (not reported in the tables). In fact adding all the timings of the columns associated with the computational steps we obtain a value much smaller than the total time spent for the `MueLu::Hierarchy::Iterate`. However, resorting to 1,728 subdomains the additive approach reduces the timings almost by a factor of two with respect to the multiplicative variant. This suggests that an actual improvement may be obtained if the computation and communication loads are properly distributed across the processes. Generally, load balancing is more complex with the additive variant, as it must account for the fact that different processors execute different algorithm sub-components (with different work requirements). The most expensive communication comes from the transfer operators. For restriction, processes from 1 to  $P$  need to communicate their own portion of the restricted right hand side to process 0 to compute the coarse correction. In the prolongation the reverse occurs since process 0 has to redistribute the prolonged coarse correction to processes from 1 to  $P$ . To validate the effectiveness of the additive variant, one should choose the global size of the problem and of each subdomain so that the computation dominates over the communication. More specifically, it would be preferable if the domain decomposition step were computed concurrently with the time spent by process 0 to receive the restricted right hand side and the actual coarse computation. Unfortunately, we could not validate these conjectures due to limited computational resources.

We also investigated performance by fixing the size of each subdomain to  $25 \times 25 \times 25$  and varying the total number of subdomains/processors (and hence the total number of d.o.f’s) in a weak scaling fashion. In Table 8.3 we report results for the multiplicative variant whereas results for the additive variant are presented in Table 8.4. Here, the additive scheme outperforms the multiplicative one for the intermediate configurations. For the extreme cases the performance of the two approaches is similar. Also in this case using 3,375 causes most of the time to be spent for the communication. In fact adding up the timings for the computational step results in a value much smaller than the total time spent in the `MueLu::Hierarchy::Iterate` method. It is likely that again poor load balancing and poor computation to communication ratios limit the run time benefit of the additive variant.

#d.o.f's	#subdomains	$I_h^H$	$I_H^h$	$A_H^{-1}$	$B_{DD}$	residual	Iterate
$150 \times 150 \times 150$	216	0.03	0.01	0.02	1.72	0.05	1.86
$225 \times 225 \times 225$	729	0.04	0.01	0.07	2.6	0.09	3.72
$300 \times 300 \times 300$	1,728	0.06	0.02	0.23	3.39	0.1	7.29
$375 \times 375 \times 375$	3,375	0.06	0.02	0.58	4.8	0.17	23.39

Table 8.3: Timings in seconds for standard approach.

#d.o.f's	#dom.	$I_h^H$	Set $\bar{I}_H^h$	$\bar{I}_H^h$	$A_H^{-1}$	$B_{DD}$	sum	Iterate
$150 \times 150 \times 150$	216	0.08	0.48	0.16	0.02	1.65	0.01	1.82
$225 \times 225 \times 225$	729	0.01	0.59	0.26	0.07	2.6	0.01	2.82
$300 \times 300 \times 300$	1,728	0.02	0.78	0.35	0.23	3.27	0.02	4.45
$375 \times 375 \times 375$	3,375	0.02	1.16	0.56	0.5	3.79	0.02	22.5

Table 8.4: Timings in seconds for modified approach. The column “Set  $\bar{I}_H^h$ ” shows the time spent in computing  $\bar{I}_H^h$ , whereas the column “ $\bar{I}_H^h$ ” shows the timings spent in applying the smoothed interpolator.

**9. Conclusions and future developments.** We have investigated additive variants that are theoretically equivalent to the standard multiplicative Two-Level Schwarz algorithm. The goal is to enhance concurrency to improve performance of multilevel algorithms on upcoming computing architectures. We provided the mathematical framework and described approaches to implement the preconditioner, highlighting the key steps of the additive variant. The experiments show that an additive approach can leverage concurrency to reduce the total computational time. However, obtaining the proper load balance via a careful choice of problem size and number of subdomains can be challenging.

A prospective development may consist of employing direct solves for the domain decomposition smoother and for the coarse grid correction in place of the incomplete LU factorization adopted here. Moreover it may be worth investigating the performance of the preconditioners when multiple processes are in charge of computing the coarse grid correction.

**Acknowledgments.** This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

## REFERENCES

- [1] Belos - The Trilinos Project. <https://trilinos.org/packages/belos/>. Accessed: 2016-08-01.
- [2] Cray XC Series Supercomputers. <http://www.cray.com/products/computing/xc-series>. Accessed: 2016-08-01.
- [3] Edison, National Energy Research Scientific Computing Center. <http://www.nersc.gov/users/computational-systems/edison/>. Accessed: 2016-08-01.
- [4] Galeri - The Trilinos Project. <https://trilinos.org/packages/galeri/>. Accessed: 2016-08-01.
- [5] Ifpack2 - The Trilinos Project. <https://trilinos.org/packages/ifpack2/>. Accessed: 2016-08-01.
- [6] Intel - Ivy Bridge processor. <http://ark.intel.com/products/codename/29902/>. Accessed: 2016-08-01.
- [7] Message Passing Interface Forum. <https://www.mpi-forum.org/>. Accessed: 2016-08-01.

- [8] MueLu - The Trilinos Project. <https://trilinos.org/packages/muelu/>. Accessed: 2016-08-01.
- [9] Open MPI: Open Source High Performance Computing. <https://www.open-mpi.org/>. Accessed: 2016-08-01.
- [10] Tpetra - The Trilinos Project. <https://trilinos.org/packages/tpetra/>. Accessed: 2016-08-01.
- [11] The Trilinos Project. <https://trilinos.org/>. Accessed: 2016-08-01.
- [12] A. BRANDT, Multi-Level Adaptive Solutions to Boundary-Value Problems, *Mathematics of Computation*, 31 (1977), pp. 333–390.
- [13] W. L. BRIGGS, V. H. HENSON, AND S. F. MCCORMICK, *A Multigrid Tutorial, Second edition*, SIAM, 2000.
- [14] L. C. EVANS, *Partial Differential Equations: Second Edition*, American Mathematical Society, 2010.
- [15] A. QUATERONI AND A. VALLI, *Numerical Approximation of Partial Differential Equations*, Springer, 2008.
- [16] Y. SAAD, *Iterative Methods for Sparse Linear Systems, Second Edition*, SIAM, 2003.
- [17] B. SMITH, P. BJORSTAD, AND W. GROPP, *Domain Decomposition, Parallel Multilevel Methods for Elliptic Partial Differential Equations*, Cambridge University Press, 1996.
- [18] A. TOSELLI AND O. B. WIDLUND, *Domain Decomposition Methods - Algorithms and Theory*, vol. 34, Springer Series in Computational Mathematics, 2005.
- [19] P. VANEK, Acceleration of Convergence of a Two-level Algorithm by Smoothing Transfer Operators, *Applications of Mathematics*, 37 (1992), pp. 265–274.
- [20] P. S. VASSILEVSKI AND U. M. YANG, Reducing Communication in Algebraic Multigrid Using Additive Variants, *Numer. Linear Algebra Appl.*, 21 (2014), pp. 275–296.
- [21] J. XU, Iterative Methods by Space Decomposition and Subspace Correction, *SIAM Review*, 34 (1992), pp. 581–613.

## HYBRID MULTIGRID METHODS FOR NEARLY STRUCTURED MESHES

PETER B. OHM\* AND RAYMOND S. TUMINARO†

**Abstract.** Multigrid methods have been developed for both structured and unstructured grids [2]. In terms of exascale computing, memory, and setup time, there are significant potential advantages to structured mesh approaches over unstructured schemes. Further, multigrid algorithms that take advantage of structure can exhibit superior convergence rates when compared with their unstructured multigrid counterparts. Unfortunately, the types of applications that can be addressed with structured meshes are quite limited. In many circumstances, however, an application may only require complex unstructured meshes within small sub-regions of the overall domain to accurately capture important physical features. For these applications, we aim to develop multigrid methods that can leverage the structure present over most of the domain while employing more general techniques only within regions where complex unstructured meshes are truly needed.

**1. Introduction.** This paper considers multigrid solvers for applications where the underlying mesh contains both structured and unstructured regions. We are specifically interested in developing a multigrid algorithm that can leverage structure even if the mesh is only structured within a region of the entire domain. Figure 1.1 provides an illustration of a mesh with regions that are both structured (the outer ring) and unstructured (the inner circle). The target application is one where most of the computational domain is covered by structured mesh, but there are a few sub-regions requiring some unstructured mesh treatment. This is quite common in many situations.

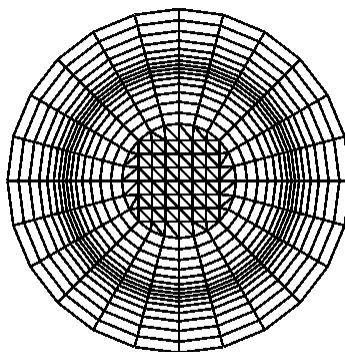


Fig. 1.1: Example of a hybrid mesh. The “structured” region is the region with the quad elements, while the “unstructured” center region is the region with the triangular elements

For structured regions one can employ a structured mesh multigrid algorithm. In this paper, we consider a multigrid algorithm that coarsens by factors of three in both dimensions of a logically structured mesh and derives grid transfer basis functions by a black box type matrix dependent algorithm [3]. Within the unstructured regions, a fully unstructured Algebraic Multigrid (AMG) algorithm is employed with the following exception: at the interfaces between structured and unstructured regions, we require that the set of coarse AMG

---

\*Tufts University, peter.ohm@tufts.edu

†Sandia National Laboratories, rstumin@sandia.gov

points be identical to the set of coarse points required by a structured multigrid algorithm. The use of structured multigrid within the structured region offers two primary potential advantages. The first advantage is algorithmic in that certain structured techniques can be applied on all grid levels of the hierarchy. This is demonstrated for an application containing a highly anisotropic mesh within the structured region, as shown in Figure 1.1. In this case, line relaxation can be employed within the structured region to effectively smooth errors during the iterative process in all directions despite the anisotropic mesh. A second potential advantage to the preservation of structure within a multigrid hierarchy might be realized on future architectures. In particular, structured multigrid gives rise to less nonzero fill-in on coarse grids (which corresponds to less communication), less indirect addressing (which corresponds to less bandwidth), and more regularity in the computations (which can be advantageous on GPUs). This second advantage will be explored in a future paper.

The paper proceeds as follows. Multigrid algorithms are described for regular meshes in Section 2. This includes a brief summary of Hierarchical Hybrid Grid (HHG) approaches that consider somewhat more complex domains by viewing them as the union of a set of structured regions. Smoothed aggregation multigrid is then summarized in Section 3. The hybrid mesh multigrid algorithm is described in Section 4, focusing on algorithmic modifications required at the interfaces between structured and unstructured meshes. Section 5 presents a computational illustration of the algorithmic advantages associated with preserving structure within the multigrid method for a sub-region of the mesh. Finally, conclusions are given in Section 6.

**2. Multigrid on Structured Grids.** Multigrid was originally developed to solve linear systems associated with elliptic partial differential equations (PDEs) on structured grids and then later adapted to address different unstructured grid situations. The multigrid method is the process of using a solution on the coarse grid to correct the solution on the fine grid. It can be shown both theoretically and computationally that simple iterative methods (e.g., Jacobi) are generally effective at removing high-frequency components of error and relatively inefficient at removing low-frequency components of error. However, low-frequency error components on the fine mesh appear as high-frequency error on the coarse mesh. Passing the problem to the coarse grid not only reduces the problem size, it also transforms the low-frequency error to high-frequency error on the coarse mesh where it can be more easily eliminated [2].

Here we lay out a basic multigrid procedure. First, one must construct the prolongation matrix  $P$  that maps from the coarse grid to the fine grid. Further, a restriction operator,  $R$ , is needed to map residuals from the fine grid to the coarse grid. Often,  $R = \alpha P^T$  where  $\alpha$  is a scalar ( $\alpha = 1$  in this paper). Overall, the general idea is that  $R$  and  $P$  should accurately transfer smooth functions. Additionally, a discretization of the original PDE problem is needed on the coarse mesh. This can be developed by applying the same discretization technique to the coarse mesh that was originally employed on the fine mesh. Alternatively, one can use a Galerkin style projection and take the coarse discretization to be the result of the triple matrix product  $RAP$ . Algorithm 2.1 illustrates a basic two level multigrid method where  $R = P^T$  and the coarse discretization is  $P^T AP$ . The pre- and post-smoothing shown in Algorithm 2.1 corresponds to a Jacobi algorithm where  $D$  is the diagonal of matrix  $A$  and  $\omega$  is a damping parameter between 0 and 1. Smoothing updates the current approximation to the solution and generally damps out high-frequency components of the error so that the error in the resulting approximation is smooth. Smoothness is normally required in order to accurately represent this error on a coarse mesh. In practice an approximation to the coarse level solution can be obtained through repeated recursive application of Algorithm 2.1 until one reaches a coarse level problem small enough that it can be solved with a direct method.



**Algorithm 2.1** Basic Two-Level Multigrid, solving  $Au = b$ 


---

**Pre-smoothing**  $u = u + \omega D^{-1}(b - Au)$   
**Compute fine grid residual**  $r^h = (b - Au)$   
**Restrict to coarse grid**  $r^{2h} = P^T r^h$   
**solve**  $(P^T A P) e^{2h} = r^{2h}$   
**Interpolate coarse grid error the fine grid**  $e^h = P e^{2h}$   
**Correct fine grid solution**  $u = u + e^h$   
**Post-smoothing**  $u = u + \omega D^{-1}(b - Au)$

---

As noted above, multigrid was initially developed for structured meshes. In this case, the application typically supplies a hierarchy of structured meshes. Typically the spacing between mesh points on the  $\ell^{th}$  mesh is taken as  $h_\ell = 2^{(\ell-1)}h_1$  where the finest mesh corresponds to  $\ell = 1$  with a mesh spacing of  $h_1$ .

Prolongation matrices are needed to map solutions between different levels of the hierarchy. This prolongation matrix is generally quite sparse and could be based on bi- or tri-linear interpolation (in two and three dimensions), which is easily derived for a structured mesh. In this case, linear functions on the coarse mesh are exactly transferred to the fine mesh. Alternatively, matrix dependent multigrid algorithms such as the well known black-box multigrid scheme [3] have been defined for structured meshes. These matrix dependent algorithms define grid transfer operators using only the matrix coefficients. In this sense they are similar to algebraic multigrid (AMG). However, unlike AMG, matrix-dependent multigrid requires that both the fine and the coarse mesh be structured. In this paper, we employ a matrix dependent scheme that has been adapted from [6] so that a coarsening factor of three is used in each dimension (i.e.,  $h_\ell = 3^{(\ell-1)}h_1$ ). Figure 2.1 depicts a structured fine mesh. The colored nodes can be used to define a coarse mesh where the mesh spacing between colored nodes is three times that of the fine mesh spacing.

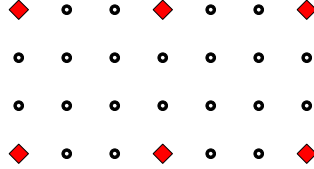


Fig. 2.1: Example structured mesh. The coarse nodes, represented with red diamonds, define a coarse mesh where the mesh spacing between coarse nodes is three times that of the fine mesh spacing.

Structured mesh multigrid algorithms have several advantages for exascale compute systems and in terms of convergence. For exascale systems, the regularity of structured meshes allows the use of stencil based algorithms avoiding the need to explicitly assemble a discretization matrix and it is easier to construct a partitioning of the mesh for parallelization. Additionally, there is less nonzero fill-in on coarse grids and less indirect addressing. Structured mesh multigrid can also exhibit significantly better convergence properties than more general purpose AMG methods in certain situations. From our experience, this is most

likely to occur when there is a certain directionality to the underlying problem that is aligned with the mesh. This might be an interface, some anisotropic phenomena, characteristics for hyperbolic problems, boundary layers, or mesh stretching. Additionally, structured mesh multigrid methods can employ line relaxation on all levels of a multigrid hierarchy. Line relaxation has well known convergence benefits for anisotropic problems and is a valuable multigrid tool when employing structured grid multigrid.

Several algorithms have been proposed to extend the applicability of structured mesh algorithms (e.g., using a set of overlapping structured meshes to cover an irregular domain or to adapt the mesh resolution to sub-regions requiring greater accuracy). The Hierarchical Hybrid Grids (HHG) method is probably the closest generalization of structured mesh ideas to the approach considered in this paper. Hierarchical Hybrid Grids (HHG) are formed from the regular refinement of a coarse grid. The result is a HHG grid hierarchy containing regions of structured mesh, even if the initial coarse mesh was unstructured [1]. Each region in an HHG mesh corresponds to one element of the original coarse mesh that has been uniformly refined. Figure 2.2 illustrates the regular refinement of a mesh composed of a triangle and a square.

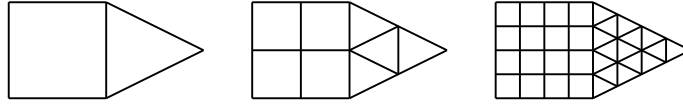


Fig. 2.2: Example of regular refinement of a mesh

One can use finite element techniques to discretize a PDE on an HHG mesh. The main difference is that it is natural within an HHG implementation to assemble the individual stiffness matrices within each region to form one partially assembled matrix for each region. Thus, there are multiple copies of unknowns along regional interfaces (one for each region containing the unknown). In this way, the partially assembled matrices along shared interfaces appear locally as Neumann boundary conditions (as contributions to degrees-of-freedom on the shared interface have not been assembled). Mathematically, the HHG discretization and the use of a structured multigrid method is completely equivalent to having formed a standard finite element discretization and applied a structured multigrid algorithm to the fully assembled matrix. However, it is natural to implement the method in a region oriented fashion with some kernels that are applied to a sub-assembled matrix or to vector subsets defined on just one region. This means that the implementation must additionally include some functions that account for interfaces and neighboring regions. For example, functions are needed to sum or average sub-vector components associated with the same shared degrees-of-freedom. We omit the details but refer the reader to [1].

**3. Algebraic Multigrid.** Most algebraic multigrid methods work by using properties of the fully assembled matrix to construct prolongation matrices. The development and use of algebraic multigrid methods has been a major advancement in the solution of linear systems associated with partial differential equations. This is because the extension of more traditional geometric multigrid algorithms to highly unstructured complex applications can be quite cumbersome, especially for application developers.

Algebraic multigrid facilitates the use of multigrid methods within complex applications. However, it can potentially incur some additional costs and may be less robust than a structured multigrid approach (though there are examples where an algebraic approach can be more robust than a standard geometric multigrid method). In particular, the cost of the setup phase for an algebraic method is typically much higher than that of a geometric method. Coarse grid discretization stencils obtained via AMG can be significantly larger than the fine grid discretization stencil. This is particularly problematic on highly concurrent grids as large stencil widths typically imply greater communication requirements. The larger stencils and the use of indirect addressing within unstructured applications also increases memory bandwidth and decreases the regularity of memory access, which can also degrade performance on some architectures. Additionally, applications that have features that are naturally aligned with the coordinate directions of the mesh are often better captured by coarse meshes that maintain this alignment. That is, structured mesh approaches have convergence benefits for some types of applications. Furthermore, structured mesh approaches can employ line relaxations techniques to further improve robustness and convergence rates. Line relaxation techniques have well known convergence benefits for anisotropic problems, but they can only be applied to structured meshes.

In this work we have considered both the smoothed aggregation method [7] and a form of energy minimization algebraic multigrid [4, 5]. Here, we restrict our discussion to the smoothed aggregation (SA) method as this is the more standard approach and is the primary AMG method used in our experiments. The basic idea of all AMG methods is to first interpret the matrix  $A$  as a graph. Each matrix row corresponds to one graph vertex and an edge between vertices  $i$  and  $j$  exists if the corresponding matrix entry  $a_{i,j}$  is nonzero. An aggregation algorithm is applied to this graph. This algorithm groups neighboring vertices into aggregates using some type of heuristic graph algorithm. The aggregation algorithm used in this work essentially chooses an unaggregated vertex and all of its immediate unaggregated neighbors to define an aggregate. Repeated application of this process leads to a set of aggregates,  $\mathcal{A}_j$ , such that intersection of any two aggregates is empty and the union of all aggregates includes all graph vertices. The node aggregates are used to define the *tentative prolongator*

$$\hat{P}_{ij} = \begin{cases} 1, & \text{if } i \in \mathcal{A}_j \\ 0, & \text{otherwise} \end{cases} \quad (3.1)$$

that is then improved via

$$P = \left( I - \frac{4}{3\rho} D^{-1} A \right) \hat{P} \quad (3.2)$$

where  $I$  is the identity matrix,  $\rho$  is the spectral radius of  $A$ , and  $D$  is the diagonal of  $A$ .

For SA to produce an efficient prolongation matrix, the aggregates  $\mathcal{A}_j$  need to be “nice” aggregates in the sense that all aggregates should be approximately the same size and shape. The basic idea behind the construction of the smoothed prolongation matrix is that  $\hat{P}$  defines a simple piecewise constant interpolation operator, which is not necessarily ideal. However, the piecewise constant basis functions (i.e. the columns of  $\hat{P}$ ) can be smoothed by applying the Jacobi-like iteration operator given in Equation 3.2. When aggregates are chosen appropriately, an efficient multigrid method can be created (as can be shown both theoretically and computationally for a wide class of elliptic PDE problems).

**4. HHG and AMG.** In many applications, it is not possible to use a structured mesh nor is it possible to use an HHG mesh. That is, the original application can contain some

features that are not easily resolved or captured with a structured mesh. However, it is quite common that the features which complicate the use of structured meshes are restricted to some small subset of the entire domain. For these cases, we seek to modify the HHG scheme so that it allows for some unstructured regions. Specifically, we consider meshes that are composed of multiple regions with some regions covered by structured meshes and other regions covered by unstructured meshes.

The principle idea is to create a prolongation operator by applying a structured multigrid algorithm (e.g., black-box multigrid) to the sub-assembled matrix associated with structured regions and then apply an AMG algorithm to the sub-assembled matrix associated with the unstructured regions. Effectively, this implies that some rows of the prolongation matrix are determined by a black-box algorithm while other rows are determined by an AMG algorithm.

The main non-trivial issue when combining AMG and structured mesh multigrid concerns the treatment at the interface. In particular, one needs to make sure that coarse points agree across an interface. The coarse points of the structured multigrid method must be defined as a uniform coarsening of a structured mesh and so they are known for each level of the mesh hierarchy. Thus, the AMG method must be adapted so that it has the same coarse points at the interface as the structured method. These coarse points are effectively the centroid of the aggregates discussed in Section 3. For SA we artificially aggregate the shared interface separately from the rest of the region. The artificial aggregation is performed so that the coarse points of the unstructured region agree with the coarse points of the adjacent structured region as shown in Figure 4.1. This forces the stencil structure of

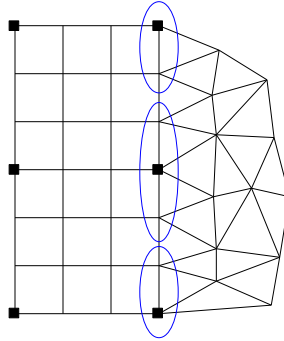


Fig. 4.1: Example of how the unstructured mesh was aggregated along a shared interface. Coarse points on the structured region are represented by the black square dots. The interface aggregates are circled in blue

the SA prolongation matrix to match the prolongation matrix of the structured multigrid. While the nonzero patterns of the two prolongation matrices agree along the interface, the actual coefficients associated with the stencils are different. As a consistent interpolation operator is needed, one can consider averaging the interpolation operators or one can simply take one of the interpolation operators to define the interface interpolation. In this paper, we take the latter approach. Specifically, the prolongation matrix of the unstructured region

is set to agree across the interface by overwriting the shared interface values with those of the prolongation matrix in the structured region. It is important to notice that interpolation at non-coarse interface points is completely in terms of coarse points only at the interface. That is, interpolation at the interface does not depend on values from the interior of either of the neighboring regions.

While not strictly necessary, it is desirable that the coarsening rates of both the AMG and the structured multigrid methods are matched. In this way, the relative distance between coarse points within structured and unstructured regions is roughly the same throughout the multigrid hierarchy. As the smoothed aggregation method generally coarsens by factors of three in each dimension, the structured multigrid algorithm is also adapted so that it coarsens by a factor of three in each dimension (as opposed to the more standard factor of two in each dimension). This algorithm will be described in greater detail in a future report.

As with the HHG method, applying the multigrid cycle to solve a system (e.g. via Algorithm 2.1) is mathematically equivalent to applying multigrid to the entire assembled discretization matrix, using hybrid grid transfers that have been created by combining both structured and unstructured multigrid algorithms as described above. However, in practice the cycle is applied in a region-oriented fashion using sub-assembled discretization matrices. This region-oriented fashion is straightforward for most kernels but becomes complicated when applying relaxation within the multigrid scheme. For example, both Jacobi and  $x$ -line Jacobi relaxation can be written as

$$u^{k+1} = u^k + \omega D^{-1}(b - Au^k) \quad (4.1)$$

where  $D$  is the matrix diagonal of the fully assembled matrix  $A$  in the Jacobi case.  $D$  contains only fully assembled matrix entries that couple mesh nodes along  $x$ -lines in the  $x$ -line Jacobi case. That is, the only non-zeros in  $D$  satisfy  $D_{i,j} = A_{i,j}$  if  $i$  and  $j$  belong to the same grid line oriented in the  $x$  direction. While the matrix-vector product,  $Au$ , is easily performed using the sub-assembled region matrices, it is easiest to fully assemble the matrix  $D$  so that it can be inverted within the Jacobi schemes.

**5. Numerical Illustration.** To demonstrate the overall approach, a Poisson-like problem is solved on a circular domain. Specifically, consider

$$(\rho(x, y)u_x)_x + (\rho(x, y)u_y)_y = f \quad (5.1)$$

with  $u(x, y) = 0$  for  $x^2 + y^2 = 9$ , and

$$f(x, y) = \begin{cases} 1, & \text{for } ((x-1)^2 + (y-1)^2) \leq (1.25)^2 \\ 0, & \text{otherwise} \end{cases}$$

$$\rho(x, y) = \begin{cases} 100, & \text{for } (1.9)^2 \leq (x^2 + y^2) \leq (2.1)^2 \\ 1, & \text{elsewhere} \end{cases}$$

Meshes such as the one depicted in Figure 1.1 are employed. Notice the regular but non-uniform mesh spacing that is used to capture the interface associated with the discontinuity in  $\rho(x, y)$ . The problem is discretized by a general finite element strategy that includes both quadrilateral elements and triangular elements. As with HHG, the discretization matrices are sub-assembled for each region by a general linear nodal finite element strategy.

A simple meshing algorithm was developed to alter the mesh spacing between rings in the structured region, providing us with some ability to experiment with different mesh aspect

ratios. The disk mesh and non-uniform mesh spacing are used to highlight the advantages of using a hybrid structured and unstructured multigrid method. The center of the disk is treated as unstructured while the outer ring is structured. The outer ring uses a black-box multigrid method to construct prolongation and relaxation matrices, which preserve the regular structure of the outer ring. The structure allows the use of line relaxation as a smoother.

The large element aspect ratios near  $r = 1.9$  and  $r = 2.1$  can be problematic for multigrid algorithms. In particular, standard pointwise smoothers such as the Jacobi algorithm can be highly ineffective at smoothing errors in the direction of large mesh spacing. Figure 5.1 illustrates the error after applying 10 sweeps of Jacobi relaxation with a damping parameter of 0.5 to solve the problem  $Au = 0$  with a random initial guess. Notice that while the errors are smooth in the radial direction they are highly oscillatory in the angular direction. This

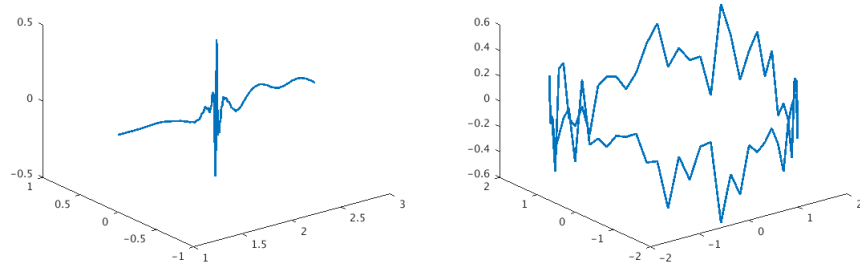


Fig. 5.1: Example of the Jacobi algorithm struggling to smooth large aspect ratio elements after 10 iterations. The figure on the left shows a radial cross-section, while the figure on the right shows an angular cross-section. The largest aspect ratio is 100.

smoothing deficiency can be overcome by employing line relaxation in the direction of the stretching, but this requires the use of a structured mesh so that line relaxation can be employed on the coarser grids within the hierarchy.

Smoothed aggregation is used to generate rows of the prolongation matrices associated with the center disk. During the multigrid cycle, Jacobi is always used as a relaxation method for unknowns associated with the inner disk.

Since different smoothers are being used on different regions, an additional communication step is needed to ensure solution agreement on the shared interface. Following any smoothing, like Jacobi or line smoothing, the solution at the shared points in the region that was not smoothed are overwritten with the solution from the region that was smoothed. This is demonstrated in Algorithm 5.1, where the subscript ' $o$ ' denotes a vector subset associated with the unknowns in the outer ring and the subscript ' $i$ ' denotes a vector subset associated with the unknowns in the disk center.  $u_{\Gamma_i}$  is the solution on the shared interface for the disk center,  $u_{\Gamma_o}$  is the solution on the shared interface for the outer ring.  $D_{angular}$  is the preconditioner for line smoothing in the angular direction,  $D_{radial}$  is the preconditioner for line smoothing in the radial direction, and  $D$  is given by the diagonal of  $A$ . For all three cases,  $D_{angular}$ ,  $D_{radial}$ , and  $D$  are constructed from the fully assembled  $A$ , yet only use the rows and columns that are relevant to the corresponding region.  $\omega$  is used as a damping factor.

Table 5.1 illustrates preconditioned conjugate gradient iteration counts using multigrid

**Algorithm 5.1** Line smoothing with additional communication

---

**angular line smoother in outer ring**  $u_o = u_o + \omega D_{angular}^{-1}(b - Au)_o$   
**overwrite shared values**  $u_{\Gamma_i} = u_{\Gamma_o}$   
**Jacobi smoother in disk center**  $u_i = u_i + \omega D^{-1}(b - Au)_i$   
**overwrite shared values**  $u_{\Gamma_o} = u_{\Gamma_i}$   
**radial line smoother in outer ring**  $u_o = u_o + \omega D_{radial}^{-1}(b - Au)_o$   
**overwrite shared values**  $u_{\Gamma_i} = u_{\Gamma_o}$   
**Jacobi smoother in disk center**  $u_i = u_i + \omega D^{-1}(b - Au)_i$   
**overwrite shared values**  $u_{\Gamma_o} = u_{\Gamma_i}$

---

Nodes Per Spoke	Number Spokes	Max Aspect Ratio	Total Unknowns	pcg iterations		
				Jacobi	Line Smooth	AMG
10	24	4.8	285	20	11	21
28	72	4.8	2417	27	13	31
82	216	4.7	21501	30	18	49
244	684	4.7	192773	32	27	62
28	24	14.2	717	47	12	45
82	72	14.2	6305	81	14	94
244	216	14.2	56493	90	18	100
10	24	132.7	285	29	12	31
28	72	124.4	2417	72	14	89
82	216	121.6	21501	100	19	100
244	684	120.6	192773	100	26	100
28	24	372.4	717	64	13	60
82	72	364.7	6305	100	14	99
244	216	361.9	56493	100	18	100

Table 5.1: Iteration counts for various methods (capped at 100). Jacobi is a two region method: black box multigrid on outside disk and SA on inside disk with a Jacobi smoother everywhere. Line Smooth is a two region method using black box multigrid and a line smoother on the outside disk and SA with Jacobi smoother on the inside disk. AMG is a one region method using SA with Jacobi smoother everywhere. (residual tolerance 1e-9, zero initial guess)

as a preconditioner for different mesh scenarios. Columns marked by “Jacobi” and “Line Smooth” are associated with hierarchies where the grid transfers are obtained in a region-oriented fashion using AMG for the unstructured center disk region and black-box multigrid for the structured outer ring region. The “Jacobi” column multigrid V cycle employs four sweeps of Jacobi pre-smoothing (with  $\omega = 0.5$ ) and four sweeps of Jacobi post-smoothing. The “Line Smooth” column applies the relaxation given in by the Algorithm 5.1 code fragment. The “AMG” column obtains grid transfers by applying AMG to the fully assembled matrix (i.e. the standard ‘one region’ method) and using four pre-smoothing and four post-smoothing Jacobi sweeps for relaxation.

For smaller aspect ratios, one can see that all of the methods are fairly competitive with each other. Overall, there is a modest improvement in using the black-box multigrid transfers in the structured region with Jacobi relaxation throughout the domain when compared with

using AMG grid transfers and Jacobi relaxation. There are additional iteration savings using the line relaxation smoothing.

For large aspect ratios, these iteration savings are much more pronounced. In fact, for some of the larger aspect ratios, the standard AMG approach with point relaxation does not even converge within 100 iterations while the line smoothing region-oriented version converges fairly robustly over a range of aspect ratios and mesh sizes. The key point to notice is that line relaxation is only possible because the structured mesh within the outer ring is maintained on all levels of the hierarchy.

**6. Conclusions.** A hybrid structured-unstructured multigrid algorithm has been presented for applications in which only a few unstructured regions are needed to accurately represent the underlying physical problem. Within the structured region a black-box multigrid algorithm is applied to generate grid transfer operators. Within the unstructured region an AMG method is applied. The main challenge in devising a hybrid approach is to assure that the prolongation operator is consistent at the interfaces between structured and unstructured regions. This primarily requires adapting the AMG's aggregation algorithm so that its aggregates at the interface are consistent with the coarse points and grid transfer sparsity pattern of the structured multigrid algorithm.

An example has been presented highlighting the potential convergence benefits of a hybrid structured-unstructured multigrid approach. In this example, the structured region included an anisotropic mesh that is used to capture features within a ring of interest. Line relaxation can be used within the structured region of the hybrid scheme to address smoothing limitations of point relaxation on anisotropic meshes. Line relaxation is not possible when a standard AMG method is applied to the entire mesh, as structure is not preserved throughout the hierarchy. Numerical results illustrate that the convergence rate of the hybrid method combined with line relaxation is relatively insensitive to the mesh element aspect ratio while convergence rates deteriorate for schemes that do not employ line relaxation.

## REFERENCES

- [1] B. K. BERGEN AND F. HÜLSEMAN, Hierarchical Hybrid Grids: Data Structures and Core Algorithms for MultiGrid, *Numerical Linear Algebra With Applications*, 11 (2004), pp. 279–291.
- [2] W. L. BRIGGS, V. E. HENSON, AND S. F. MCCORMICK, *A Multigrid Tutorial*, SIAM, second ed., 2000.
- [3] J. J. E. DENDY, Black Box Multigrid, *Journal of Computational Physics*, 48 (1982), pp. 366–386.
- [4] J. MANDEL, M. BREZINA, AND P. VANĚK, Energy Optimization of Algebraic Multigrid Bases, *Computing*, 62 (1999), pp. 205–228.
- [5] L. N. OLSON, J. B. SCHRODER, AND R. S. TUMINARO, A General Interpolation Strategy for Algebraic Multigrid Using Energy Minimization, *SIAM J. Sci. Comput.*, 33 (2011), pp. 966–991.
- [6] R. TUMINARO, M. PEREGO, I. TEZAUER, A. SALINGER, AND S. PRICE, A Matrix Dependent/Algebraic Multigrid Approach for Extruded Meshes with Applications to Ice Sheet Modeling, *SIAM J. Sci. Comput.*, accepted for publication (2016).
- [7] P. VANĚK, M. BREZINA, AND J. MANDEL, Convergence of Algebraic Multigrid Based on Smoothed Aggregation, *Numer. Math.*, 88 (2001), pp. 559–579.



## A HYBRIDIZED DISCONTINUOUS GALERKIN METHOD FOR RESISTIVE INCOMPRESSIBLE MAGNETOHYDRODYNAMICS

STEPHEN J. SHANNON\*, JOHN N. SHADID†, TAN BUI-THANH‡, AND JEONGHUN J. LEE§

**Abstract.** An important base-level representation for continuum approximation of the dynamics of charged fluids in the presence of electromagnetic fields is the resistive magnetohydrodynamics (MHD) model. MHD models describe important physical phenomena in the physical world (e.g., solar flares, and planetary magnetic field generation) as well as in science and technological applications (e.g., magnetically confined fusion energy [e.g. ITER]). This multiphysics PDE system is strongly coupled, highly nonlinear, and characterized by multiple interacting physical phenomena spanning a very large range of length- and time-scales. For effective discrete approximation of these systems, high-order spatial discretizations and some form of implicit temporal integration can be attractive methods. In this study the development and analysis of a new hybridizable (also known as hybridized) discontinuous Galerkin (HDG) method for the solution of resistive incompressible MHD systems is presented. This development considers the construction of an HDG solver for a linearized system, which is used as a sub-step in a fixed-point nonlinear solver. Optimal convergence in the velocity and magnetic field variables for steady and unsteady benchmark problems is demonstrated.

**1. Introduction.** The discontinuous Galerkin (DG) method, originally developed by Reed and Hill [21] for the neutron transport equation, roughly speaking, combines advantages of classical finite volume and finite element methods. In particular, it has the ability to treat solutions with large gradients including shocks, provides the flexibility to deal with complex geometries, and is highly parallelizable due to its compact stencil. However, for steady-state problems or time-dependent ones that require implicit time-integrators, DG methods typically have many more coupled unknowns compared to the other existing numerical methods, and hence are more expensive in general.

In order to mitigate the computational expense associated with DG methods, Cockburn, coauthors, and others have introduced hybridizable (also known as hybridized) discontinuous Galerkin (HDG) methods for various types of PDEs including Poisson-type equation [5, 16], Stokes equation [14], and Euler, Navier-Stokes, and wave equations [17, 19, 18]. The upwind HDG framework proposed in [3] provides a unified and a systematic construction of HDG methods for a large class of PDEs. In HDG discretizations, the coupled unknowns are single-valued traces introduced on the mesh skeleton, i.e., the faces, and the resulting matrix is substantially smaller and sparser compared to standard DG approaches. Once the trace unknowns are solved for, the usual DG unknowns can be recovered in an element-by-element fashion, completely independent of each other.

In the context of MHD simulations, the most common computational solution strategies for many years have been the use of explicit [2, 24] and partially implicit time integration methods such as implicit-explicit [25, 20, 10], semi-implicit [8, 4], and operator-splitting [22] techniques (see [9] for a recent review). With the exception of fully-explicit strategies, that due to numerical stability constraints must resolve the fastest component time-scales, all these time integration methods include an aspect of implicit integration. The implicitness of these approaches is used to enhance efficiency by removing stringent explicit time-scale constraints in the problem, either from diffusion or from fast-wave phenomena.

In addition to the challenges associated with robustly and efficiently dealing with the

---

\*Institute for Computational and Engineering Sciences, University of Texas at Austin, shannon@ices.utexas.edu

†Sandia National Laboratories, jnshadi@sandia.gov

‡Department of Aerospace Engineering and Engineering Mechanics, Institute for Computational and Engineering Sciences, University of Texas at Austin, tanbui@ices.utexas.edu

§Institute for Computational and Engineering Sciences, University of Texas at Austin, jeonghun@ices.utexas.edu

very large range of time scales inherent in the resistive MHD system, there are a number of spatial discretization issues that add considerable complexity to the numerical approximation of these systems. These include dealing with the dual saddle-point structure of the velocity-pressure  $(\mathbf{u}, p)$  interactions in the Navier-Stokes operators, and the satisfaction/enforcement of the solenoidal involution/constraint on the magnetic induction  $(\nabla \cdot \mathbf{b} = 0)$  in conjunction with the evolution of the induction equation for  $\mathbf{b}$ .

The current study presents an HDG formulation that deals with the double saddle-point structure of the linearized incompressible resistive MHD system and analyzes well-posedness. Numerical studies then demonstrate these results for nonlinear steady and unsteady problems using the linear solver as a sub-step in a fixed-point nonlinear solver. It is intended that in future work the linear HDG scheme will be employed as the fast-time-scale operator in a linear HDG-nonlinear DG implicit-explicit (IMEX) scheme.

The remainder of this paper is organized as follows. In section 2 we review the basic concepts of HDG. In section 3 we introduce an HDG scheme for a linearization of the resistive incompressible MHD equations, discuss ways to enforce the zero mean pressure condition, and provide well-posedness results and analysis for the scheme. In section 4 we present a nonlinear fixed point iteration that we use to solve the full nonlinear resistive incompressible MHD system using the linear scheme as substep. In section 5 we review (implicit) time integration methods suitable for evolving HDG schemes. In section 6 we present numerical results before concluding.

**2. A Brief Introduction to HDG.** In this section, we outline the basic concepts of HDG in the context of a general class of PDEs. Consider the abstract symmetric hyperbolic system of PDEs,

$$\nabla \cdot \mathbf{F}(\mathbf{U}) + \mathbf{C}\mathbf{U} := \sum_{k=1}^d \partial_k \mathbf{F}_k(\mathbf{U}) + \mathbf{C}\mathbf{U} = \mathbf{f} \quad \text{in } \Omega, \quad (2.1)$$

where  $\mathbf{F}_k(\mathbf{U}) = \mathbf{A}^k \mathbf{U}$ , with each  $\mathbf{A}^k$  symmetric, and  $\Omega \subset \mathbb{R}^d$ .

Toward solving Eq. 2.1 numerically, we partition the computational domain  $\Omega \subset \mathbb{R}^d$ , with boundary  $\partial\Omega$ , into disjoint elements  $K$  (simplices or quadrilaterals/hexahedra), and define  $\mathcal{T}$  as the collection of elements  $K$ . We define  $\partial\mathcal{T} := \{\partial K : K \in \mathcal{T}\}$  as the collection of element boundaries. For any  $K$ ,  $e = \partial K \cap \partial\Omega$  is a  $(d-1)$  dimensional boundary face if  $e$  has a nonzero  $d-1$  Lebesgue measure. For any distinct  $K^-$  and  $K^+$ ,  $e = \partial K^- \cap \partial K^+$  is an interior face if  $e$  has a nonzero  $d-1$  Lebesgue measure. The mesh skeleton,  $\mathcal{E}$ , is defined as the collection of all faces, boundary and interior.

We define  $(\cdot, \cdot)_K$  as the  $L^2$  inner product over element  $K$ , and  $(\cdot, \cdot)_{\mathcal{T}} := \sum_K (\cdot, \cdot)_K$ . Similarly, we define  $\langle \cdot, \cdot \rangle_{\partial K}$  as the  $L^2$  inner product over the boundary of element  $K$ , and  $\langle \cdot, \cdot \rangle_{\partial\mathcal{T}} := \sum_K \langle \cdot, \cdot \rangle_{\partial K}$ .

Formally, multiplying Eq. 2.1 by an elementwise continuous test function, integrating over  $\Omega$ , and integrating by parts, we have

$$-(\mathbf{F}(\mathbf{U}), \nabla \mathbf{v})_K + (\mathbf{C}\mathbf{U}, \mathbf{v})_K + \langle \mathbf{F}(\mathbf{U}) \cdot \mathbf{n}, \mathbf{v} \rangle_{\partial K} = (\mathbf{f}, \mathbf{v})_K \quad \forall K \in \mathcal{T}. \quad (2.2)$$

Replacing the boundary term by a single-valued flux that depends on the solution  $\mathbf{U}$  on each side of the interface,  $\mathbf{F}^* = \mathbf{F}^*(\mathbf{U}^-, \mathbf{U}^+)$ , we have DG scheme:

$$-(\mathbf{F}(\mathbf{U}), \nabla \mathbf{v})_K + (\mathbf{C}\mathbf{U}, \mathbf{v})_K + \langle \mathbf{F}^*(\mathbf{U}^-, \mathbf{U}^+), \mathbf{v} \rangle_{\partial K} = (\mathbf{f}, \mathbf{v})_K \quad \forall K \in \mathcal{T}. \quad (2.3)$$

The implicit DG scheme, Eq. 2.3, leads to a system where all the unknowns are globally coupled.

Instead, to construct an HDG scheme, introduce the trace unknown  $\hat{\mathbf{U}}$  and replace the flux on the boundary in Eq. 2.2 by a one sided HDG flux  $\hat{\mathbf{F}} = \hat{\mathbf{F}}(\mathbf{U}^-, \hat{\mathbf{U}})$ ,

$$-(\mathbf{F}(\mathbf{U}), \nabla \mathbf{v})_K + (\mathbf{C}\mathbf{U}, \mathbf{v})_K + \langle \hat{\mathbf{F}}(\mathbf{U}, \hat{\mathbf{U}}) \cdot \mathbf{n}, \mathbf{v} \rangle_{\partial K} = (\mathbf{f}, \mathbf{v})_K \quad \forall K \in \mathcal{T}, \quad (2.4)$$

and, to close the system, enforce that the normal flux is (weakly) continuous across element interfaces,

$$\langle \hat{\mathbf{F}}(\mathbf{U}, \hat{\mathbf{U}}) \cdot \mathbf{n}, \mu \rangle_{\partial \mathcal{T} \setminus \partial \Omega} = 0 \quad (2.5)$$

for facewise continuous test functions,  $\mu$ .

The HDG scheme comprises the local solver, Eq. 2.4, the transmission or conservation conditions, Eq. 2.5, and boundary conditions. We refer the reader to [3] for more details, including a framework for constructing HDG fluxes leading to well-posed *upwind* HDG schemes for PDEs of the form Eq. 2.1 with linear fluxes  $\mathbf{F}$ , and an extension to nonlinear fluxes.

For linear systems, the HDG scheme, Eqs. 2.4 and 2.5, gives rise to the following matrix equations, where  $\mathbb{U}$  represents the vector degrees of freedom of  $\mathbf{U}$ , and  $\hat{\mathbf{U}}$  represents the vector degrees of freedom of  $\hat{\mathbf{U}}$ ,

$$\begin{bmatrix} \mathbb{A} & \mathbb{C} \\ \mathbb{B} & \mathbb{E} \end{bmatrix}^* \begin{Bmatrix} \mathbb{U} \\ \hat{\mathbf{U}} \end{Bmatrix} = \begin{Bmatrix} \mathbb{F}_l \\ \mathbb{F}_c \end{Bmatrix}. \quad (2.6)$$

Here, the subscripts  $l$  and  $c$  stand for local and conservation, respectively.

The power of HDG comes from the following.

- The HDG flux is one-sided, i.e., for a given element, the flux depends only on the solution in that element and the neighboring skeleton faces. Together with the fact that the discontinuous basis functions are local to one element,  $\mathbb{A}$  is *block diagonal*.
- If the local solver  $(\hat{\mathbf{U}}, \mathbf{f}) \mapsto \mathbf{U}$  given by Eq. 2.4 is well-posed, then  $\mathbb{A}$  is *invertible*.

Therefore, we can easily eliminate  $\mathbb{U}$  from Eq. 2.6 by a static condensation procedure, and write

$$\mathbb{U} = \mathbb{A}^{-1} [\mathbb{F}_l - \mathbb{C}\hat{\mathbf{U}}]. \quad (2.7)$$

The global system, Eq. 2.6, then reduces to

$$\underbrace{(\mathbb{E} - \mathbb{B}[\mathbb{A}]^{-1}\mathbb{C})}_{\mathbb{K}} \hat{\mathbf{U}} = \underbrace{\mathbb{F}_g - \mathbb{B}[\mathbb{A}]^{-1}\mathbb{F}_l}_{\mathbb{F}}. \quad (2.8)$$

In practice,  $\mathbb{K}$  and  $\mathbb{F}$  are formed by a local assembly procedure,  $\hat{\mathbf{U}}$  is solved for from the reduced global system, Eq. 2.8, and then  $\mathbb{U}$  is recovered in an element by element fashion from Eq. 2.7.

**3. An HDG Scheme for Resistive, Incompressible MHD.** We consider the resistive, incompressible MHD equations written in non-dimensionalized form.

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} + \nabla p - \frac{1}{\text{Re}} \Delta \mathbf{u} - \kappa (\nabla \times \mathbf{b}) \times \mathbf{b} = \mathbf{0} \quad (\text{momentum}) \quad (3.1a)$$

$$\nabla \cdot \mathbf{u} = 0 \quad (\text{mass}) \quad (3.1b)$$

$$\kappa \frac{\partial \mathbf{b}}{\partial t} + \nabla r - \kappa \nabla \times (\mathbf{u} \times \mathbf{b}) + \frac{\kappa}{\text{Rm}} \nabla \times (\nabla \times \mathbf{b}) = \mathbf{0} \quad (\text{induction}) \quad (3.1c)$$

$$\nabla \cdot \mathbf{b} = 0 \quad (\text{no monopole}) \quad (3.1d)$$

Here  $\mathbf{u}$  is velocity of the fluid,  $\mathbf{b}$  is magnetic induction,  $\text{Re} := \frac{\rho u_0 l_0}{\mu}$  is the Reynolds number,  $\text{Rm} := \frac{\mu_0 u_0 l_0}{\eta}$  is magnetic Reynolds number, and interaction parameter  $\kappa := \frac{\text{Ha}^2}{\text{ReRm}}$ , where  $\text{Ha} := \frac{b_0 l_0}{\sqrt{\mu\eta}}$  is the Hartmann number. See [13] for more details on dimensionless parameters related to MHD.  $l_0$ ,  $u_0$ , and  $b_0$  are some characteristic length, velocity, and magnetic field, respectively.  $r$  is a scalar potential introduced to enforce the solenoidal constraint ( $\nabla \cdot \mathbf{b} = 0$ ).

(Note: to recover the full dimensional form of the equations, we make the substitutions  $\text{Re} \rightarrow \frac{\rho}{\mu}$ ,  $\text{Rm} \rightarrow \frac{\mu_0}{\eta}$ ,  $\kappa \rightarrow \frac{1}{\rho\mu_0}$ , and  $p \rightarrow \frac{p}{\rho}$ ).

In this paper, we consider the Dirichlet boundary conditions  $\mathbf{u} = \mathbf{u}_D$ ,  $\mathbf{b}^t := -\mathbf{n} \times (\mathbf{n} \times \mathbf{b}) = \mathbf{h}_D$ , and  $r = 0$  on  $\partial\Omega$ , where we have defined the tangent component of a vector  $\mathbf{a}$  as  $\mathbf{a}^t := -\mathbf{n} \times (\mathbf{n} \times \mathbf{a})$ . For uniqueness of the pressure, we require the zero average pressure constraint

$$(p, 1)_\Omega = 0. \quad (3.2)$$

Additionally, we require the compatibility condition

$$\langle \mathbf{u}_D, 1 \rangle_{\partial\Omega} = 0. \quad (3.3)$$

**3.1. An HDG Scheme.** We introduce the following discontinuous piecewise polynomial approximation spaces.

$$\begin{aligned} \mathbf{G}_h &:= \left\{ \mathbf{G} \in [L^2(\mathcal{T})]^{d \times d} : \mathbf{G}|_K \in [P^k(K)]^{d \times d} \ \forall K \in \mathcal{T} \right\}, \\ \mathbf{V}_h &:= \left\{ \mathbf{v} \in [L^2(\mathcal{T})]^d : \mathbf{v}|_K \in [P^k(K)]^d \ \forall K \in \mathcal{T} \right\}, \\ Q_h &:= \left\{ q \in L^2(\mathcal{T}) : q|_K \in P^k(K) \ \forall K \in \mathcal{T} \right\}, \\ \mathbf{H}_h &:= \left\{ \mathbf{H} \in [L^2(\mathcal{T})]^{\tilde{d}} : \mathbf{H}|_K \in [P^k(K)]^{\tilde{d}} \ \forall K \in \mathcal{T} \right\}, \\ \mathbf{C}_h &:= \left\{ \mathbf{c} \in [L^2(\mathcal{T})]^d : \mathbf{c}|_K \in [P^k(K)]^d \ \forall K \in \mathcal{T} \right\}, \\ S_h &:= \left\{ s \in L^2(\mathcal{T}) : s|_K \in P^k(K) \ \forall K \in \mathcal{T} \right\}, \\ \mathbf{M}_h &:= \left\{ \boldsymbol{\mu} \in [L^2(\mathcal{E})]^d : \boldsymbol{\mu}|_e \in [P^k(e)]^d \ \forall e \in \mathcal{E} \right\}, \\ \boldsymbol{\Lambda}_h^t &:= \left\{ \boldsymbol{\lambda} \in [L^2(\mathcal{E})]^d : \boldsymbol{\lambda}|_e \in [P^k(e)]^d, \boldsymbol{\lambda} \cdot \mathbf{n}_e = 0 \ \forall e \in \mathcal{E} \right\}, \\ \Gamma_h &:= \left\{ \gamma \in [L^2(\mathcal{E})]^d : \gamma|_e \in [P^k(e)]^d \ \forall e \in \mathcal{E} \right\}, \end{aligned}$$

where  $\tilde{d} = 3$  if  $d = 3$ , and  $\tilde{d} = 1$  if  $d = 2$ , and  $\mathbf{n}_e$  is a unit vector normal to face  $e$ .

Inspired by the procedure outlined in [3], we linearize Eq. 3.1 about some velocity  $\mathbf{w}$  and some magnetic field  $\mathbf{d}$ , and introduce auxiliary variables to write the system as a first order hyperbolic system:

$$\text{Re} \mathbf{L} - \nabla \mathbf{u} = \mathbf{0}, \quad (3.4a)$$

$$\frac{\partial \mathbf{u}}{\partial t} - \nabla \cdot \mathbf{L} + \nabla p + (\mathbf{w} \cdot \nabla) \mathbf{u} - \kappa (\nabla \times \mathbf{b}) \times \mathbf{d} = \mathbf{g}, \quad (3.4b)$$

$$\nabla \cdot \mathbf{u} = 0, \quad (3.4c)$$

$$\frac{\text{Rm}}{\kappa} \mathbf{J} - \nabla \times \mathbf{b} = \mathbf{0}, \quad (3.4d)$$

$$\kappa \frac{\partial \mathbf{b}}{\partial t} + \nabla \times \mathbf{J} + \nabla r - \kappa \nabla \times (\mathbf{u} \times \mathbf{d}) = \mathbf{f}, \quad (3.4e)$$

$$\nabla \cdot \mathbf{b} = 0. \quad (3.4f)$$

Next, we multiply Eqs. 3.4 by test functions  $(\mathbf{G}, \mathbf{v}, q, \mathbf{H}, \mathbf{c}, s)$  integrate by parts all terms with a spatial derivative, and replace the resulting boundary terms by the normal component of the HDG flux, upwind in nature,

$$\operatorname{Re}(\mathbf{L}, \mathbf{G})_K + (\mathbf{u}, \nabla \cdot \mathbf{G})_K + \left\langle \left( \hat{\mathbf{F}} \cdot \mathbf{n} \right)_1, \mathbf{G} \right\rangle_{\partial K} = 0, \quad (3.5a)$$

$$\begin{aligned} \left( \frac{\partial \mathbf{u}}{\partial t}, \mathbf{v} \right)_K + (\mathbf{L}, \nabla \mathbf{v})_K - (p, \nabla \cdot \mathbf{v})_K - (\mathbf{u} \otimes \mathbf{w}, \nabla \mathbf{v})_K \\ + \kappa (\mathbf{b}, \nabla \times (\mathbf{v} \times \mathbf{d}))_K + \left\langle \left( \hat{\mathbf{F}} \cdot \mathbf{n} \right)_2, \mathbf{v} \right\rangle_{\partial K} = (\mathbf{g}, \mathbf{v})_K, \end{aligned} \quad (3.5b)$$

$$-(\mathbf{u}, \nabla q)_K + \left\langle \left( \hat{\mathbf{F}} \cdot \mathbf{n} \right)_3, q \right\rangle_{\partial K} = 0, \quad (3.5c)$$

$$\frac{\operatorname{Rm}}{\kappa} (\mathbf{J}, \mathbf{H})_K - (\mathbf{b}, \nabla \times \mathbf{H})_K + \left\langle \left( \hat{\mathbf{F}} \cdot \mathbf{n} \right)_4, \mathbf{H} \right\rangle_{\partial K} = 0, \quad (3.5d)$$

$$\begin{aligned} \kappa \left( \frac{\partial \mathbf{b}}{\partial t}, \mathbf{c} \right)_K + (\mathbf{J}, \nabla \times \mathbf{c})_K - (r, \nabla \cdot \mathbf{c})_K \\ - \kappa (\mathbf{u}, \mathbf{d} \times (\nabla \times \mathbf{c}))_K + \left\langle \left( \hat{\mathbf{F}} \cdot \mathbf{n} \right)_5, \mathbf{c} \right\rangle_{\partial K} = (\mathbf{f}, \mathbf{c})_K, \end{aligned} \quad (3.5e)$$

$$-(\mathbf{b}, \nabla s)_K + \left\langle \left( \hat{\mathbf{F}} \cdot \mathbf{n} \right)_6, s \right\rangle_{\partial K} = 0. \quad (3.5f)$$

The normal HDG flux is given by

$$\hat{\mathbf{F}} \cdot \mathbf{n} := \begin{bmatrix} -\hat{\mathbf{u}} \otimes \mathbf{n} \\ -\mathbf{L}\mathbf{n} + (\mathbf{w} \cdot \mathbf{n})\mathbf{u} + p\mathbf{n} + \frac{1}{2}\kappa\mathbf{d} \times \left( \mathbf{n} \times (\mathbf{b}^t + \hat{\mathbf{b}}^t) \right) + \alpha_1 (\mathbf{u} - \hat{\mathbf{u}}) \\ \hat{\mathbf{u}} \cdot \mathbf{n} \\ -\mathbf{n} \times \hat{\mathbf{b}}^t \\ \mathbf{n} \times \mathbf{J} + \hat{r}\mathbf{n} - \frac{1}{2}\kappa\mathbf{n} \times ((\mathbf{u} + \hat{\mathbf{u}}) \times \mathbf{d}) + \alpha_2 (\mathbf{b}^t - \hat{\mathbf{b}}^t) \\ \mathbf{b} \cdot \mathbf{n} + \alpha_3 (r - \hat{r}) \end{bmatrix}, \quad (3.6)$$

where  $\alpha_1$ ,  $\alpha_2$ , and  $\alpha_3$  are constants specified later. To define the conservation conditions, realize that the jump of the 1st, 3rd, and 4th components of the normal HDG flux is automatically zero, because the trace unknowns  $\hat{\mathbf{u}}$  and  $\hat{\mathbf{b}}^t$  are single valued on the skeleton, and the normal vector across the interface is equal and opposite. Therefore, the conservation conditions reduce to

$$\left\langle \left( \hat{\mathbf{F}} \cdot \mathbf{n} \right)_2, \boldsymbol{\mu} \right\rangle_{\partial T \setminus \partial \Omega} = 0, \quad (3.7a)$$

$$\left\langle \left( \hat{\mathbf{F}} \cdot \mathbf{n} \right)_5, \boldsymbol{\lambda}^t \right\rangle_{\partial T \setminus \partial \Omega} = 0, \quad (3.7b)$$

$$\left\langle \left( \hat{\mathbf{F}} \cdot \mathbf{n} \right)_6, \gamma \right\rangle_{\partial T \setminus \partial \Omega} = 0. \quad (3.7c)$$

The Dirichlet boundary conditions are enforced through the trace unknowns by  $L^2$  projection,

$$\langle \hat{\mathbf{u}}, \boldsymbol{\mu} \rangle_{\partial \Omega} = \langle \hat{\mathbf{u}}, \boldsymbol{\mu} \rangle_{\partial \Omega}, \quad (3.8a)$$

$$\langle \hat{\mathbf{b}}^t, \boldsymbol{\lambda}^t \rangle_{\partial \Omega} = \langle \mathbf{h}_D, \boldsymbol{\lambda}^t \rangle_{\partial \Omega}, \quad (3.8b)$$

$$\langle \hat{r}, \gamma \rangle_{\partial \Omega} = 0. \quad (3.8c)$$

As will be discussed below, *with some additional conditions to enforce the zero average pressure condition*, Eq. 3.2, the following HDG scheme is well-posed: find  $(\mathbf{L}, \mathbf{u}, p, \mathbf{J}, \mathbf{b}, r, \hat{\mathbf{u}}, \hat{\mathbf{b}}, \hat{r})$  in  $\mathbf{G}_h \times \mathbf{V}_h \times Q_h \times \mathbf{H}_h \times \mathbf{C}_h \times S_h \times \mathbf{M}_h \times \Lambda_h^t \times \Gamma_h$  such that Eqs. 3.5, 3.7, and 3.8 hold true for all  $(\mathbf{G}, \mathbf{v}, q, \mathbf{H}, \mathbf{c}, s, \boldsymbol{\mu}, \boldsymbol{\lambda}^t, \gamma)$  in  $\mathbf{G}_h \times \mathbf{V}_h \times Q_h \times \mathbf{H}_h \times \mathbf{C}_h \times S_h \times \mathbf{M}_h \times \Lambda_h^t \times \Gamma_h$ .

Note that despite the fact that (in 3D) with the introduction of the auxiliary variables  $\mathbf{L}$  and  $\mathbf{J}$  we have 20 scalar volume unknowns, we have only 6 scalar global trace variables (velocity, tangent magnetic field, and scalar Lagrange multiplier), which is less than the 8 primal scalar variables from the original PDE.

**3.2. Well-posedness.** Linear HDG schemes that are constructed through the framework in [3], as well as other HDG schemes, are well suited for a simple energy analysis to prove existence and uniqueness of a numerical solution. Since we have a linear, finite dimensional, square system, existence is equivalent to uniqueness. For the HDG scheme, we prove existence/uniqueness by setting the forcing terms to zero, and showing that we recover the trivial solution, proving that the linear operator is invertible. An example of application of this energy analysis for an HDG scheme for Stokes flow is given in [14].

**3.2.1. Enforcing zero average pressure.** Before proving well-posedness, we briefly discuss ways to enforce the zero average pressure condition, Eq. 3.2 on the numerical scheme (which will be necessary for well-posedness). Nguyen, Peraire, and Cockburn discuss two methods of doing this; one direct way and one iterative way [14, 17]. Ultimately more efficient (for problems with a fine mesh) is the iterative method, an Augmented Lagrangian approach.

The Augmented Lagrangian approach involves adding an artificial time derivative term  $\left(\frac{\partial p}{\partial \tau}, q\right)_K$  to Eq. 3.5c. Discretizing the time derivative with implicit Euler and giving an initial condition of  $p_0 = 0$ , it turns out that solution converges to the solution of the original scheme (without the artificial time derivative) with the pressure satisfying the zero average condition.

For this study, we consider a direct approach. We introduce a new global scalar variable,  $\rho_K$  that is constant over each element, and represents the integral of the pressure over each element. Next, we modify the local equation, Eq. 3.5c, to read

$$-(\mathbf{u}, \nabla q)_K + \langle \hat{\mathbf{u}} \cdot \mathbf{n}, q \rangle_{\partial K} + (p, 1)_K = \rho_K \quad (3.9)$$

and introduce the global equation

$$\langle \hat{\mathbf{u}} \cdot \mathbf{n}, 1 \rangle_{\partial K} + \sum_K \rho_K = 0. \quad (3.10)$$

Note that our system is still square (we have added as many equations as unknowns to the HDG system).

Also note that Eqs. 3.9 and 3.10 imply that Eq. 3.5c holds, that  $(p, 1)_K = \rho_K$  for each element, and that  $\sum_K \rho_K = 0$ , satisfying the zero average pressure condition. Indeed, summing Eq. 3.10 over all elements, and using the compatibility condition, Eq. 3.3, we recover  $\sum_K \rho_K = 0$  and  $\langle \hat{\mathbf{u}} \cdot \mathbf{n}, 1 \rangle_{\partial K} = 0$  for each  $K$ . Then in Eq. 3.9, setting  $q = 1$  on  $K$ , we recover  $(p, 1)_K = \rho_K$ , and subsequently recover Eq. 3.5c.

Finally, our local HDG system consists of Eqs. 3.5a, 3.5b, 3.9, 3.5d, 3.5e, and 3.5f. The conservation conditions, Eq. 3.7, the boundary conditions, Eq. 3.8, and the additional equations, Eq. 3.10, complete the scheme.

Note that even though  $p$  and  $r$  are analogous in the original PDE, Eq. 3.1, the same care does not need to be taken to ensure uniqueness of  $r$  as does for  $p$ . In short, this is due

to the boundary conditions studied in this paper. Here, the boundary condition on  $r$  leads to the inclusion of  $\hat{r}$  (through which the boundary condition is enforced) in the flux, Eq. 3.6. This turns out to be sufficient for the uniqueness of the numerical solution for  $r$ .

**3.2.2. Well-posedness of the local solver.** Recall that an important aspect of an HDG scheme is that the local solver is well-posed, which is equivalent to saying that we are able to perform static condensation to have a reduced global system in terms of the trace variables only.

We first consider the steady-state problem.

**THEOREM 3.1.** *Assume that  $\alpha_1 > \frac{\mathbf{w} \cdot \mathbf{n}}{2}$ ,  $\alpha_2 > 0$ ,  $\alpha_3 > 0$ , and that  $\nabla \cdot \mathbf{w} = 0$ . Then the steady-state local solver  $(\mathbf{g}, \mathbf{f}, \hat{\mathbf{u}}, \hat{\mathbf{b}}^t, \hat{r}, \rho) \mapsto (\mathbf{L}, \mathbf{u}, p, \mathbf{J}, \mathbf{b}, r)$  defined by Eqs. 3.5a, 3.5b, 3.9, 3.5d, 3.5e, and 3.5f, with  $\frac{\partial}{\partial t} = 0$ , is well-posed.*

*Proof.* Setting  $(\mathbf{g}, \mathbf{f}, \hat{\mathbf{u}}, \hat{\mathbf{b}}^t, \hat{r}, \rho) = \mathbf{0}$ , setting  $(\mathbf{G}, \mathbf{v}, q, \mathbf{H}, \mathbf{c}, s) = (\mathbf{L}, \mathbf{u}, p, \mathbf{J}, \mathbf{b}, r)$ , summing the local equations, and performing some integrations by parts, we arrive at an energy identity,

$$\begin{aligned} \operatorname{Re}(\mathbf{L}, \mathbf{L})_K + \left\langle \left( \alpha_1 + \frac{\mathbf{w} \cdot \mathbf{n}}{2} \right) \mathbf{u}, \mathbf{u} \right\rangle_{\partial K} \\ + \frac{\operatorname{Rm}}{\kappa}(\mathbf{J}, \mathbf{J})_K + \langle \alpha_2 \mathbf{b}^t, \mathbf{b}^t \rangle_{\partial K} + \langle \alpha_3 r, r \rangle_{\partial K} = 0. \end{aligned} \quad (3.11)$$

From Eq. 3.11, we conclude that  $\mathbf{L} = \mathbf{0}$  and  $\mathbf{J} = \mathbf{0}$  on  $K$ , and that  $\mathbf{u} = \mathbf{0}$ ,  $\mathbf{b}^t = \mathbf{0}$ , and  $r = 0$  on  $\partial K$ . Eq. 3.5a reduces to  $(\nabla \mathbf{u}, \mathbf{G})_K = 0 \forall \mathbf{G}$  which implies  $\mathbf{u}$  is constant in  $K$ , and therefore with the energy identity that  $\mathbf{u} = \mathbf{0}$  in  $K$ .

Eq. 3.5d and Eq. 3.5f reduce to  $(\nabla \times \mathbf{b}, \mathbf{H})_K = 0 \forall \mathbf{H}$  and  $(\nabla \cdot \mathbf{b}, s)_K = 0 \forall s$ , respectively. Therefore  $\nabla \times \mathbf{b} = \mathbf{0}$  and  $\nabla \cdot \mathbf{b} = 0$ . Lemma 3.4 in [7] tells us that for a function in  $H(\operatorname{curl})_0 \cap H(\operatorname{div})$ , that  $\|\mathbf{b}\|_{L^2} \leq C(\|\nabla \times \mathbf{b}\|_{L^2} + \|\nabla \cdot \mathbf{b}\|_{L^2})$ . Since  $\mathbf{b}^t = \mathbf{0}$  on  $\partial K$ , this lemma applies, and therefore  $\mathbf{b} = \mathbf{0}$  in  $K$ .

Eq. 3.5e reduces to  $(\nabla r, \mathbf{c})_K = 0 \forall \mathbf{c}$  which implies  $r$  is constant in  $K$ , and therefore with the energy identity that  $r = 0$  in  $K$ . Eq. 3.5b reduces to  $(\nabla p, \mathbf{v})_K = 0 \forall \mathbf{v}$  which implies  $p$  is constant in  $K$ . Finally, Eq. 3.9 reduces to  $(p, 1)_K = 0$ , which gives  $p = 0$  in  $K$ .  $\square$

Note that the proof above is valid for the *equal order polynomial* spaces that we have defined in Section 3.1. More generally, the proof above holds for discrete spaces that satisfy  $\nabla \mathbf{V}_h(K) \subset \mathbf{G}_h(K)$ ,  $\nabla \times \mathbf{C}_h(K) \subset \mathbf{J}_h(K)$ ,  $\nabla \cdot \mathbf{C}_h(K) \subset S_h(K)$ ,  $\nabla S_h(K) \subset \mathbf{C}_h(K)$ , and  $\nabla Q_h(K) \subset \mathbf{V}_h(K)$ .

Semidiscrete stability for the unsteady problem can be proven in a similar manner, now with initial conditions  $\mathbf{b}_0$  and  $\mathbf{u}_0$  included in the forcing terms.

**3.2.3. Well-posedness of the global solver.** We state, without proof, the result for the well-posedness of the steady-state global solver. The proof will appear in [12]. Similar proofs can be found in [3, 14, 16].

**THEOREM 3.2.** *Assume that  $\alpha_1 > \frac{\mathbf{w} \cdot \mathbf{n}}{2}$ ,  $\alpha_2 > 0$ ,  $\alpha_3 > 0$ , and that  $\nabla \cdot \mathbf{w} = 0$ . Also assume that  $\mathbf{w} \in H(\operatorname{div}, \Omega)$ , that  $\mathbf{d} \in H(\operatorname{div}, \Omega) \cap H(\operatorname{curl}, \Omega)$ , and that  $\Omega$  is simply connected with one component to the boundary [7]. Then the steady-state global solver  $(\mathbf{g}, \mathbf{f}, \mathbf{u}_D, \mathbf{h}_D) \mapsto (\hat{\mathbf{u}}, \hat{\mathbf{b}}^t, \hat{r}, \rho, \mathbf{L}, \mathbf{u}, p, \mathbf{J}, \mathbf{b}, r)$  defined by the HDG scheme, Eqs. 3.7, 3.8, 3.10, 3.5a, 3.5b, 3.9, 3.5d, 3.5e, and 3.5f, with  $\frac{\partial}{\partial t} = 0$ , is well-posed.*

Again, semidiscrete stability for the unsteady problem can be proven in a similar manner, now with initial conditions  $\mathbf{b}_0$  and  $\mathbf{u}_0$  included in the forcing terms.



**4. A Nonlinear Solver.** In this work, we employ the linear HDG scheme, Eqs. 3.5a, 3.5b, 3.9, 3.5d, 3.5e, 3.5f, 3.7, 3.8, and 3.10, in a fixed point Picard iteration to find a numerical solution to the nonlinear problem, Eq. 3.1. That is, at iteration count  $k + 1$ , we set  $(\mathbf{w}, \mathbf{d})$  to the solution  $(\mathbf{u}, \mathbf{b})$  at iteration  $k$  and solve the linear system. We stop iterating when we reach certain stopping criteria. See [23] for suitable stopping criteria.

For the time dependent problem, numerical experiments have shown that the Picard iteration is robust with respect to the time step size for various Runge-Kutta time stepping schemes. For steady-state problems, we use a pseudotime step approach, reintroducing the time derivatives and taking large steps with implicit Euler to reach steady state.

**5. Time Integration.** For unsteady HDG, since  $\mathbf{u}$  and  $\mathbf{b}$  have time derivatives and the remaining variables do not, the discretization gives rise to a differential algebraic system of index 1 [15]. For these systems, implicit time stepping schemes such as BDF and diagonally implicit Runge-Kutta (DIRK) are suitable.

In this work, we consider DIRK schemes. Briefly, an  $s$ -stage DIRK scheme involves  $s$  implicit solves for the intermediate-stage values, followed by an explicit update to compute the solution at  $t^{n+1}$ . At each stage, we apply the Picard linearization, which results in linear systems that still take the form of Eq. 2.6, where we solve for all variables in the usual way. Then, in the explicit update step of the DIRK scheme, we compute  $(\mathbf{u}^{n+1}, \mathbf{b}^{n+1})$  from the intermediate-stage values of  $\mathbf{u}$  and  $\mathbf{b}$ . Finally, because of the nature of unsteady HDG scheme, in order to calculate  $(\mathbf{L}^{n+1}, p^{n+1}, \mathbf{J}^{n+1}, r^{n+1}, \hat{\mathbf{u}}^{n+1}, \hat{\mathbf{b}}^{t,n+1}, \hat{r}^{n+1}, \rho^{n+1})$  we must perform an additional implicit solve. Note that we can advance in time without performing this additional solve, so we only need to perform this additional implicit solve at time steps of interest. See [15] for details.

The class of *stiffly-accurate* DIRK methods has the desirable property (among other desirable properties) that the final intermediate stage coincides the next time step. Therefore, when using stiffly-accurate DIRK methods, the additional implicit step described above is never necessary. In the numerical simulations below, we employ 2 stage, 2nd-order and 3 stage, 3rd-order stiffly-accurate DIRK methods from [1], and 5 stage, 4th-order stiffly-accurate DIRK method given in Table 22 of [11]. In the remainder of this paper, we refer to these methods as DIRK(2,2), DIRK(3,3), and DIRK(5,4), respectively.

## 6. Numerical Results.

**6.1. Hartmann Flow.** As a steady-state verification problem, we consider Hartmann flow, studied in [23]. Consider a conducting incompressible fluid (liquid metal, for example) in a domain  $[-\infty, \infty] \times [-l_0, l_0] \times [-\infty, \infty]$  bounded by infinite parallel plates. The fluid is subject to a uniform pressure gradient  $G := -\frac{\partial p}{\partial x}$  in the  $x$  direction, and a uniform external magnetic field  $b_0$  in the  $y$  direction. If we consider no-slip boundary conditions, the resulting flow pattern is known as *Hartmann flow* and is described by an analytical solution that is 1D in nature. If the infinite parallel plates are perfectly insulating, the analytical solution is given by  $\mathbf{u} = \left( \frac{Gl_0^2}{\mu \text{Ha} \tanh(\text{Ha})} \left( 1 - \frac{\cosh(\frac{y}{l_0} \text{Ha})}{\cosh(\text{Ha})} \right), 0, 0 \right)$ , and  $\mathbf{b} = \left( \frac{Gl_0 \mu_0}{b_0} \left( \frac{\sinh(\frac{y}{l_0} \text{Ha})}{\sinh(\text{Ha})} - \frac{y}{l_0} \right), b_0, 0 \right)$ .

For this study, we consider a 2D domain  $[0, 10] \times [-l_0, l_0]$ . We choose  $l_0 = \rho = \mu = \mu_0 = \eta = 1$ ,  $G = 50$ , and vary  $b_0$  to produce velocity and magnetic field profiles with steeper and steeper gradients. Periodic boundary conditions are applied in the  $x$  direction. Zero initial conditions were given for the pseudo time stepping.

Figure 6.1 shows the  $x$  velocity and  $x$  magnetic field profiles for various values of  $b_0$ . In each case, 4th-order basis functions are used, with a mesh of  $1 \times 8$  elements strategically clustered toward the edge of the domain in order to capture the sharp features. Without this clustering, oscillations can appear on the element interior. In future work, we will consider



stabilization/shock capturing [6].

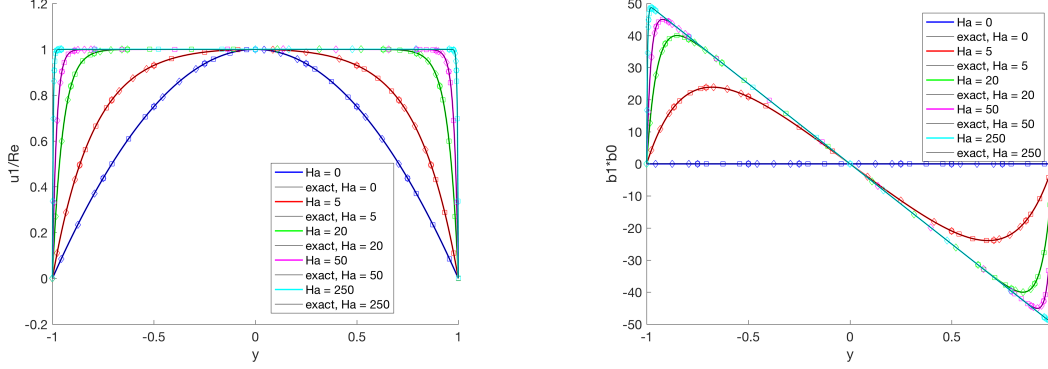


Fig. 6.1: Hartmann flow problem: varying the external magnetic field  $b_0$  and keeping all other parameters constant. On the left is the  $x$  velocity profile (scaled). On the right is the  $x$  magnetic field (scaled). Polynomial order  $p = 4$ , mesh is  $1 \times 8$  elements, strategically clustered toward  $y = \pm 1$ .

Figure 6.2 demonstrates the optimal spatial convergence of the velocity and magnetic field. A mesh of  $1 \times K$  uniformly spaced elements is used.

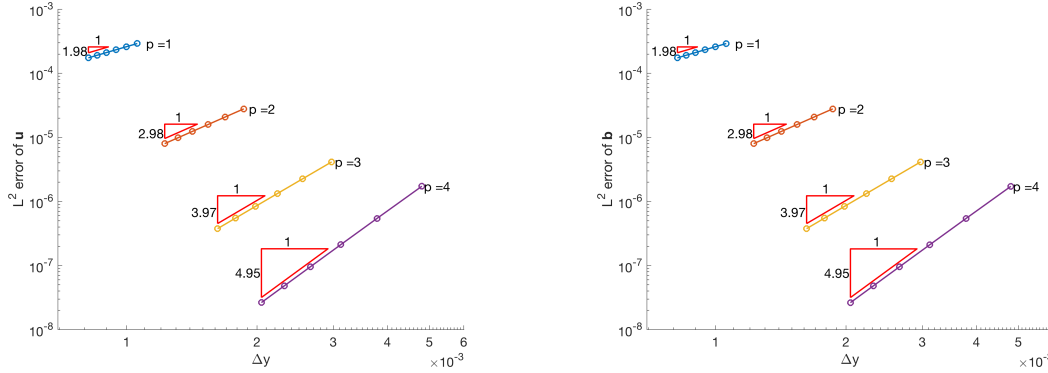


Fig. 6.2: Hartmann flow problem: spatial convergence of velocity and magnetic field, uniformly spaced mesh of  $1 \times K$  elements.

**6.2. Rayleigh Flow.** As a time dependent verification problem, we consider a Rayleigh flow problem with Alfvén wave propagation, studied in [23]. Consider a conducting incompressible fluid in a semi-infinite domain  $[-\infty, \infty] \times [0, \infty] \times [-\infty, \infty]$  bounded by an infinite parallel insulating plate, subject to a uniform external magnetic field  $b_0$  in the  $y$  direction. The fluid is initially at rest, and the plate is instantaneously set into motion at a velocity  $U$  in the  $x$  direction. The resulting flow pattern is described by an analytical solution that is 1D in nature,  $\mathbf{u} = \left( \frac{U}{4} \left[ \left( 1 + e^{-\frac{A_0 y}{d}} \right) \operatorname{erfc} \left( \frac{y - A_0 t}{2\sqrt{dt}} \right) + \left( 1 + e^{\frac{A_0 y}{d}} \right) \operatorname{erfc} \left( \frac{y + A_0 t}{2\sqrt{dt}} \right) \right], 0, 0 \right)$ , and  $\mathbf{b} = \left( -\frac{U\sqrt{\mu\rho}}{4} \left( 1 - e^{-\frac{A_0 y}{d}} \right) \left[ \operatorname{erfc} \left( \frac{y - A_0 t}{2\sqrt{dt}} \right) + e^{\frac{A_0 y}{d}} \operatorname{erfc} \left( \frac{y + A_0 t}{2\sqrt{dt}} \right) \right], b_0, 0 \right)$ .

For this study, we consider a 2D domain  $[0, 5] \times [0, l_0]$ , with periodic boundary conditions in the  $x$  direction, a no-flow, insulating boundary condition at  $y = l_0$ , and a no-slip, insulating boundary condition at  $y = 0$ . We choose  $\rho = \mu = \mu_0 = \eta = 1$ , and  $U = 1$ .

As a first test, we choose  $b_0 = 10$ , and  $l_0 = 5$ . Figure 6.3 shows the  $x$  velocity and  $x$  magnetic field profiles for various times up to  $t = 0.1$ . Fourth-order basis functions are used, with a mesh of  $1 \times 32$  elements strategically clustered. DIRK(5,4) time stepping was used with a time step of  $\Delta t = 5 \times 10^{-3}$ .

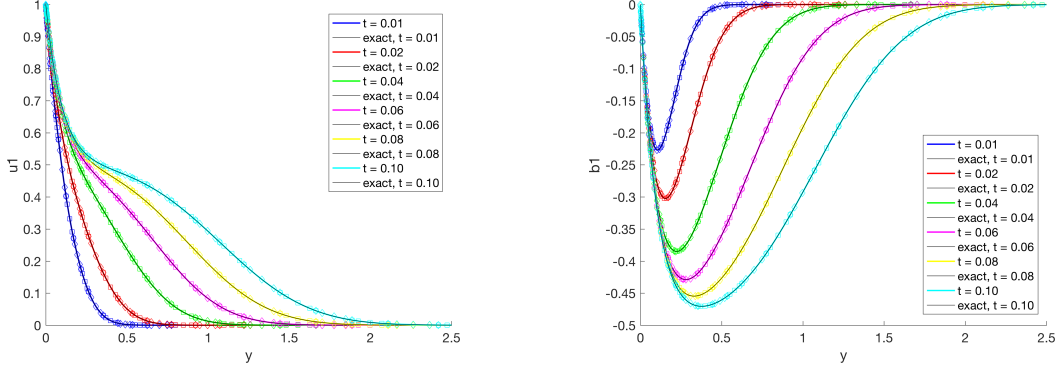


Fig. 6.3: Rayleigh flow problem: On the left is the  $x$  velocity profile. On the right is the  $x$  magnetic field profile. Calculated on a domain of  $[0, 5] \times [0, 5]$ , with a mesh of  $1 \times 32$  4th-order elements, strategically clustered. Time stepping is DIRK(5,4) with  $\Delta t = 0.5 \times 10^{-3}$ .

As a more demanding test of the robustness of the scheme, we apply an external magnetic field of  $b_0 = 100$ , and set  $l_0 = 50$  to accommodate the faster speed of the propagating wave while reasonably representing a semi-infinite domain. Figure 6.4 shows the  $x$  velocity and  $x$  magnetic field profiles for various times up to  $t = 0.1$ . Fourth-order basis functions are used, with a mesh of  $1 \times 288$  elements strategically clustered to capture the very steep gradient around  $y = 0$ , and the propagating wave, which become less steep as it travels away from  $y = 0$ . DIRK(5,4) time stepping was used with a time step of  $\Delta t = 5 \times 10^{-4}$ .

Figure 6.5 demonstrates the optimal temporal convergence of the velocity and magnetic field for stiffly-accurate, L-stable, diagonally-implicit Runge-Kutta methods of order 1 through 4. We set  $b_0 = 10$  and  $l_0 = 5$ . A mesh of  $1 \times 32$  strategically clustered 7th-order elements is used, and the system is evolved to  $t = 0.01$ .

Figure 6.6 demonstrates the optimal spatial convergence of the velocity and magnetic field. Again  $b_0 = 10$  and  $l_0 = 5$ . A mesh of  $1 \times K$  uniformly spaced elements is used. DIRK(5,4) time stepping was used to step to a final time of 0.01, with a time step of  $\Delta t = 5 \times 10^{-4}$ .

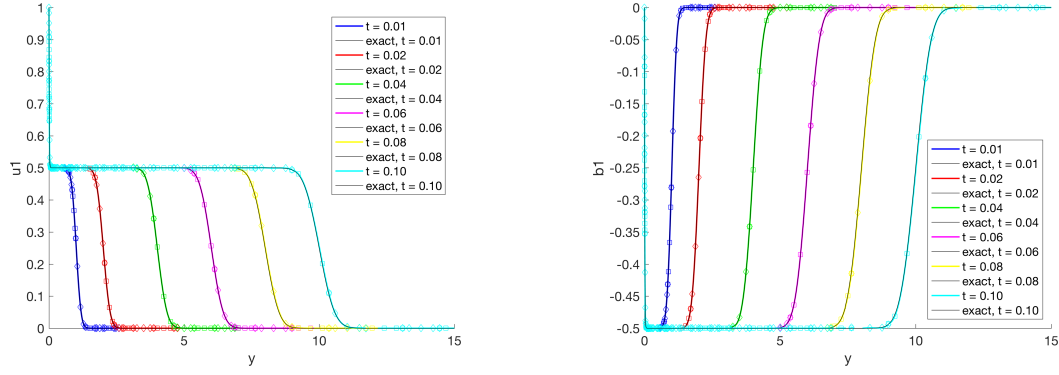


Fig. 6.4: Rayleigh flow problem: On the left is the  $x$  velocity profile. On the right is the  $x$  magnetic field profile. Calculated on a domain of  $[0, 5] \times [0, 50]$ , with a mesh of  $1 \times 288$  4th order elements elements, strategically clustered. Time stepping is DIRK(5,4) with  $\Delta t = 0.5 \times 10^{-4}$ .

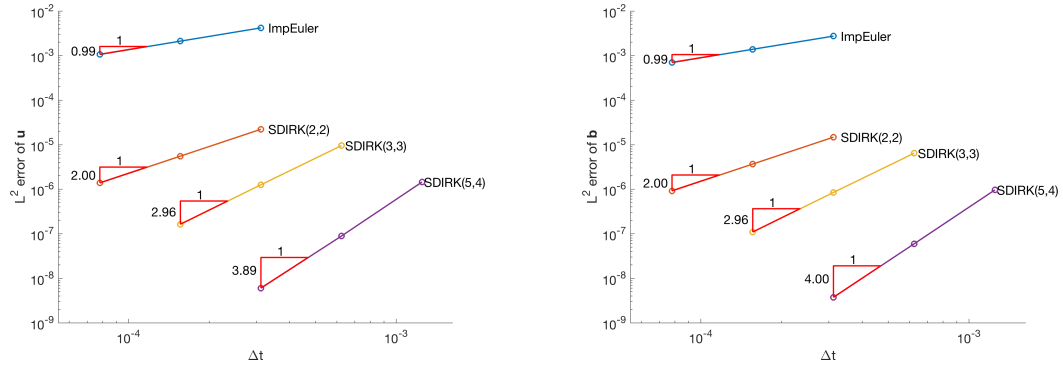


Fig. 6.5: Rayleigh flow problem: temporal convergence of velocity and magnetic field,  $1 \times 32$  non-uniformly spaced elements, 7th-order basis, final time of 0.01.

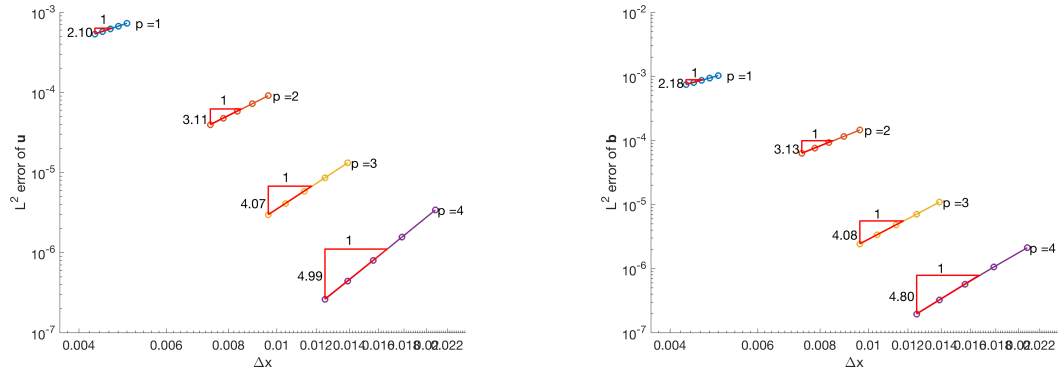


Fig. 6.6: Rayleigh flow problem: spatial convergence of velocity and magnetic field,  $1 \times K$  uniformly-spaced elements, DIRK(5,4), final time of 0.01.

**7. Conclusions.** The multiphysics PDE system of resistive MHD is strongly coupled, highly nonlinear, and characterized by multiple interacting physical phenomena spanning a very large range of length- and time-scales. These characteristics make the scalable, robust, accurate, and efficient computational solution of these systems, over relevant dynamical length- and time-scales of interest, extremely challenging.

Toward development of such a scheme, we have constructed a new high-order implicit HDG method for a linearized, resistive, incompressible MHD system. We have proven stability of the scheme, and have applied the scheme to steady and unsteady benchmark problems with smooth solutions. The linear scheme was employed in a fixed-point nonlinear solver, and has demonstrated optimal convergence in the velocity and magnetic field variables ( $p + 1$ ) for these benchmark problems.

## REFERENCES

- [1] R. ALEXANDER, Diagonally Implicit Runge-Kutta Methods for Stiff ODE's, *SIAM Journal on Numerical Analysis*, 14 (1977), pp. 1006–1021.
- [2] D. S. BALSARA, Divergence-Free Adaptive Mesh Refinement for Magnetohydrodynamics, *J. Comput. Phys.*, 174 (2001), pp. 614–648.
- [3] T. BUI-THANH, From Godunov to a Unified Hybridized Discontinuous Galerkin Framework for Partial Differential Equations, *Journal of Computational Physics*, 295 (2015), pp. 114–146.
- [4] E. A. C. R. SOVINEC AND THE NIMROD TEAM, Nonlinear Magnetohydrodynamics Simulation Using High-Order Finite Elements, *J. Comput. Phys.*, 195 (2004), pp. 355–386.
- [5] B. COCKBURN, J. GOPALAKRISHNAN, AND R. LAZAROV, Unified Hybridization of Discontinuous Galerkin, Mixed, and Continuous Galerkin Methods for Second Order Elliptic Problems, *SIAM J. Numer. Anal.*, 47 (2009), pp. 1319–1365.
- [6] M. DUMBSER, O. ZANOTTI, R. LOUBÈRE, AND S. DIOT, A Posteriori Subcell Limiting of the Discontinuous Galerkin Finite Element Method for Hyperbolic Conservation Laws, *Journal of Computational Physics*, 278 (2014), pp. 47–75.
- [7] V. GIRAULT AND P.-A. RAVIART, *Finite Element Methods for Navier-Stokes Equations: Theory and Algorithms*, vol. 5, Springer Science & Business Media, 2012.
- [8] D. S. HARNED AND W. KERNER, Semi-Implicit Method for Three-Dimensional Compressible Magnetohydrodynamic Simulation, *J. Comput. Phys.*, 60 (1985), pp. 62–75.
- [9] S. C. JARDIN, Review of Implicit Methods for the Magnetohydrodynamic Description of Magnetically Confined Plasmas, *J. Comp. Physics*, 231 (2012), pp. 822–838.
- [10] S. C. JARDIN AND J. A. BRESLAU, Implicit Solution of the Four-Field Extended-Magnetohydrodynamic Equations Using High-Order High-Continuity Finite Elements. , *Phys. Plasmas*, 12 (2005), p. 056101.
- [11] C. A. KENNEDY AND M. H. CARPENTER, Diagonally Implicit Runge-Kutta Methods for Ordinary Differential Equations. A Review, (2016).
- [12] J. J. LEE, S. J. SHANNON, T. BUI-THANH, AND J. N. SHADID, Construction and Analysis of an HDG Method for Incompressible Magnetohydrodynamics, (in progress).
- [13] U. MÜLLER AND L. BÜHLER, *Magnetofluidynamics in Channels and Containers*, Springer Science & Business Media, 2013.
- [14] N. NGUYEN, J. PERAIRE, AND B. COCKBURN, A Hybridizable Discontinuous Galerkin Method for Stokes Flow, *Computer Methods in Applied Mechanics and Engineering*, 199 (2010), pp. 582–597.
- [15] N. C. NGUYEN AND J. PERAIRE, Hybridizable Discontinuous Galerkin Methods for Partial Differential Equations in Continuum Mechanics, *Journal of Computational Physics*, 231 (2012), pp. 5955–5988.
- [16] N. C. NGUYEN, J. PERAIRE, AND B. COCKBURN, An Implicit High-Order Hybridizable Discontinuous Galerkin Method for Linear Convection-Diffusion Equations, *Journal Computational Physics*, 228 (2009), pp. 3232–3254.
- [17] N. C. NGUYEN, J. PERAIRE, AND B. COCKBURN, An Implicit High-Order Hybridizable Discontinuous Galerkin Method for the Incompressible Navier-Stokes Equations, *Journal of Computational Physics*, 230 (2011), pp. 1147–1170.
- [18] N. C. NGUYEN, J. PERAIRE, AND B. COCKBURN, High-Order Implicit Hybridizable Discontinuous Galerkin Method for Acoustics and Elastodynamics, *Journal Computational Physics*, 230 (2011), pp. 3695–3718.
- [19] ———, Hybridizable Discontinuous Galerkin Method for the Time Harmonic Maxwell's Equations, *Journal Computational Physics*, 230 (2011), pp. 7151–7175.
- [20] W. PARK, J. BRESLAU, J. CHEN, G. Y. FU, S. C. JARDIN, S. KLASKY, J. MENARD, A. PLETZER, B. C.

- STRATTON, D. STUTMAN, H. R. STRAUSS, AND L. E. SUGIYAMA, Nonlinear Simulation Studies of Tokamaks and STs. , Nuclear fusion, 43 (2003), pp. 483 – 9.
- [21] W. H. REED AND T. R. HILL, Triangular Mesh Methods for the Neutron Transport Equation, Tech. Rep. LA-UR-73-479, Los Alamos Scientific Laboratory, 1973.
- [22] A. C. ROBINSON AND ET. AL., ALEGRA: An Arbitrary Lagrangian-Eulerian Multimaterial, Multiphysics Code, in AIAA 2008-1235 46th AIAA Aerospace Sciences Meeting and Exhibit, Reno, NV, 2008.
- [23] J. N. SHADID, R. P. PAWLOWSKI, E. C. CYR, R. S. TUMINARO, L. CHACÓN, AND P. D. WEBER, Scalable Implicit Incompressible Resistive MHD with Stabilized FE and Fully-Coupled Newton-Krylov-AMG, Computer Methods in Applied Mechanics and Engineering, 304 (2016), pp. 1–25.
- [24] G. TÓTH, The  $\nabla \cdot \mathbf{B} = 0$  Constraint in Shock-Capturing Magnetohydrodynamics Codes, J. Comput. Phys., 161 (2000), pp. 605–652.
- [25] G. TÓTH, R. KEPPENS, AND M. A. BOTCHEV, Implicit and Semi-Implicit Schemes in the Versatile Advection Code: Numerical Tests, Astron. Astrophys., 332 (1998), pp. 1159–1170.

## AN IMPORTANCE SAMPLING APPROACH TO RISK ESTIMATION

TIMUR A. TAKHTAGANOV\*, DREW P. KOURI†, AND DENIS RIDZAL‡

**Abstract.** We propose and study an approach for estimating finite coherent risk functions of computationally expensive quantities of interest. The proposed approach is based on the conjugate duality properties of risk functions. We utilize the idea of risk identifiers to provide a strategy for a more efficient sampling of the input parameter space than plain Monte Carlo sampling. We discuss and evaluate several methods for reweighting obtained samples that are based on minimum distance estimation. This approach is different from a standard approach in importance sampling literature that is based on likelihood ratios. The results of applying our method to the simple test problems indicate general effectiveness of our method as compared to the standard Monte Carlo method.

**1. Introduction.** This paper develops an efficient method for estimating risk functions of uncertain outcomes. The motivation for this research comes from optimization problems governed by PDEs with random inputs. Such problems generally require a large number of deterministic PDE solves. Our goal is to reduce the cost of solving such problems by reducing the number of samples required with a guided sampling of the random input parameters.

In this work we consider sampling-based discretization of problems with random inputs. The advantages of using sampling-based methods, such as Monte Carlo sampling, are easy implementation, by reusing existing deterministic solvers, and easy parallelization. However, the slow rate of convergence of the Monte Carlo method can render it intractable for expensive problems, such as PDE-constrained optimization problems. The efficiency of the Monte Carlo method can be improved by employing variance reduction methods, such as importance sampling (IS) [7]. The main idea of IS is to generate weighted samples not from the original distribution of random inputs but from an auxiliary probability distribution referred to as the biasing distribution. Successful application of IS relies heavily on the choice of biasing distribution that is used to generate samples. A general approach that does not assume any knowledge of the form of biasing distribution is known in the literature as nonparametric IS or NIS [25, 14]. This approach proceeds by constructing a biasing distribution using kernel density estimation (KDE) [22]. When samples are generated from the biasing distribution they are weighted by the ratio of the original density to the biasing density evaluated at the sample (likelihood ratio). The appearance of the biasing density in the denominator may cause extremely unbalanced weights leading to very inaccurate IS estimates [15]. Several studies suggest that application of NIS becomes especially hard in higher-dimensional cases (5-10 random inputs) [24, 14]. It is also important to note that KDE methods suffer from increased bias with increasing dimension [22].

The approach to risk estimation proposed in this paper combines ideas from the IS literature with the concepts of conjugate duality of convex risk functions from stochastic programming literature [21]. Using biconjugate representations of convex risk functions, our approach identifies the regions of the input parameter space that are “important” for a given risk function. We use KDE or acceptance-rejection (AR) methods to generate samples in the identified important regions. We further propose the use of goodness-of-fit tests from statistical literature [5, 4, 12] to identify weights for the generated samples as an alternative to using likelihood ratios.

The paper is organized as follows. In Section 2 we identify the risk functions of interest and review their properties. In Section 3 we propose an algorithm for estimating risk

---

\*Rice University, tat3@rice.edu

†Sandia National Laboratories, dpkouri@sandia.gov

‡Sandia National Laboratories, dridzal@sandia.gov

functions with adaptive sampling. Sections 4 and 5 present the details of the proposed algorithm. Section 6 provides test examples and numerical results. Section 7 draws conclusions from the numerical results and outlines future work.

**2. Risk functions.** Let  $f : \mathcal{Z} \rightarrow \mathcal{H}$  be a function from a Banach space of decision variables  $\mathcal{Z}$  to a vector space of random variables  $\mathcal{H}$ , i.e.,  $f(z)$  is a random variable with realizations denoted by  $f(z, \xi) = f(z)(\xi)$ . For example,  $f(z, \xi)$  might represent a quantity of interest associated with a solution of a complex PDE model for a given  $z$ . Consider the problem of minimizing a composite risk function of the quantity of interest  $f$ :

$$\min_{z \in \mathcal{Z}} \mathcal{R}[f(z, \xi)]. \quad (2.1)$$

This problem provides the motivation for the current work. In this paper we only address a question of efficient approximation of the objective function in the problem above,  $\mathcal{R}[f(z, \xi)]$ . Therefore, in the following we suppress the dependence on  $z$  and simply write  $f(\xi)$  or  $X(\xi)$  depending on the context.

In a broad sense by a risk function we understand a function  $\mathcal{R}$  that assigns a real value  $\mathcal{R}[X]$  to an uncertain outcome  $X \in \mathcal{H}$ . To make this more precise we consider the Banach space  $\mathcal{H} = L^p_\rho(\Xi)$  of measurable functions  $X : \Xi \subset \mathbb{R}^M \rightarrow \mathbb{R}$  satisfying

$$\int_{\Xi} |X(\xi)|^p \rho(\xi) d\xi < +\infty,$$

where  $\rho : \Xi \rightarrow [0, \infty]$  is a probability density function. Particularly useful in applications are the risk functions that satisfy the following four properties: for  $X, Y \in \mathcal{H}$

1. convexity:  $\mathcal{R}[\alpha X + (1 - \alpha)Y] \leq \alpha \mathcal{R}[X] + (1 - \alpha)\mathcal{R}[Y]$ ,  $\alpha \in [0, 1]$ ;
2. monotonicity:  $\mathcal{R}[Y] \geq \mathcal{R}[X]$  if  $Y \geq X$  almost surely;
3. translation equivariance:  $\mathcal{R}[X + a] = \mathcal{R}[X] + a$  for  $a \in \mathbb{R}$ ;
4. positive homogeneity:  $\mathcal{R}[tX] = t\mathcal{R}[X]$ , for  $t \geq 0$ ,  $t \in \mathbb{R}$ .

The functions  $\mathcal{R} : \mathcal{H} \rightarrow \overline{\mathbb{R}} = \mathbb{R} \cup \{+\infty\}$  satisfying these four properties are called coherent measures of risk [2]. Furthermore, we assume that risk functions are proper, i.e.,

$$\mathcal{R}[X] > -\infty, \forall X, \text{ and } \text{dom}(\mathcal{R}) = \{X \in \mathcal{H} : \mathcal{R}[X] < +\infty\} \neq \emptyset.$$

In this work we consider two proper coherent risk functions that fall into the category of mean-risk models. In these models the objective is a combination of the mean of an uncertain outcome and the risk associated with it. The two risk functions under consideration are:

1. Mean-CVaR:

$$\mathcal{R}[X] = (1 - c)\mathbb{E}[X] + c\text{CVaR}_\beta[X], \quad c \in [0, 1]$$

where [19]

$$\text{CVaR}_\beta[X] = \inf_{t \in \mathbb{R}} \left\{ t + \frac{1}{1 - \beta} \mathbb{E}[(X - t)^+] \right\}$$

with  $(\cdot)^+ = \max\{\cdot, 0\}$ .

2. Mean-semideviation:

$$\mathcal{R}[X] = \mathbb{E}[X] + c\mathbb{E}[(X - \mathbb{E}[X])^+], \quad c \in [0, 1].$$



Both considered models are characterized by quantifying the risk of one-sided deviations of the quantity of interest: for the mean-semideviation model it is the risk of being above the mean, and for the mean-CVaR model - the risk associated with an upper tail of the distribution of the quantity of interest. These formulations are particularly well-suited for the minimization problem (2.1). To make the problems more specific we consider the mean-CVaR model with  $c = 1$  (pure CVaR),  $c = 0.5$  (mean-CVaR), and the mean-semideviation model with  $c = 1$  (mean-SD).

Before introducing our approach to estimation and minimization of the considered risk functions we review the relevant properties of convex risk functions.

**2.1. Conjugate duality of risk functions.** The following review is based on [21, 23].

As in the previous section, let  $\mathcal{R} : \mathcal{H} \rightarrow \overline{\mathbb{R}}$  be a risk function defined on an appropriate space  $\mathcal{H}$  of random outcomes. Specifically, for mean-CVaR and mean-semideviation the domain is  $\mathcal{H} = L_\rho^1(\Xi)$ . The topological dual  $\mathcal{H}^*$  for these two functions is  $L_\rho^\infty(\Xi)$ . From now on we restrict discussion to these spaces.

For  $X \in L_\rho^1(\Xi)$  and  $\vartheta \in L_\rho^\infty(\Xi)$  define a dual pairing as

$$\langle \vartheta, X \rangle = \int_{\Xi} \vartheta(\xi) X(\xi) \rho(\xi) d\xi.$$

Then the conjugate function  $\mathcal{R}^* : L_\rho^\infty(\Xi) \rightarrow \overline{\mathbb{R}}$  is defined as

$$\mathcal{R}^*(\vartheta) = \sup_{X \in L_\rho^1(\Xi)} \{ \langle \vartheta, X \rangle - \mathcal{R}(X) \}$$

and the biconjugate function as

$$\mathcal{R}^{**}(X) = \sup_{\vartheta \in L_\rho^\infty(\Xi)} \{ \langle \vartheta, X \rangle - \mathcal{R}^*(\vartheta) \}.$$

By the Fenchel-Moreau theorem [18, Theorem 5] we have that if  $\mathcal{R} : L_\rho^1(\Xi) \rightarrow \overline{\mathbb{R}}$  is convex, proper, and lower semi-continuous (l.s.c.), then  $\mathcal{R}^{**} = \mathcal{R}$ , i.e.,

$$\mathcal{R}(X) = \sup_{\vartheta \in L_\rho^\infty(\Xi)} \{ \langle \vartheta, X \rangle - \mathcal{R}^*(\vartheta) \} = \sup_{\vartheta \in \text{dom}(\mathcal{R}^*)} \{ \langle \vartheta, X \rangle - \mathcal{R}^*(\vartheta) \}.$$

Furthermore, if  $\mathcal{R}$  is monotonic, translation equivariant, and positively homogeneous, then [23, p. 263]

$$\mathcal{R}(X) = \sup_{\vartheta \in \mathcal{A}} \langle \vartheta, X \rangle, \quad \forall X \in L_\rho^1(\Xi), \quad (2.2)$$

with  $\mathcal{A} \equiv \text{dom}(\mathcal{R}^*)$  given by (see [17, p. 114])

$$\mathcal{A} = \{ \vartheta \in \mathcal{P} \mid \langle \vartheta, X \rangle \leq \mathcal{R}(X), \forall X \in \mathcal{H} \},$$

where  $\mathcal{P}$  is a subset of the set of probability density functions

$$\mathcal{P} = \left\{ \vartheta \in L_\rho^\infty(\Xi) \mid \int_{\Xi} \vartheta(\xi) \rho(\xi) d\xi = 1, \vartheta \geq 0 \right\}.$$

The set  $\mathcal{A}$  is called the *risk envelope* [13].

The maximizers  $\vartheta \in \mathcal{A}$  in (2.2) are called *risk identifiers* [13] and form a set

$$\partial \mathcal{R}(X) = \arg \max_{\vartheta \in \mathcal{A}} \langle \vartheta, X \rangle. \quad (2.3)$$

**2.2. Application to specific risk functions.** We now review the implications of the general theory in the previous section for the risk models that we consider.

**Example 2.1** Consider the mean-CVaR risk function

$$\mathcal{R}[X] = (1 - c) \mathbb{E}[X] + c \text{CVaR}_\beta[X], \quad c \in [0, 1].$$

Its conjugate function is the indicator function of the set  $\mathcal{A} = \text{dom}(\mathcal{R}^*)$ . We can represent  $\mathcal{R}$  in its dual form (2.2) with

$$\mathcal{A} = \{\vartheta \in L_\rho^\infty(\Xi) \mid \vartheta(\xi) \in [\nu, \eta] \text{ a.e. } \xi \in \Xi, \mathbb{E}[\vartheta] = 1\}$$

where  $\nu = 1 - c$  and  $\eta = 1 + \frac{c\beta}{1-\beta}$  (see [21]).

In particular, for  $c = 1$  (pure CVaR) we have  $\mathcal{R}[X]$  represented as

$$\mathcal{R}[X] = \text{CVaR}_\beta[X] = \sup_{\vartheta \in \mathcal{A}} \mathbb{E}[\vartheta X]$$

with

$$\mathcal{A} = \left\{ \vartheta \in L_\rho^\infty(\Xi) \mid \vartheta(\xi) \in \left[0, \frac{1}{1-\beta}\right] \text{ a.e. } \xi \in \Xi, \mathbb{E}[\vartheta] = 1 \right\}.$$

Since  $\text{CVaR}_\beta[X]$  is convex and continuous, it is subdifferentiable and its subdifferential is given by

$$\partial \text{CVaR}_\beta[X] = \left\{ \vartheta \mid \mathbb{E}[\vartheta] = 1, \begin{cases} \vartheta(\xi) = 0 & \text{if } X < \text{VaR}_\beta(X) \\ \vartheta(\xi) = \frac{1}{1-\beta} & \text{if } X > \text{VaR}_\beta(X) \\ \vartheta(\xi) \in [0, \frac{1}{1-\beta}] & \text{if } X = \text{VaR}_\beta(X) \end{cases} \right\},$$

where  $\text{VaR}_\beta(X) = \inf\{t \in \mathbb{R} : \Pr[X \leq t] \geq \beta\}$  is the  $\beta$ -quantile of  $X$ .

For the case  $c \in (0, 1)$

$$\partial \mathcal{R}[X] = \left\{ \vartheta \mid \mathbb{E}[\vartheta] = 1, \begin{cases} \vartheta(\xi) = \nu & \text{if } X < \text{VaR}_\beta(X) \\ \vartheta(\xi) = \eta & \text{if } X > \text{VaR}_\beta(X) \\ \vartheta(\xi) \in [\nu, \eta] & \text{if } X = \text{VaR}_\beta(X) \end{cases} \right\}.$$

◇

**Example 2.2** Consider the mean-semideviation risk function

$$\mathcal{R}[X] = \mathbb{E}[X] + c \mathbb{E}[(X - \mathbb{E}[X])^+].$$

The set  $\mathcal{A}$  in this case is given by [21]

$$\mathcal{A} = \{\vartheta \in L_\rho^\infty(\Xi) \mid \vartheta(\xi) = 1 + \zeta(\xi) - \int_\Xi \zeta(\xi) \rho(\xi) d\xi, \quad \|\zeta\|_\infty \leq c, \quad \zeta \geq 0 \text{ a.e. in } \Xi\}.$$

The set  $\mathcal{A}$  is a set of probability densities if  $c \in [0, 1]$ .

Because  $\mathcal{R}$  is convex, positively homogeneous, and continuous, for any  $X \in L_\rho^1(\Xi)$  its subdifferential is nonempty and given by

$$\partial \mathcal{R}(X) = \left\{ 1 + \zeta(\xi) - \int_\Xi \zeta(\xi) \rho(\xi) d\xi \mid \zeta \in \mathcal{D}_X \right\}$$

with

$$\mathcal{D}_X = \arg \max_{\vartheta \in L_p^\infty(\Xi)} \left\{ \int_{\Xi} (X(\xi) - \mathbb{E}[X]) \vartheta(\xi) \rho(\xi) d\xi \mid \|\vartheta\|_\infty \leq c, \quad \vartheta \geq 0 \text{ a.e. in } \Xi \right\},$$

which has the following form [21]

$$\mathcal{D}_X = \left\{ \vartheta \in cB_\infty : \begin{cases} \vartheta(\xi) = c, & \text{if } X > \mathbb{E}[X] \\ \vartheta(\xi) = 0, & \text{if } X < \mathbb{E}[X] \\ \vartheta(\xi) \in [0, c], & \text{if } X = \mathbb{E}[X] \end{cases} \right\},$$

where  $B_\infty = \{\vartheta \in L_p^\infty(\Xi) \mid \|\vartheta\|_\infty \leq 1\}$ . In conclusion,

$$\partial \mathcal{R}(X) = \left\{ \vartheta \in cB_\infty \mid \begin{cases} \vartheta(\xi) = 1 + c(1 - \Pr(X > \mathbb{E}[X])), & \text{if } X > \mathbb{E}[X] \\ \vartheta(\xi) = 1 - c\Pr(X > \mathbb{E}[X]), & \text{if } X < \mathbb{E}[X] \end{cases} \right\}.$$

For  $X = \mathbb{E}[X]$  we have  $\vartheta(\xi) \in [1 + c(1 - \Pr(X > \mathbb{E}[X])), 1 - c\Pr(X > \mathbb{E}[X])]$ .

◇

**3. Risk-informed sampling.** The two risk functions considered - mean-CVaR and mean-semideviation - are difficult to estimate efficiently. This is due to, on the one hand, the non-smoothness of the positive part function  $(\cdot)^+$  which makes efficient sampling methods such as sparse grids intractable [3]. On the other hand there are sampling deficiencies arising from the fact that only a fraction of samples contribute to the estimates of CVaR and semideviation when using Monte Carlo sampling.

Our approach is based on the dual representation of risk measures presented in the previous section. Specifically, we propose to use the risk identifiers forming the set (2.3) and the biconjugate representation of the risk functions (2.2) to sample more efficiently. That is we would like to sample from the distributions defined by risk identifiers in order to estimate risk functions more accurately. The general framework is defined in the following algorithm 3.1. The inputs provided are the initial input distribution specified by the density function  $\rho(\xi)$ , the objective function (quantity of interest)  $X : \Xi \rightarrow \mathbb{R}$ , and the sizes of samples at each step  $N_0, N_1, \dots, N_S$ .

**Remark.** Algorithm 3.1 resembles the NIS algorithm, specifically its adaptive version NAIS (see [25]). Here, however, we are not attempting to build the biasing distribution based on the integrand (e.g., plus function in CVaR), but rather guide the sampling by utilizing information about the risk function itself (biconjugate representation). Another important difference is the way we address the problem of adjusting probabilities when combining the samples from different densities. The standard approach in importance sampling literature is to use the importance weights coming from the ratios of probability densities:

$$p^{(j)} = \frac{1}{N} l^{(j)}, \text{ for } j = 1, \dots, N, \text{ with } l^{(j)} = \frac{\rho(\xi^{(j)})}{\phi(\xi^{(j)})} \text{ and } N = \sum_{i=0}^k N_i. \quad (3.1)$$

Here  $l^{(j)}$ 's are the so-called likelihood ratios. In general, the sum of the likelihood ratios is not  $N$ , thus, the weights  $p^{(j)}$  might not sum up to 1. This might not matter for some problems, for example, for estimating CVaR, since the estimate only relies on the probabilities in the tail. In general, however, it is desirable to have the weights that sum up to 1. Several approaches to normalize the weights based on likelihood ratios exist in the literature [8]. Here we include the results obtained using likelihood ratio probabilities (3.1) for comparison

**Algorithm 3.1** Risk-informed sampling (RIS) for estimating  $\mathcal{R}[X(\xi)]$ .

- 
- 1: **procedure** RIS( $\rho(\cdot)$ ,  $X$ ,  $\{N_k\}_{k=0}^S$ )  
 2:   Draw  $N_0$  samples from the original distribution  $\rho(\cdot)$ :

$$\Xi_0 = \{\xi^{(j)}, p^{(j)}\}_{j=1}^{N_0}.$$

- Here  $p^{(j)}$ ,  $j = 1, \dots, N_0$ , are the weights assigned to samples (e.g.,  $p^{(j)} = \frac{1}{N_0}$ ).  
 3:   Estimate  $\mathcal{R}[X(\xi)]$  by  $\mathcal{R}[X(\Xi_0)]$  and  $\vartheta(\xi)$  by  $\vartheta(\Xi_0)$ .  
 4:   **for**  $k \leftarrow 1$ ,  $S$  **do**  
 5:     Approximate density  $\phi(\xi) \approx \vartheta(\xi)\rho(\xi)$  from  $\Xi_{k-1}$ .  
 6:     Generate samples from  $\phi(\xi)$ :  $\delta\Xi_k = \{\bar{\xi}^{(j)}, \bar{p}^{(j)}\}_{j=1}^{N_k}$ .  
 7:     Define

$$\Xi_k = \Xi_{k-1} \cup \delta\Xi_k = \{\xi^{(j)}, p^{(j)}\}_{j=1}^{\sum_{i=0}^k N_i}.$$

- 8:     Adjust probabilities  $p^{(j)}$  so that  $\Xi_k$  is a random variable.  
 9:     Estimate  $\mathcal{R}[X(\xi)]$  by  $\mathcal{R}[X(\Xi_k)]$  and  $\vartheta(\xi)$  by  $\vartheta(\Xi_k)$ .
- 

purposes, however, our main approach to adjusting probabilities of newly computed samples is based on solving a distribution matching problem to be discussed in Section 5.

Several points need to be clarified in the algorithm above. Firstly,  $\mathcal{R}[X(\Xi_k)]$  and  $\vartheta(\Xi_k)$  mean the estimates of a risk function and a risk identifier of a discrete variable  $\Xi_k$ . E.g., in the case  $\mathcal{R}[X] = \text{CVaR}_\beta[X]$  the estimates are obtained as follows: first,  $X(\xi^{(j)})$  are sorted in descending order and samples are relabeled so that  $X(\xi^{(1)}) > X(\xi^{(2)}) > \dots > X(\xi^{(N)})$  with  $N = \sum_{i=0}^k N_i$ , then the index  $k_\beta$  is found such that

$$\sum_{j=1}^{k_\beta-1} p^{(j)} \leq 1 - \beta < \sum_{j=1}^{k_\beta} p^{(j)},$$

and  $\text{CVaR}_\beta[\Xi_k]$  is estimated as

$$\text{CVaR}_\beta[\Xi_k] = \frac{1}{1 - \beta} \left( \left( 1 - \beta - \sum_{j=1}^{k_\beta-1} p^{(j)} \right) \text{VaR}_\beta[X] + \sum_{j=1}^{k_\beta-1} p^{(j)} X(\xi^{(j)}) \right),$$

where  $\text{VaR}_\beta[X] = X(\xi^{(k_\beta)})$ . The risk identifier values are assigned as follows:

$$\vartheta(\xi^{(j)}) = \begin{cases} \frac{1}{1 - \beta} & \text{for } j = 1, \dots, k_\beta - 1, \\ 0 & \text{for } j = k_\beta + 1, \dots, N, \\ \frac{1}{p^{(k_\beta)}} \left( 1 - \frac{1}{1 - \beta} \sum_{j=1}^{k_\beta-1} p^{(j)} \right) & \text{for } j = k_\beta, \end{cases}$$

(assuming that there is only one index  $i$  such that  $X(\xi^{(i)}) = \text{VaR}_\beta[X]$ , otherwise, the last formula is adjusted accordingly), so that  $\sum_{j=1}^N \vartheta(\xi^{(j)}) p^{(j)} = 1$ .

Secondly, generation of the new samples on line 6 of algorithm 3.1 requires sampling from the density  $\vartheta(\xi)\rho(\xi)$ . We consider two ways to generate such samples: the first way involves

using kernel density estimation (KDE) to construct an approximation  $\phi(\xi) \approx \vartheta(\xi)\rho(\xi)$  and sampling from  $\phi(\xi)$ ; the second way is to sample from the original density  $\rho(\xi)$  and to use the acceptance-rejection method to keep samples distributed as  $\vartheta(\xi)\rho(\xi)$ . These two methods are considered in more detail in the next section.

**4. Sampling in important regions.** In this section we address the question of generating new samples. We consider two ways to generate new samples based on the information from the previous iteration. Suppose we have a set of points  $\Xi_{k-1}$  and an estimate of the risk identifier  $\vartheta(\xi)$ .

**4.1. Kernel density estimation.** Kernel density estimation (KDE) is a nonparametric density estimation method that is formulated by centering a smooth, symmetric kernel at a given set of sample points [11, Section 4.5]. In short it can be summarized as follows: given samples  $\xi^{(1)}, \dots, \xi^{(N)} \in \mathbb{R}^M$  (from a discrete distribution), build kernel density estimate

$$\phi(\xi) = \frac{1}{\bar{w}} \frac{1}{N \det(B)} \sum_{j=1}^N w(\xi^{(j)}) K_M(B^{-1}(\xi^{(j)} - \xi)),$$

where  $B = \text{diag}(b_1, \dots, b_M)$  is a diagonal matrix of window widths,  
 $K_M(\cdot)$  is a kernel function (e.g., Gaussian kernel),  
 $w(\xi)$  is a weighting function, and  $\bar{w} = \sum_{j=1}^N w(\xi^{(j)})$ .

Originally we chose the weighting function  $w(\xi)$  to be equal to  $\vartheta(\xi)$  (the risk identifier). However, better results were obtained with

$$w(\xi^{(j)}) = \vartheta(\xi^{(j)}) l^{(j)},$$

where  $l^{(j)}$ 's are the likelihood ratios associated with the samples from the previous step (with likelihood ratios corresponding to initial samples being identically 1).

Furthermore, we choose the kernel function to be Gaussian. The window width parameters  $b_1, \dots, b_M$  are chosen using the so-called ‘‘Rule-of-Thumb’’ method [11, Section 4.5.4].

**4.2. Acceptance-rejection method.** The acceptance-rejection algorithm for continuous random variables is a basic method that can be used to generate samples from a desired distribution  $\phi(\xi)$  given another distribution  $\rho(\xi)$ . A necessary assumption is that  $\phi(\xi)/\rho(\xi) \leq C$  with  $C > 0$ . Since in our case  $\phi(\xi) = \vartheta(\xi)\rho(\xi)$  and  $\vartheta(\xi)$  takes values in a bounded interval  $[\vartheta_{LB}, \vartheta_{UB}]$  this assumption is satisfied with  $C = \vartheta_{UB}$ . The procedure to generate  $N$  samples from  $\phi(\xi)$  is then as follows: for  $j = 1, \dots, N$

1. generate a candidate sample  $\xi^c$  from  $\rho(\xi)$ ;
2. generate a uniformly distributed variable  $U$ ;
3. if

$$U \leq \frac{\phi(\xi^c)}{C\rho(\xi^c)} = \frac{\vartheta(\xi^c)}{\vartheta_{UB}}$$

set  $\xi^{(j)} = \xi^c$  (‘‘accept’’) and  $j = j + 1$ ; otherwise go back to 1 (‘‘reject’’).

Of course, we cannot compute  $\vartheta(\xi)$  at a given candidate sample  $\xi^c$  without computing first the value of objective function at  $\xi^c$ . However, we can approximate  $\vartheta(\xi^c)$  by extrapolating the values of  $\vartheta(\xi)$  at already computed samples. For example, we can take

$$\vartheta(\xi^c) = \vartheta(\xi^{(i)}), \quad \text{where } \xi^{(i)} = \arg \min_{\xi^{(j)}} d(\xi^c, \xi^{(j)})$$

and  $d(\cdot, \cdot)$  is some distance metric (e.g., Euclidean distance). E.g., for the case  $\mathcal{R}[X] = \text{CVaR}_\beta[X]$  we will only be accepting samples that are ‘‘close’’ to the samples that lie in the upper tail of the distribution of  $X(\xi)$ .

**5. Adjusting probabilities.** An important part of algorithm 3.1 is adjusting probabilities  $p^{(j)}$  when newly generated samples are added to the old set. We pursued the idea of determining the probabilities of atoms in the combined set

$$\{\xi^{(j)}\}_{j=1}^{\sum_{i=0}^{k-1} N_i} \cup \{\bar{\xi}^{(j)}\}_{j=1}^{N_k}$$

by matching the empirical cumulative distribution function of the discrete random variable  $\Xi_k$  to the known cumulative distribution function of the input random variable  $\xi$ . The general problem is formulated as follows:

$$\min_{\mathbf{p}} D(F_N(x; \mathbf{p}), F_\xi(x)), \quad (5.1)$$

where  $F_N(x; \mathbf{p})$  is an empirical distribution function,

$$F_N(x; \mathbf{p}) = \sum_{j=1}^N p^{(j)} \mathbb{1}[\xi^{(j)} \leq x], \quad N = \sum_{i=0}^k N_i, \quad (5.2)$$

that depends on the vector of probabilities  $\mathbf{p} = [p^{(1)}, \dots, p^{(N)}]$ . Here,  $F_\xi(x)$  is a true c.d.f. of input variable  $\xi$

$$F_\xi(x) = \Pr(\xi \leq x), \quad (5.3)$$

and  $D(\cdot, \cdot)$  is a distance function measuring the difference between two distributions.

The problem (5.1) is similar to the minimum distance estimation problem in statistical literature [6]. Most studies of minimum distance estimation are based on goodness-of-fit tests, with test statistics being used as distance measures. Some examples of goodness-of-fit tests are Kolmogorov-Smirnov and Cramèr-von Mises tests [5]: given two univariate distribution functions  $F_N(\cdot)$  and  $F(\cdot)$

1. Kolmogorov-Smirnov (KS) distance is defined as

$$D_{KS}(F_N, F) = \sup_{x \in \mathbb{R}} |F_N(x) - F(x)| \quad (5.4)$$

2. Cramèr-von Mises (CvM) distance is defined as

$$D^{CvM}(F_N, F) = \int_{x \in \mathbb{R}} (F_N(x) - F(x))^2 dF(x). \quad (5.5)$$

Note that in the univariate case one could also consider minimizing distance between the empirical distribution  $F_N(\cdot; \mathbf{p})$  and the true distribution  $F_\xi(\cdot)$  by simply ordering the samples  $\xi^{(1)}, \dots, \xi^{(N)}$  in the ascending order and setting probabilities  $p^{(j)}$  to be

$$p^{(1)} = F_\xi(\xi^{(1)}), \quad p^{(j)} = F_\xi(\xi^{(j)}) - \sum_{i=1}^{j-1} p^{(i)}, \quad p^{(N)} = 1 - \sum_{i=1}^{N-1} p^{(i)}.$$

This would always provide a “lower” approximation to  $F_\xi(x)$ , that is, one would get  $F_N(x; \mathbf{p}) \leq F_\xi(x)$ . By shifting probabilities above to the left, i.e., setting  $p^{(j)} \leftarrow p^{j+1}$ , one would get an “upper” approximation to  $F_\xi(x)$ , i.e.,  $F_N(x; \mathbf{p}) \geq F_\xi(x)$ . In the multivariate case,  $\Xi \subset \mathbb{R}^M$ , we do not have an ordering of the set of samples; therefore, assigning probabilities as above is not possible and instead we solve the minimization problem (5.1).

In the following we only consider distances based on Cramèr-von Mises test. Based on our initial experiments Kolmogorov-Smirnov distance proved to be too conservative for

our problem: focusing on the largest absolute difference between the empirical and the true distribution functions does not allow to sufficiently reduce the small differences in the regions that might be especially important. We have also considered the relaxation of the KS distance - the so-called CVaR-distance defined in [16]

$$D_\alpha^U(F_N, F) = \text{CVaR}_\alpha[|F_N(\zeta) - F(\zeta)|], \quad (5.6)$$

however, it didn't provide sufficient improvement over KS distance for problems with more than one random variable.

**5.1. Distance between distributions.** The original Kolmogorov-Smirnov and Cramér-von Mises tests (5.4)-(5.5) can be extended to a multivariate case  $M \geq 2$  using a so-called Rosenblatt transformation [12, 20].

**THEOREM 5.1** (Rosenblatt transformation). *Let  $\boldsymbol{\xi} = (\xi_1, \dots, \xi_M)$  be a random vector with joint density*

$$\rho(\boldsymbol{\xi}) = \rho_1(\xi_1)\rho_2(\xi_2|\xi_1) \dots \rho(\xi_M|\xi_1, \dots, \xi_{M-1}),$$

*and define the transformation  $\boldsymbol{\zeta} = T_R(\boldsymbol{\xi})$  by*

$$\zeta_1 = F_{\xi_1}(\xi_1), \quad \zeta_i = F_{\xi_i}(\xi_i|\xi_1, \dots, \xi_{i-1}), \quad i = 2, \dots, M.$$

*Then  $\zeta_1, \dots, \zeta_M$  are i.i.d. uniform 0-1.*

In the theorem above  $F_{\xi_i}(\cdot)$  are marginal distribution functions. In general, the transformation  $T_R$  is not unique since there are  $M!$  ways to number the coordinates. We consider the case when components of the vector  $\boldsymbol{\xi} = (\xi_1, \dots, \xi_M)$  are independent and  $F_{\boldsymbol{\xi}}(\mathbf{x}) = \prod_{i=1}^M F_{\xi_i}(x_i)$  (similarly  $\rho(\boldsymbol{\xi}) = \prod_{i=1}^M \rho_i(\xi_i)$ ), in which case Rosenblatt transformation is uniquely determined.

The multivariate KS and CvM tests rewritten using Rosenblatt transformation are special cases of the so-called  $L_p$ -star discrepancy [9]:

$$D_p^* = \left( \int_{[0,1]^M} |U_N(\mathbf{x}; \mathbf{p}) - \lambda(\mathbf{x})|^p d\mathbf{x} \right)^{1/p}, \quad (5.7)$$

where

$$U_N(\mathbf{x}; \mathbf{p}) = \sum_{j=1}^n p^{(j)} \mathbb{1}[\boldsymbol{\zeta}^{(j)} \leq \mathbf{x}], \quad \boldsymbol{\zeta}^{(j)} = T_R(\boldsymbol{\xi}^{(j)}),$$

and  $\lambda(\mathbf{x}) = x_1 \dots x_M$  is the volume of hyper-rectangle  $[\mathbf{0}, \mathbf{x}]$ . In particular, CvM distance is a squared  $L_2$ -star discrepancy which we denote by  $D_2$ :

$$D_2 = \int_{[0,1]^M} |U_N(\mathbf{x}; \mathbf{p}) - \lambda(\mathbf{x})|^2 d\mathbf{x}. \quad (5.8)$$

This discrepancy measures the uniformity of the points in the  $M$ -dimensional cube. There are several generalizations of star discrepancy which we also consider (taken from [9, 10, 4]). These require some additional notation: let  $S = \{1, \dots, M\}$  and for  $s \subset S$  let  $\mathbf{x}^s$  denote a  $|s|$ -dimensional vector indexed by elements of  $s$  and  $\tilde{\mathbf{x}}^s = (\tilde{x}_1, \dots, \tilde{x}_M)$  be an  $M$ -dimensional vector with  $\tilde{x}_k = x_k$  if  $k \in s$  and  $\tilde{x}_k = 1$  otherwise.

- Modified  $L_2$ -star discrepancy (squared) is given by:

$$MD_2 = \sum_{\emptyset \subseteq s \subseteq S} \int_{[0,1]^s} |U_N(\tilde{\mathbf{x}}^s; \mathbf{p}) - \lambda(\tilde{\mathbf{x}}^s)|^2 d\mathbf{x}^s. \quad (5.9)$$

This discrepancy measures uniformity of all projections of points onto lower-dimensional hypercubes.

- Centered  $L_2$ -star discrepancy (squared) is given by:

$$CD_2 = \sum_{\emptyset \subseteq s \subseteq S} \int_{[0,1]^s} |U_N^c(\tilde{\mathbf{x}}^s; \mathbf{p}) - \lambda^c(\tilde{\mathbf{x}}^s)|^2 d\mathbf{x}^s. \quad (5.10)$$

Here  $U_N^c(\mathbf{x}; \mathbf{p})$  is the empirical distribution function of random vector  $\boldsymbol{\zeta}$  taking values in the hyper-rectangle formed by  $\mathbf{x}$  and its nearest vertex in  $[0, 1]^s$  (i.e., not necessarily  $(0, \dots, 0)$ ) and  $\lambda^c(\mathbf{x})$  is the volume of that hyper-rectangle.

- Symmetric  $L_2$ -star discrepancy (squared) is given by:

$$SD_2 = \sum_{\emptyset \subseteq s \subseteq S} \int_{[0,1]^s} |U_N^e(\tilde{\mathbf{x}}^s; \mathbf{p}) - \lambda^e(\tilde{\mathbf{x}}^s)|^2 d\mathbf{x}^s. \quad (5.11)$$

Here  $U_N^e(\mathbf{x}; \mathbf{p})$  is the empirical distribution function of random vector  $\boldsymbol{\zeta}$  taking values in the hyper-rectangles formed by  $\mathbf{x}$  and all even vertices in  $[0, 1]^s$  (i.e., vertices with sum of coordinates even) and  $\lambda^e(\mathbf{x})$  is the volume of those hyper-rectangles.

The advantage of using discrepancy-based distances described above is the existence of closed-form expressions that allow fast computation (see Table 5.1).

Distance	Formula
$D_2$	$\sum_{j,k=1}^N p^{(j)} p^{(k)} \prod_{d=1}^M \left( 1 - \max\{\zeta_d^{(j)}, \zeta_d^{(k)}\} \right)$ $- 2 \sum_{j=1}^N p^{(j)} \prod_{d=1}^M \left( \frac{1 - (\zeta_d^{(j)})^2}{2} \right) + \left( \frac{1}{3} \right)^M$
$MD_2$	$\sum_{j,k=1}^N p^{(j)} p^{(k)} \prod_{d=1}^M \left( 2 - \max\{\zeta_d^{(j)}, \zeta_d^{(k)}\} \right)$ $- 2 \sum_{j=1}^N p^{(j)} \prod_{d=1}^M \left( \frac{3 - (\zeta_d^{(j)})^2}{2} \right) + \left( \frac{4}{3} \right)^M$
$CD_2$	$\sum_{j,k=1}^N p^{(j)} p^{(k)} \prod_{d=1}^M \left( 1 + \frac{1}{2} \left  \zeta_d^{(j)} - \frac{1}{2} \right  + \frac{1}{2} \left  \zeta_d^{(k)} - \frac{1}{2} \right  - \frac{1}{2} \left  \zeta_d^{(j)} - \zeta_d^{(k)} \right  \right)$ $- 2 \sum_{j=1}^N p^{(j)} \prod_{d=1}^M \left( 1 + \frac{1}{2} \left  \zeta_d^{(j)} - \frac{1}{2} \right  - \frac{1}{2} \left  \zeta_d^{(j)} - \frac{1}{2} \right ^2 \right) + \left( \frac{13}{12} \right)^M$
$SD_2$	$\sum_{j,k=1}^N p^{(j)} p^{(k)} \prod_{d=1}^M 2 \left( 1 - \left  \zeta_d^{(j)} - \zeta_d^{(k)} \right  \right)$ $- 2 \sum_{j=1}^N p^{(j)} \prod_{d=1}^M \left( 1 + 2\zeta_d^{(j)} - 2(\zeta_d^{(j)})^2 \right) + \left( \frac{4}{3} \right)^M$

Table 5.1: Formulas for estimating discrepancy-based distances.

The other distance we consider here is a modification of multivariate CvM distance

$$D_{L2} = \int_{\mathbf{x} \in \mathbb{R}^M} (F_N(\mathbf{x}; \mathbf{p}) - F_{\xi}(\mathbf{x}))^2 d\mathbf{x}, \quad (5.12)$$



which we denote by  $D_{L2}$  as it can be seen as a squared  $L_2$ -norm of difference between distributions with respect to Lebesgue measure. This distance has been considered in [1] for the problems of estimating expectations. Note, that when  $F_{\xi}(\mathbf{x})$  is a distribution function of a multivariate uniform 0-1 variable  $D_{L2}$  coincides with the multivariate CvM distance and with the squared  $L_2$ -star discrepancy  $D_2$ . However, for unbounded distributions  $D_{L2}$  is not well-defined. Therefore, we modify the definition of  $D_{L2}$  as follows

$$\tilde{D}_{L2} = \int_{\mathbf{x} \in \tilde{\Xi}} (F_N(\mathbf{x}; \mathbf{p}) - F_{\xi}(\mathbf{x}))^2 d\mathbf{x}, \quad (5.13)$$

where

$$\tilde{\Xi} = \begin{cases} \Xi, & \text{if } \Xi \text{ is bounded,} \\ \text{appropriately defined truncation of } \Xi, & \text{else.} \end{cases}$$

With this definition the practical formula for  $\tilde{D}_{L2}$  is as follows

$$\begin{aligned} \tilde{D}_{L2} &= \sum_{j=1}^N \sum_{k=1}^N p^{(j)} p^{(k)} \prod_{d=1}^M \left( b_d - \max\{\xi_d^{(j)}, \xi_d^{(k)}\} \right) \\ &\quad - 2 \sum_{j=1}^N \prod_{d=1}^M \int_{\xi_d^{(j)}}^{b_d} F_{\xi_d}(x) dx + \prod_{d=1}^M \int_{a_d}^{b_d} F_{\xi_d}^2(x) dx, \end{aligned} \quad (5.14)$$

where  $\tilde{\Xi} = \bigotimes_{d=1}^M [a_d, b_d]$  and  $F_{\xi}(\mathbf{x}) = \prod_{d=1}^M F_{\xi_d}(x_d)$ .

Analogously to the definition of  $MD_2$  we can define  $M\tilde{D}_{L2}$

$$\begin{aligned} M\tilde{D}_{L2} &= \sum_{\emptyset \subseteq s \subseteq S} \int_{\mathbf{x}^s \in \tilde{\Xi}^s} |F_N(\tilde{\mathbf{x}}^s; \mathbf{p}) - F_{\xi}(\tilde{\mathbf{x}}^s)|^2 d\mathbf{x}^s \\ &= \sum_{j=1}^N \sum_{k=1}^N p^{(j)} p^{(k)} \prod_{d=1}^M \left( 1 + b_d - \max\{\xi_d^{(j)}, \xi_d^{(k)}\} \right) \\ &\quad - 2 \sum_{j=1}^N \prod_{d=1}^M \left( 1 + \int_{\xi_d^{(j)}}^{b_d} F_{\xi_d}(x) dx \right) + \prod_{d=1}^M \left( 1 + \int_{a_d}^{b_d} F_{\xi_d}^2(x) dx \right), \end{aligned} \quad (5.15)$$

where  $\tilde{\Xi}^s = \bigotimes_{d \in s} [a_d, b_d]$  and  $\tilde{\mathbf{x}}^s$  is a vector with components  $\tilde{x}_k = x_k$  if  $k \in s$  and  $\tilde{x}_k = b_k$  otherwise.

Finally, we propose using weighted versions of  $\tilde{D}_{L2}$  and  $M\tilde{D}_{L2}$

$$\tilde{D}_{WL2} = \int_{\mathbf{x} \in \tilde{\Xi}} \left( F_N(\mathbf{x}; \mathbf{p}) - F_{\xi}(\mathbf{x}) \right)^2 \psi(\mathbf{x}) d\mathbf{x}, \quad (5.16)$$

$$M\tilde{D}_{WL2} = \sum_{\emptyset \subseteq s \subseteq S} \int_{\mathbf{x}^s \in \tilde{\Xi}^s} \left( F_N(\tilde{\mathbf{x}}^s; \mathbf{p}) - F_{\xi}(\tilde{\mathbf{x}}^s) \right)^2 \tilde{\psi}(\tilde{\mathbf{x}}^s) d\mathbf{x}^s, \quad (5.17)$$

where the weight functions  $\psi(\mathbf{x})$  and  $\tilde{\psi}(\mathbf{x})$  are chosen to prioritize certain regions of parameter space. Specifically, let  $\gamma = \frac{\vartheta_{LB}}{\vartheta_{LB} + \vartheta_{UB}}$  and set

$$\psi(\mathbf{x}) = \begin{cases} 1 - \gamma, & \text{if } \vartheta(\xi^{(i)}) = \vartheta_{UB} \text{ for } \xi^{(i)} = \arg \min_{\xi^{(j)}} \|\mathbf{x} - \xi^{(j)}\| \\ \gamma, & \text{otherwise.} \end{cases} \quad (5.18)$$

Thus we give larger weight  $\left(\frac{\vartheta_{UB}}{\vartheta_{LB} + \vartheta_{UB}}\right)$  to the points in  $\tilde{\Xi}$  that lie closer to samples  $\xi^{(j)}$  for which the risk identifier  $\vartheta(\xi)$  is at the upper bound (“important” samples) and smaller weight  $\left(\frac{\vartheta_{LB}}{\vartheta_{LB} + \vartheta_{UB}}\right)$  to all other points. For the case  $\mathcal{R}[X] = \text{CVaR}_\beta[X]$  we would have  $\gamma = 0$  but we reassign it to be  $\gamma = 10^{-8}$ .

For the  $M\tilde{D}_{WL2}$  distance (5.16) we define  $\tilde{\psi}(\tilde{\mathbf{x}}^s)$  to simply be  $\psi(\mathbf{x})$  (i.e., when  $\mathbf{x}$  is projected onto the sides of hypercube  $\bigotimes_{d=1}^M [a_d, b_d]$  it is given the same weight as when not projected).

If the integrals in (5.16)-(5.17) are discretized using a set of Monte Carlo grid points  $\{\mathbf{x}^{(i)}\}_{i=1}^Q$  then computation of  $\tilde{D}_{WL2}$  and  $M\tilde{D}_{WL2}$  is performed as follows

$$\begin{aligned} \tilde{D}_{WL2} = & \sum_{j=1}^N \sum_{k=1}^N p^{(j)} p^{(k)} \frac{V}{Q} \sum_{i=1}^Q \left( \psi(\mathbf{x}^{(i)}) \prod_{d=1}^M \mathbb{1}[x_d^{(i)} \leq \max\{\xi_d^{(j)}, \xi_d^{(k)}\}] \right) \\ & - 2 \sum_{j=1}^N \frac{V}{Q} \sum_{i=1}^Q \left( \psi(\mathbf{x}^{(i)}) \prod_{d=1}^M \mathbb{1}[x_d^{(i)} \leq \xi_d^{(j)}] F_{\xi_d}(x_d^{(i)}) \right) \\ & + \frac{V}{Q} \sum_{i=1}^Q \left( \psi(\mathbf{x}^{(i)}) \prod_{d=1}^M F_{\xi_d}^2(x_d^{(i)}) \right) \end{aligned} \quad (5.19)$$

and

$$\begin{aligned} M\tilde{D}_{WL2} = & \sum_{j=1}^N \sum_{k=1}^N p^{(j)} p^{(k)} \prod_{d=1}^M \left( 1 + \frac{V_d}{Q} \sum_{i=1}^Q \left( \psi(\mathbf{x}^{(i)}) \mathbb{1}[x_d^{(i)} \leq \max\{\xi_d^{(j)}, \xi_d^{(k)}\}] \right) \right) \\ & - 2 \sum_{j=1}^N \prod_{d=1}^M \left( 1 + \frac{V_d}{Q} \sum_{i=1}^Q \left( \psi(\mathbf{x}^{(i)}) \mathbb{1}[x_d^{(i)} \leq \xi_d^{(j)}] F_{\xi_d}(x_d^{(i)}) \right) \right) \\ & + \prod_{d=1}^M \left( \frac{V_d}{Q} \sum_{i=1}^Q \left( \psi(\mathbf{x}^{(i)}) F_{\xi_d}^2(x_d^{(i)}) \right) \right). \end{aligned} \quad (5.20)$$

In the formulas above  $V_d = b_d - a_d$  and  $V = \prod_{d=1}^M V_d$ . Again (for simplicity) it is assumed that  $F_\xi(\mathbf{x}) = \prod_{i=1}^M F_{\xi_i}(x_i)$ .

**6. Numerical experiments.** We study the performance of the proposed algorithm on several example problems.

**Example 6.1** Sum of normal variables.

Let

$$f(\xi) = \sum_{d=1}^M \xi_d, \quad \xi_d \sim \mathcal{N}(\mu_d, \sigma_d^2),$$

$$\mu_d = (d-1) \frac{2}{M-1}, \quad \sigma_d^2 = 1 + (d-1) \frac{2}{M-1}.$$

Then  $f(\xi) \sim \mathcal{N}(\mu, \sigma^2)$  where  $\mu = \sum_{d=1}^M \mu_d = M$  and  $\sigma^2 = \sum_{d=1}^M \sigma_d^2 = 2M$ . In this case we have exact formulas for  $\text{CVaR}_\beta$  and semideviation of the objective function:

$$\text{CVaR}_\beta[f(\xi)] = \mu + \frac{\sigma \varphi(\Phi^{-1}(\beta))}{1 - \beta},$$

where  $\varphi(x)$  is the standard normal p.d.f. and  $\Phi(\cdot)$  is the c.d.f. of the standard normal distribution, and semideviation is given by

$$SD[f(\xi)] = \frac{\sigma}{\sqrt{2\pi}}.$$

We set  $\tilde{\Xi} = \prod_{d=1}^M [\mu_d - 10\sigma_d, \mu_d + 10\sigma_d]$ . ◇

**Example 6.2** Sum of (0,1)-uniform variables.

Let

$$f(\xi) = \sum_{d=1}^M \xi_d, \quad \xi_d \sim \mathcal{U}(0, 1).$$

In this case  $f(\xi)$  is distributed according to Irwin-Hall distribution with mean  $\mu = \frac{M}{2}$  and c.d.f. given by

$$F_{f(\xi)}(x) = \frac{1}{M!} \sum_{k=0}^{\lfloor x \rfloor} (-1)^k \binom{M}{k} (x - k)^M.$$

The  $\text{CVaR}_\beta(f(\xi))$  and  $SD[f(\xi)]$  are estimated with Monte Carlo sample of size  $10^8$ . ◇

**Example 6.3** Sum of squares of normal variables.

Let

$$f(\xi) = \sum_{d=1}^M \xi_d^2, \quad \xi_d \sim \mathcal{N}(\mu_d, \sigma_d^2),$$

$$\mu_d = 0, \quad \sigma_d^2 = 1.$$

Then  $f(\xi) \sim \chi^2(M)$  with mean  $\mu = M$ . We estimate  $\text{CVaR}_\beta[f(\xi)]$  by integrating the inverse c.d.f. of  $\chi_M^2$ -distribution, and estimate  $SD[f(\xi)]$  using Monte Carlo sample of size  $10^8$ . We set  $\tilde{\Xi} = \prod_{d=1}^M [\mu_d - 10\sigma_d, \mu_d + 10\sigma_d]$ . ◇

**Example 6.4** Exponential of sum of normal variables.

Let

$$f(\xi) = \exp\left(\sum_{d=1}^M \xi_d\right), \quad \xi_d \sim \mathcal{N}(\mu_d, \sigma_d^2),$$

$$\mu_d = -1 + \frac{d-1}{M-1}, \quad \sigma_d^2 = 1.$$

In this case  $\sum_{d=1}^M \mu_d = \frac{M-2}{2}$  and  $\sum_{d=1}^M \sigma_d^2 = M$ , and  $f(\xi)$  is distributed according to log-normal distribution  $f(\xi) \sim \ln \mathcal{N}(\frac{M-2}{2}, M)$  with mean

$$\mu = \exp\left(\sum_{d=1}^M \mu_d + \frac{1}{2} \sum_{d=1}^M \sigma_d^2\right) = \exp(M-1).$$

We estimate  $\text{CVaR}_\beta[f(\xi)]$  by integrating the inverse c.d.f. of the log-normal distribution, and estimate  $SD[f(\xi)]$  using Monte Carlo sample of size  $10^8$ . We set  $\tilde{\Xi} = \prod_{d=1}^M [\mu_d - 10\sigma_d, \mu_d + 10\sigma_d]$ .

◇

**Example 6.5** Ackley's function.

This example is taken from [1]. Let

$$f(\xi) = -20 \exp \left( -0.2 \sqrt{\frac{1}{M} \sum_{d=1}^M \xi_d^2} \right) - \exp \left( \frac{1}{M} \sum_{d=1}^M \cos(2\pi \xi_d) \right) + 20 + \exp(1),$$

$$\xi_d \sim \mathcal{N}(\mu_d, \sigma_d^2), \quad \mu_d = 0, \quad \sigma_d^2 = 1.$$

The mean,  $\text{CVaR}_\beta(f(\xi))$  and  $SD[f(\xi)]$  are estimated with Monte Carlo sample of size  $10^8$ . We set  $\tilde{\Xi} = \prod_{d=1}^M [\mu_d - 10\sigma_d, \mu_d + 10\sigma_d]$ .

◇

Ex.	$M = 2$		$M = 3$		$M = 5$		$M = 10$	
6.1	2.38	[-]	2.86	[-]	2.23	[-]	1.83	[-]
	0.35	$[D_2]$	1.03	$[SD_2]$	1.70	$[CD_2]$	2.30	$[LR]$
	0.92	$[\tilde{D}_{WL2}]$	2.97	$[SD_2]$	1.31	$[SD_2]$	5.60	$[CD_2]$
6.2	1.07	[-]	1.82	[-]	1.00	[-]	0.93	[-]
	0.15	$[CD_2]$	0.67	$[SD_2]$	0.79	$[CD_2]$	1.36	$[M\tilde{D}_{WL2}]$
	0.38	$[D_2]$	1.47	$[SD_2]$	0.68	$[SD_2]$	2.90	$[SD_2]$
6.3	4.63	[-]	4.79	[-]	3.62	[-]	2.76	[-]
	1.34	$[\tilde{D}_{L2}]$	1.40	$[M\tilde{D}_{WL2}]$	5.17	$[\tilde{D}_{L2}]$	8.13	$[LR]$
	1.81	$[\tilde{D}_{WL2}]$	2.99	$[MD_2]$	3.62	$[CD_2]$	2.72	$[MD_2]$
6.4	10.8	[-]	15.5	[-]	28.8	[-]	65.4	[-]
	3.81	$[M\tilde{D}_{L2}]$	10.0	$[LR]$	28.3	$[CD_2]$	33.3	$[LR]$
	4.29	$[\tilde{D}_{WL2}]$	18.8	$[SD_2]$	48.7	$[CD_2]$	55.6	$[CD_2]$
6.5	1.75	[-]	1.26	[-]	0.89	[-]	0.78	[-]
	0.45	$[\tilde{D}_{WL2}]$	0.53	$[M\tilde{D}_{L2}]$	1.41	$[\tilde{D}_{WL2}]$	2.55	$[LR]$
	0.49	$[M\tilde{D}_{WL2}]$	1.31	$[D_2]$	0.91	$[CD_2]$	0.88	$[MD_2]$

Table 6.1: Best relative errors (%) in estimation of  $\text{CVaR}_\beta$  with  $\beta = 0.9$  for examples 6.1-6.5. In brackets the distance that provided best estimate. For each example the first row corresponds to plain Monte Carlo sampling, the second - RIS with KDE, the third - RIS with AR.

**6.1. Results.** We test algorithm 3.1 on the problems of estimating risk functions introduced in Section 2 with the objective functions described in Examples 6.1-6.5. We report

	$M = 2$	$M = 3$	$M = 5$	$M = 10$
Example 6.1	65.7/70.0	63.8/68.8	61.1/56.8	58.0/45.6
Example 6.2	62.9/77.0	64.5/73.6	61.0/62.6	58.3/35.0
Example 6.3	60.3/67.6	63.6/60.0	67.0/44.8	72.3/35.6
Example 6.4	65.5/69.8	62.7/71.6	61.3/59.6	57.6/47.6
Example 6.5	59.2/63.2	57.3/58.8	62.4/43.0	75.4/33.4

Table 6.2: Percentage of samples above  $\text{VaR}_\beta$  when estimating  $\text{CVaR}_\beta$  (out of 500) (KDE/AR). For KDE and AR percentages reported are average over 10 runs for the best distance estimate from Table 6.1. For MC corresponding value is 10.2%.

Ex.	$M = 2$		$M = 3$		$M = 5$		$M = 10$	
6.1	2.86	[-]	1.99	[-]	2.37	[-]	1.43	[-]
	0.14	[ $SD_2$ ]	0.54	[ $CD_2$ ]	0.77	[ $\tilde{D}_{L2}$ ]	0.89	[ $MD_2$ ]
	0.21	[ $M\tilde{D}_{WL2}$ ]	0.51	[ $SD_2$ ]	0.87	[ $CD_2$ ]	0.61	[ $M\tilde{D}_{L2}$ ]
6.2	1.29	[-]	1.45	[-]	1.05	[-]	0.84	[-]
	0.05	[ $M\tilde{D}_{L2}$ ]	0.25	[ $CD_2$ ]	0.51	[ $CD_2$ ]	0.80	[ $MD_2$ ]
	0.06	[ $M\tilde{D}_{WL2}$ ]	0.23	[ $SD_2$ ]	0.24	[ $SD_2$ ]	0.42	[ $CD_2$ ]
6.3	3.76	[-]	3.26	[-]	3.09	[-]	2.49	[-]
	0.59	[ $M\tilde{D}_{WL2}$ ]	1.04	[ $M\tilde{D}_{L2}$ ]	3.54	[ $M\tilde{D}_{WL2}$ ]	4.70	[ $LR$ ]
	0.82	[ $M\tilde{D}_{WL2}$ ]	0.95	[ $\tilde{D}_{WL2}$ ]	1.25	[ $\tilde{D}_{L2}$ ]	1.23	[ $\tilde{D}_{L2}$ ]
6.4	11.4	[-]	12.1	[-]	27.6	[-]	54.1	[-]
	2.06	[ $\tilde{D}_{WL2}$ ]	10.2	[ $\tilde{D}_{WL2}$ ]	21.0	[ $MD_2$ ]	22.5	[ $LR$ ]
	1.68	[ $\tilde{D}_{WL2}$ ]	10.1	[ $SD_2$ ]	20.2	[ $SD_2$ ]	43.3	[ $M\tilde{D}_{L2}$ ]
6.5	1.31	[-]	1.07	[-]	0.79	[-]	0.74	[-]
	0.16	[ $\tilde{D}_{L2}$ ]	0.27	[ $\tilde{D}_{L2}$ ]	0.56	[ $\tilde{D}_{L2}$ ]	1.78	[ $\tilde{D}_{L2}$ ]
	0.23	[ $M\tilde{D}_{WL2}$ ]	0.31	[ $M\tilde{D}_{L2}$ ]	0.30	[ $\tilde{D}_{L2}$ ]	0.39	[ $M\tilde{D}_{L2}$ ]

Table 6.3: Best relative errors (%) in estimation of mean plus  $\text{CVaR}_\beta$  with  $\beta = 0.9$  for examples 6.1-6.5. In brackets the distance that provided best estimate. For each example the first row corresponds to plain Monte Carlo sampling, the second - RIS with KDE, the third - RIS with AR.

results for using both KDE and AR for sample generation in Tables 6.1-6.4. For KDE sampling we use Gaussian kernel (with unbounded support), and apply the same truncation to the support of  $\phi(\xi)$  as we apply to  $\rho(\xi)$  (see definitions of  $\tilde{\Xi}$  above). The sample sizes are taken to be  $N_0 = N_1 = \dots = N_S = 100$  with  $S = 4$ . The initial  $N_0$  samples are generated using Latin Hypercube Sampling (LHS). The results reported are for the last sample of size

Ex.	$M = 2$		$M = 3$		$M = 5$		$M = 10$	
6.1	2.14	[-]	2.10	[-]	1.67	[-]	1.49	[-]
	0.11	$[M\tilde{D}_{WL2}]$	0.22	$[MD_2]$	0.27	$[CD_2]$	0.62	$[\tilde{D}_{L2}]$
	0.08	$[\tilde{D}_{WL2}]$	0.15	$[SD_2]$	0.12	$[\tilde{D}_{L2}]$	0.22	$[M\tilde{D}_{WL2}]$
6.2	1.32	[-]	0.74	[-]	0.70	[-]	0.55	[-]
	0.03	$[M\tilde{D}_{L2}]$	0.11	$[CD_2]$	0.19	$[M\tilde{D}_{WL2}]$	0.26	$[M\tilde{D}_{WL2}]$
	0.02	$[SD_2]$	0.07	$[CD_2]$	0.12	$[CD_2]$	0.12	$[CD_2]$
6.3	3.65	[-]	3.78	[-]	3.04	[-]	1.60	[-]
	0.34	$[\tilde{D}_{L2}]$	0.30	$[\tilde{D}_{L2}]$	0.48	$[\tilde{D}_{L2}]$	3.00	$[M\tilde{D}_{WL2}]$
	0.54	$[\tilde{D}_{L2}]$	0.62	$[CD_2]$	0.67	$[MD_2]$	0.73	$[\tilde{D}_{L2}]$
6.4	6.97	[-]	15.4	[-]	23.9	[-]	57.3	[-]
	2.56	$[\tilde{D}_{L2}]$	4.89	$[SD_2]$	12.7	$[LR]$	39.7	$[\tilde{D}_{L2}]$
	1.94	$[SD_2]$	4.81	$[\tilde{D}_{WL2}]$	12.8	$[M\tilde{D}_{WL2}]$	44.3	$[MD_2]$
6.5	1.47	[-]	1.15	[-]	0.72	[-]	0.70	[-]
	0.09	$[\tilde{D}_{WL2}]$	0.15	$[M\tilde{D}_{WL2}]$	0.15	$[M\tilde{D}_{WL2}]$	0.64	$[M\tilde{D}_{WL2}]$
	0.08	$[\tilde{D}_{WL2}]$	0.15	$[CD_2]$	0.19	$[M\tilde{D}_{WL2}]$	0.21	$[M\tilde{D}_{WL2}]$

Table 6.4: Best relative errors (%) in estimation of mean plus semideviation for examples 6.1-6.5. In brackets the distance that provided best estimate. For each example the first row corresponds to plain Monte Carlo sampling, the second - RIS with KDE, the third - RIS with AR.

500. The error reported is the relative error expressed as a percentage, i.e.,

$$e = \frac{|\mathcal{R}[f(\xi)] - \mathcal{R}[f(\Xi_S)]|}{|\mathcal{R}[f(\xi)]|} \times 100\%,$$

where  $\mathcal{R}[f(\xi)]$  represents the true value of risk function and  $\mathcal{R}[f(\Xi_S)]$  the estimate after  $S$  steps of algorithm 3.1. The errors are averaged over 10 independent runs; i.e., the errors reported are the average errors rather than the errors of the average estimates. The tables present only the smallest errors among all the distances used for adjusting probabilities, which are  $D_2$ ,  $MD_2$ ,  $SD_2$ ,  $CD_2$ ,  $\tilde{D}_{L2}$ ,  $M\tilde{D}_{L2}$ ,  $\tilde{D}_{WL2}$ , and  $M\tilde{D}_{WL2}$ . In addition to using the distance minimization method of Section 5 we include results obtained using likelihood ratios (3.1) (indicated as  $LR$ ). The tables also present errors when using standard Monte Carlo sampling for the last four steps, i.e., initial LHS samples are combined with MC samples for a full sample of size 500. For the weighted distances (5.19)-(5.20) the grid points  $\{\mathbf{x}^{(i)}\}_{i=1}^Q$  are generated using Sobol' sequence generator with  $Q = 10^5$ .

The samples generated for Example 6.2 for the case  $M = 2$  are plotted in Figures 6.1(a)-6.1(f). The line in these figures corresponds to  $f(\xi) = \text{VaR}_\beta[f(\xi)]$  (true value) for the cases of pure CVaR and mean-CVaR risk functions and  $f(\xi) = \mathbb{E}[f(\xi)]$  (true value) for the case of mean-semideviation. Figures 6.1(a) show 6.1(c), 6.1(e) the samples generated using KDE sampling with black squares corresponding to the samples from the second to last step lying below the estimate of  $\text{VaR}_\beta[f(\xi)]$  (or  $\mathbb{E}[f(\xi)]$ ) at that step, red stars - samples

from the second to last step lying above the estimate of  $\text{VaR}_\beta[f(\xi)]$  (or  $\mathbb{E}[f(\xi)]$ ) at that step, and blue circles - samples generated at the last step. The distance used is  $D_2$ . The same convention is used in Figures 6.1(b), 6.1(d), 6.1(f), however, the samples here are generated using AR strategy. Comparing figures on the left with those on the right we observe that KDE places samples not only directly above the lines but also just below as is expected since we use smooth kernels. At the same time when using AR the samples tend to be more clustered which might be problematic if the estimate of VaR (or  $\mathbb{E}[f(\xi)]$ ) is not sufficiently accurate.

Results presented in Tables 6.1-6.4 show that, in general, the errors in estimates obtained with the Algorithm 3.1 increase with the dimensionality of the random inputs, and, thus, the advantage of it over plain Monte Carlo declines. In most cases the errors are within 1-3% with the notable exception of examples 6.3 and 6.4. In both cases the distributions of objective functions have long tails, which makes them hard to approximate accurately. The effect is especially pronounced when estimating  $\text{CVaR}_\beta$ . For these examples the estimates obtained with plain Monte Carlo sampling also exhibit large errors.

Another notable outcome is that in cases when the samples are generated using AR, the errors tend to be smaller for higher dimensions as compared to the errors when KDE is used. At the same time the percentage of samples in the upper tail tends to be smaller for AR. Table 6.2 shows the percentage of samples above  $\text{VaR}_\beta$  when estimating  $\text{CVaR}_\beta$  using KDE and AR.

As far as different types of distances used for distribution matching the best results when using KDE for sample generation are mostly achieved with  $L_2$ -type distances 5.13-5.15 and their weighted versions 5.16-5.17. Interestingly, for the pure CVaR and mean-CVaR risk functions the best results in high-dimensional case are often obtained by using the likelihood ratio approach. The LR approach, however, often fails for the mean-semideviation case due to large errors in estimation of the mean coming from the fact that LR weights do not sum to 1. The distances 5.8-5.11 utilizing Rosenblatt transformation tend to work better with AR sampling. There is, however, no clear winner, and it is not obvious which approach to adjusting the probabilities of the samples to take. We view this question as a main bottleneck for the proposed method.

Weighted distances 5.16-5.17 tend to perform better than their unweighted variants 5.13-5.15 for the cases when the volumes of important regions are two-thirds or less of the total volume (see Table 6.5 for the relevant information).

	$M = 2$	$M = 3$	$M = 5$	$M = 10$
Example 6.1	0.45	0.47	0.47	0.58
Example 6.2	0.08	0.12	0.1	0.18
Example 6.3	0.96	0.99	0.99	0.98
Example 6.4	0.48	0.47	0.55	0.59
Example 6.5	0.96	0.99	0.98	0.96

Table 6.5: Volumes of “important” regions (for  $\text{CVaR}_\beta$ ) as fractions of  $\tilde{\Xi}$  for distances 5.16-5.17.

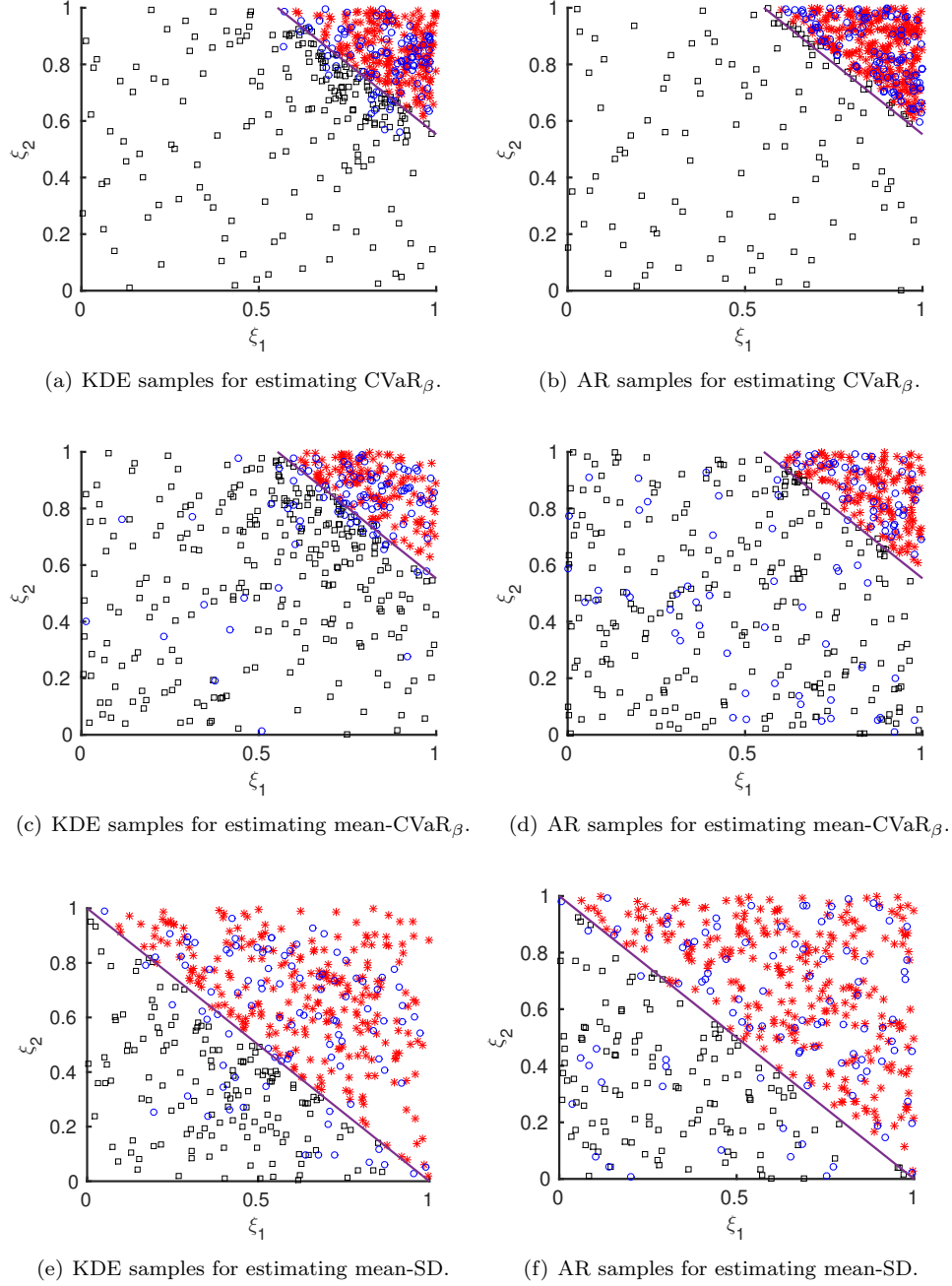


Fig. 6.1: Samples for example 6.2 with  $M = 2$  after step 4. Black squares are the samples from previous step that lie below the estimate of  $\text{VaR}_\beta$  for  $\text{CVaR}_\beta$  and mean- $\text{CVaR}_\beta$ , and below the estimate of expected value for mean-SD, red stars are the samples from the previous step that lie above the estimate of  $\text{VaR}_\beta$  for  $\text{CVaR}_\beta$  and mean- $\text{CVaR}_\beta$ , and above the estimate of expected value for mean-SD, and blue circles are the new samples generated at step 4. The lines correspond to  $f(\xi) = \text{VaR}_\beta[f(\xi)]$  for  $\text{CVaR}_\beta$  and mean- $\text{CVaR}_\beta$ , and  $f(\xi) = \mathbb{E}[f(\xi)]$  for mean-SD. Here  $\beta = 0.9$ .



**7. Conclusions and future work.** In this work we proposed an algorithm for estimating risk functions of random quantities of interest. We assume that the quantities of interest are expensive to estimate, e.g., require solving a PDE. Therefore, our algorithm intends to reduce the number of samples (i.e., PDE solves) required to estimate the risk functions. We consider the risk functions that include some notion of risk-averseness and are especially hard to compute in practice due to dependence on rare events (CVaR). Our algorithm is based on conjugate duality theory of convex risk functions and resembles non-parametric importance sampling.

We have benchmarked the proposed algorithm on simple example problems with known statistics. The results indicate the general effectiveness of the algorithm for problems with a few random inputs, however, more research is required for it to be useful in higher-dimensional cases.

One part of the algorithm is concerned with the generation of samples in the regions that are relevant to a given risk function. We have utilized KDE and AR approaches to this problem achieving a large increase in the number of said samples.

Another important part of the proposed algorithm is adjusting the probabilities of generated samples. We have tested several approaches to this problem including the traditional likelihood ratio approach and distribution matching approaches with various distance measures. More research is required in this direction, especially for the higher-dimensional problems, as the proposed approach to distribution matching becomes more difficult, and the likelihood ratio approach in some cases fails completely.

Note that the assumption of independence of random inputs made here in several places is not essential for the applicability of the proposed method and was mainly used as a simplification for defining some of the distance measures in Section 5.

In future work we want to consider building surrogates of the quantities of interest and using them to obtain probabilities for the samples produced by our algorithm. In other words we would like to substitute the problem of (multivariate) input distribution matching with the problem of matching the distribution of the surrogate of the quantity of interest. As indicated in Section 5 matching one-dimensional distributions is very easy, however, the complexity is shifted to the problem of building a surrogate model and quantifying its error.

## REFERENCES

- [1] S. AMARAL, D. ALLAIRE, AND K. WILLCOX, Optimal  $L_2$ -Norm Empirical Importance Weights for the Change of Probability Measure, *Stat. Comput.*, (2016), pp. 1–19.
- [2] P. ARTZNER, F. DELBAEN, J.-M. EBER, AND D. HEATH, Coherent Measures of Risk, *Math. Finance*, 9 (1999), pp. 203–228.
- [3] H.-J. BUNGARTZ AND M. GRIEBEL, Sparse Grids, *Acta Numerica*, 13 (2004), pp. 147–269.
- [4] S. N. CHIU AND K. I. LIU, Generalized Cramér–von Mises Goodness-of-Fit Tests for Multivariate Distributions, *Computational Statistics & Data Analysis*, 53 (2009), pp. 3817–3834.
- [5] D. A. DARLING, The Kolmogorov-Smirnov, Cramér-von Mises Tests, *The Annals of Mathematical Statistics*, 28 (1957), pp. 823–838.
- [6] C. A. DROSSOS AND A. N. PHILIPPOU, A Note on Minimum Distance Estimates, *Annals of the Institute of Statistical Mathematics*, 32 (1980), pp. 121–123.
- [7] J. M. HAMMERSLEY AND D. C. HANDSCOMB, *Monte Carlo Methods*, Methuen & Co., Ltd., London; Barnes & Noble, Inc., New York, 1965.
- [8] T. HESTERBERG, Weighted Average Importance Sampling and Defensive Mixture Distributions, *Technometrics*, 37 (1995), pp. 185–194.
- [9] F. J. HICKERNELL, A Generalized Discrepancy and Quadrature Error Bound, *Mathematics of Computation of the American Mathematical Society*, 67 (1998), pp. 299–322.
- [10] ———, Goodness-of-Fit Statistics, Discrepancies and Robust Designs, *Statistics & Probability Letters*, 44 (1999), pp. 73–78.
- [11] A. J. IZENMAN, *Modern Multivariate Statistical Techniques*, Springer Texts in Statistics, Springer, New York, 2008. Regression, Classification, and Manifold Learning.

- [12] A. JUSTEL, D. PEÑA, AND R. ZAMAR, A Multivariate Kolmogorov-Smirnov Test of Goodness of Fit, *Statistics & Probability Letters*, 35 (1997), pp. 251–259.
- [13] P. KROKHMAL, M. ZABARANKIN, AND S. URYASEV, Modeling and Optimization of Risk, *Surveys in Operations Research and Management Science*, 16 (2011), pp. 49 – 66.
- [14] J. MORIO, Extreme Quantile Estimation with Nonparametric Adaptive Importance Sampling, *Simulation Modelling Practice and Theory*, 27 (2012), pp. 76–89.
- [15] A. B. OWEN, *Monte Carlo Theory, Methods and Examples*, 2013. Available at <http://www-stat.stanford.edu/~owen/mc/> (accessed Nov 11, 2013).
- [16] K. PAVLIKOV AND S. URYASEV, CVaR Distance Between Univariate Probability Distributions and Approximation Problems, Tech. Rep. Research Report 2015-6, ISE Dept., University of Florida, 2016.
- [17] R. T. ROCKAFELLAR, *Convex Analysis*, Princeton University Press, Princeton, N.J., 1970.
- [18] R. T. ROCKAFELLAR, *Conjugate Duality and Optimization*, vol. 16, SIAM, 1974.
- [19] R. T. ROCKAFELLAR AND S. URYASEV, Optimization of Conditional Value-at-Risk, *The Journal of Risk*, 2 (2000), pp. 21–41.
- [20] M. ROSENBLATT, Remarks on a Multivariate Transformation, *The Annals of Mathematical Statistics*, 23 (1952), pp. 470–472.
- [21] A. RUSZCZYŃSKI AND A. SHAPIRO, Optimization of Convex Risk Functions, *Math. Oper. Res.*, 31 (2006), pp. 433–452.
- [22] D. W. SCOTT AND S. R. SAIN, Multidimensional Density Estimation, *Handbook of Statistics*, 24 (2005), pp. 229–261.
- [23] A. SHAPIRO, D. DENTCHEVA, AND A. RUSZCZYŃSKI, *Lectures on Stochastic Programming. Modeling and Theory*, vol. 9 of MPS/SIAM Series on Optimization, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA; Mathematical Programming Society (MPS), Philadelphia, PA, 2009.
- [24] L. SWILER AND N. WEST, Importance Sampling: Promises and Limitations, in 51st AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, Orlando, Florida. Paper AIAA 2010-2850, 2010.
- [25] P. ZHANG, Nonparametric Importance Sampling, *Journal of the American Statistical Association*, 91 (1996), pp. 1245–1253.

## INFORMATION THEORETIC REFINEMENT OF SEISMIC ARRIVAL TIME WITH UNCERTAINTY ESTIMATION

CHARLIE VOLLMER <sup>\*</sup>, MAX CHEN <sup>†</sup>, AND DAVID STRACUZZI <sup>‡</sup>

**Abstract.** Estimation of Signal Arrival Times for seismic sensors is the fundamental step before all downstream analyses of seismic activity. Any errors in the estimation are further exacerbated as the errors are propagated through location-detecting earth models. In this paper we detail a general framework for detecting seismic signal arrival time picks along with full estimation of their associated uncertainty. Using an STA/LTA threshold algorithm we can identify an Onset Search Window to be further refined through an Information Theoretic approach to estimate a precise Arrival Time Pick. Time Series statistical models are fit through an iterative algorithm to maximize the likelihood of the Arrival Time Pick estimate. The models can then be sampled through a Monte Carlo scheme to produce a full *a posteriori* estimate of uncertainty of Arrival Time of the seismic signal.

**1. Signal Detection.** Signals in the processed data channels are detected with an algorithm that computes the ratio of the short-term energy to the long-term energy: Short Term Average vs Long Term Average (STA/LTA)[4, 5]. Subsequently, a threshold is established for each detection channel in the associated beam recipe, that, when exceeded, identifies a potential “detection.” A detection is declared when the STA/LTA thresholds for a specified number of channels (typically one) have been exceeded. When the STA/LTA is above the threshold for a channel, that channel is in a “detecting state.” A channel in a detecting state remains in that state until the STA/LTA value stays below the threshold for a specified time interval. Output from the STA/LTA algorithm is a window of raw data points which contains a time series of two distinct signals: a signal of pure noise that changes at some time point into a noisy seismic signal reading (Figure 1.1).

The STA/LTA algorithm is designed for reliable detection of events, but only provides a crude estimate of the actual Onset/Arrival time of the event. This paper proposes a method of refining this crude estimate as well as providing a means of estimating a probability distribution around that estimate: a full measure of certainty.

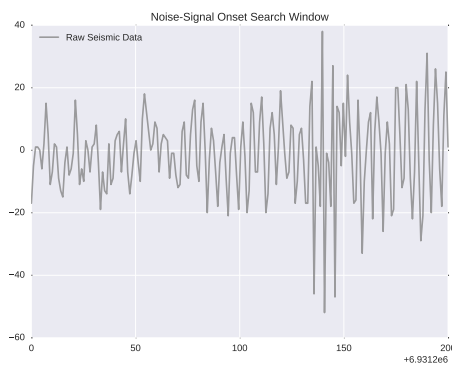


Fig. 1.1: Onset Search Window output from STA/LTA Detection

<sup>\*</sup>Dept. of Statistics, Colorado State University, charlesv@rams.colostate.edu

<sup>†</sup>Sandia National Laboratories, mgchen@sandia.gov

<sup>‡</sup>Sandia National Laboratories, djstrac@sandia.gov

**2. Estimating Arrival Time.** As was briefly outlined in Section 1, the initial STA/LTA algorithm gives a reliable -yet crude- detection of the arrival of a signal at the seismic sensor. The output from this initial first-pass algorithm is the Onset Search Window, shown in Figure 1.1.

This section will detail the methodological approach to refining the initial Arrival Time of the seismic signal, at the sensor. Our approach is computationally efficient and takes advantage of well-founded theory in statistical time series analysis. By following this approach, we also are afforded the luxury of being able to estimate a distribution over the Arrival Time; thus providing us a complete *a posteriori* estimate of the uncertainty of our Arrival Time estimate.

**2.1. Conceptual Approach to Estimating Arrival Time.** Since the initial detection output from the STA/LTA algorithm is a window of time series data which corresponds to a noise signal at which some time point becomes a signal with a noise component (Figure 1.1), we have the foundations to build an efficient algorithm to pinpoint the exact Arrival Time of the seismic signal.

We know that the initial data points in this time series are simply noise. Then -at some time point- the sensor begins to receive and record a signal composed of a signal process along with a noise component. Thus, we want to take advantage of these assumptions and simply fit two models,  $\mathcal{M}_1$  and  $\mathcal{M}_2$ , corresponding to each of the noise and signal data, respectively, and identify the time point in the series that corresponds to the best fit of these two models.

This approach suggests the use of a *sliding window*.

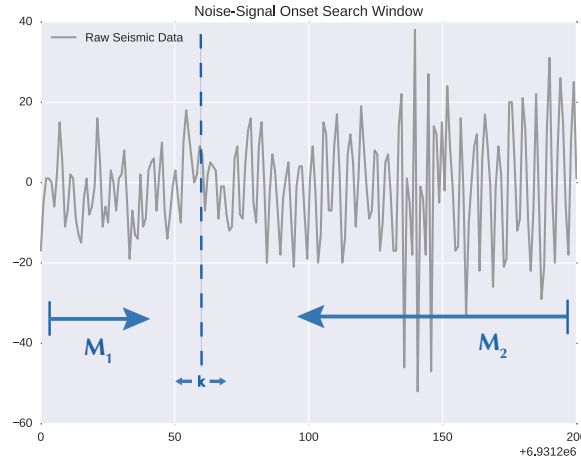


Fig. 2.1: Sliding Window Approach to Estimating Arrival Time

As shown in Figure 2.1, for every value of  $k$ , for  $k = 0, \dots, T$ , we fit the noise model,  $\mathcal{M}_1$ , to all data points  $y_t$  for  $t < k$ , and the signal model,  $\mathcal{M}_2$ , to all data points  $y_t$  for  $t \geq k$ .

Our estimate of the Arrival Time,  $\hat{k}$ , is simply the  $k$  that provides the “best fit” of the two models,  $\mathcal{M}_1$  and  $\mathcal{M}_2$ , simultaneously. Our method provides a description of “best fit” under an Information Theoretic framework.

**2.2. Proposed Initial Time Series Models.** Our approach uses two models,  $\mathcal{M}_1$  and  $\mathcal{M}_2$ , to effectively model the time series data that the seismic sensor is constantly recording. The first model,  $\mathcal{M}_1$ , is used for the purely noisy reading of the sensor. Since this is noise, it is natural to first model this as white noise: that is, as simple Gaussian random variables, each data point being an independent and identically distributed realization from a Gaussian (Normal) probability distribution with zero mean and finite variance, to be estimated from the data at each individual station.  $\mathcal{M}_1$  is specified in the following:

$$Y_t \sim N(0, \sigma_n^2) \quad (2.1)$$

where  $\sigma_n^2 < \infty$  for  $t = 1, \dots, k - 1$ .

The second model is used to model the generative process of the true seismic signal. Since we can assume that the sensor will record a noisy measurement of this true signal, the model must also explicitly model both noise and signal components during this time period.

Under the current proposal, we are assuming that while a signal is arriving and reaching the sensor, a weak signal will be followed by a weak signal, whereas a strong signal should immediately be followed by a strong signal. This is the same as assuming a smooth functional form to the signal, apart from the noisy component. We thus have a strong, natural justification for using an auto-regressive statistical model for modeling the seismic signal. Therefore, we will model the latter part of the Onset Search Window data points using an auto-regressive time series model with dependency between  $p$  adjacent points; AR(p) [2].  $\mathcal{M}_2$  is specified in the following:

$$Y_t = c + \sum_{i=1}^p \phi_i Y_{t-i} + \epsilon_t \quad (2.2)$$

where  $\epsilon_t \sim N(0, \sigma_s^2)$  for  $t = k, \dots, T$ . Here,  $\sigma_s^2$  is the finite variance of the noise component of the signal model.

Due to this explicit specification of our models and noise distributions, Equations 2.1 and 2.2, we immediately have likelihoods for our data and can use them to estimate not only the Arrival Time of the signal, but also the uncertainty around it. The Arrival Time estimate is detailed in the following section.

**2.3. Information Theoretic Refinement of Arrival Time Estimate.** The idea of refinement of the Arrival Time Estimates, from the given Onset Search Window from the STA/LTA algorithms, is based on evaluating the difference between the statistical models specified *a priori* of both the signal and noise processes and the observed statistics from our time series data. This paper proposes using an Akaike Information Criterion (AIC), [1], approach to measure the differences between these specified prediction models and the observed time-series data. AIC is a generalized entropy that provides a measure of the information difference between a mathematical process and a set of observations. The general form for AIC is the following:

$$AIC(\mathcal{M}) = -2 \log \mathcal{L}(\mathcal{M}) + 2\rho(\mathcal{M}) \quad (2.3)$$

where  $\mathcal{L}(\mathcal{M})$  is the likelihood of model  $\mathcal{M}$ , and  $\rho(\mathcal{M})$  is the complexity of the model (e.g. degrees of freedom).

In the following section we detail the likelihoods of our probability models, as well as how to implement our method.

**2.3.1. Stationary AR(1) Model.** For clarity, what follows is the treatment of a stationary autoregressive model of order one, AR(1), easily generalizable to larger dependence lags. A time series,  $Y_t$ , following this process can be written as

$$Y_t = c + \phi Y_{t-1} + \epsilon_t, \quad (2.4)$$

where  $\epsilon_t \sim N(0, \sigma^2)$ ,  $\theta = (c, \phi, \sigma^2)'$ , and  $|\phi| < 1$  for  $t = 1, \dots, T$ .

Let  $I_t = \{Y_0, \dots, Y_t\}$ . Since there is clearly dependence, we have that conditioned on  $I_{t-1}$ ,

$$Y_t | I_{t-1} \sim N(c + \phi Y_{t-1}, \sigma^2) \quad (2.5)$$

for  $t = 1, \dots, T$ .

The conditional density is

$$f(Y_t, Y_{t-1}, \theta) = (2\pi\sigma^2)^{-1/2} \exp\left\{-\frac{1}{2\sigma^2}(Y_t - c - \phi Y_{t-1})^2\right\} \quad (2.6)$$

for  $t = 2, \dots, T$ .

To derive the density of  $Y_1$ , we use the following property of stationarity:

$$E(Y_1) = \frac{c}{1 - \phi} \quad (2.7)$$

$$\text{Var}(Y_1) = \frac{\sigma^2}{1 - \phi^2}. \quad (2.8)$$

It follows that

$$Y_1 \sim N\left(\frac{c}{1 - \phi}, \frac{\sigma^2}{1 - \phi^2}\right). \quad (2.9)$$

The density of  $Y_1$  is

$$f(y_1 | \theta) = \left(2\pi \frac{\sigma^2}{1 - \phi^2}\right)^{-1/2} \exp\left\{-\frac{1 - \phi^2}{2\sigma^2} \left(y_1 - \frac{c}{1 - \phi}\right)^2\right\}. \quad (2.10)$$

Therefore, we have that the conditional log-likelihood function is

$$\begin{aligned} \sum_{t=2}^T \ln f(y_t | t_{t-1}, \theta) = \\ - \frac{(T-1)}{2} \ln(2\pi) - \frac{(T-1)}{2} \ln(\sigma^2) - \frac{1}{2\sigma^2} \sum_{t=2}^T (y_t - c - \phi y_{t-1})^2. \end{aligned} \quad (2.11)$$

This is the same as the log-likelihood of a simple linear regression model with normal errors. Therefore, the conditional Maximum Likelihood Estimates (MLEs) for  $c$  and  $\phi$  are identical to the least squares estimates.

The conditional MLE for  $\sigma$  is

$$\hat{\sigma}_{\text{MLE}}^2 = (T-1)^{-1} \sum_{t=2}^T (y_t - \hat{c}_{\text{CMLE}} - \hat{\phi}_{\text{CMLE}} y_{t-1})^2. \quad (2.12)$$

The marginal log-likelihood for the initial value  $Y_1$  is

$$\ln f(y_1|\theta) = -\frac{1}{2}\ln(2\pi) - \frac{1}{2}\ln\left(\frac{\sigma^2}{1-\phi^2}\right) - \left(\frac{1-\phi^2}{2\sigma^2}\right)\left(y_1 - \frac{c}{1-\phi}\right)^2. \quad (2.13)$$

Using all of this, the *exact* log-likelihood function is

$$\begin{aligned} l(\theta|y) = \ln L(\theta|y) = & -\frac{T}{2}\ln(2\pi) - \frac{1}{2}\ln\left(\frac{\sigma^2}{1-\phi^2}\right) - \frac{1-\phi^2}{2\sigma^2}\left(y_1 - \frac{c}{1-\phi}\right)^2 \\ & - \frac{(T-1)}{2}\ln(\sigma^2) - \frac{1}{2\sigma^2}\sum_{t=2}^T(y_t - c - \phi y_{t-1})^2. \end{aligned} \quad (2.14)$$

**2.3.2. Additive Likelihood of Noise-to-Signal Time Series.** Suppose we observe  $T$  data points on a seismic sensor,  $Y_t$ , for  $t = 1, \dots, T$ . We assume that there exists a time point  $\alpha \in \{t : t = 1, \dots, T\}$ , such that for all time points  $t \leq \alpha$  we record only noise. For all time points  $t$  after  $\alpha$ ,  $\alpha < t \leq T$ , we observe both a seismic signal and a noise component. By independence, we have directly that

$$f(Y_1, \dots, Y_T|\theta_1, \theta_2) = f(Y_1, \dots, Y_k|\theta_1)f(Y_{k+1}, \dots, Y_T|\theta_2). \quad (2.15)$$

Therefore, to find the MLE of the arrival time pick, we need to solve the following explicit optimization problem:

Find  $k$  that satisfies the following

$$\arg \max_k \{l(\theta_1|Y_1, \dots, Y_k) + l(\theta_2|Y_{k+1}, \dots, Y_T)\} \quad (2.16)$$

By our definitions of  $\mathcal{M}_1$  and  $\mathcal{M}_2$ , Equations 2.1 and 2.2, their log-likelihoods are

$$l(\theta_1|y_1, \dots, y_k) = -\frac{k}{2}\ln(2\pi) - \frac{k}{2}\ln(\sigma_n^2) - \frac{1}{2\sigma_n^2}\sum_{t=1}^k y_t^2 \quad (2.17)$$

$$\begin{aligned} l(\theta_2|y_{k+1}, \dots, y_T) = & -\frac{T-k}{2}\ln(2\pi) - \frac{1}{2}\ln\left(\frac{\sigma_s^2}{1-\phi^2}\right) \\ & - \frac{1-\phi^2}{2\sigma_s^2}\left(y_{k+1} - \frac{c}{1-\phi}\right)^2 - \frac{(T-k-1)}{2}\ln(\sigma_s^2) \\ & - \frac{1}{2\sigma_s^2}\sum_{t=k+2}^T(y_t - c - \phi y_{t-1})^2 \end{aligned} \quad (2.18)$$

Therefore, we have the following AIC criterion to minimize for our Arrival Time Estimation:

$$\begin{aligned}
l &= l(\theta_1|Y_1, \dots, Y_k) + l(\theta_2|Y_{k+1}, \dots, Y_T) \\
&= -\frac{k}{2} \ln(2\pi) - \frac{k}{2} \ln(\sigma_n^2) - \frac{1}{2\sigma_n^2} \sum_{t=1}^k y_t^2 - \frac{T-k}{2} \ln(2\pi) \\
&\quad - \frac{1}{2} \ln\left(\frac{\sigma_s^2}{1-\phi^2}\right) - \frac{1-\phi^2}{2\sigma_s^2} \left(y_{k+1} - \frac{c}{1-\phi}\right)^2 \\
&\quad - \frac{(T-k-1)}{2} \ln(\sigma_s^2) - \frac{1}{2\sigma_s^2} \sum_{t=k+2}^T (y_t - c - \phi y_{t-1})^2 \quad (2.19)
\end{aligned}$$

**3. Estimating Uncertainty of Arrival Time Estimate.** By specifying a model form for both our noise and signal time series,  $\mathcal{M}_1$  and  $\mathcal{M}_2$ , not only do we have a rigorous way of refining the Arrival Time Estimates of the signals from the STA/LTA algorithm output, but we also have a manner of using a Monte Carlo sampling scheme to derive an *a posteriori* distribution of the Arrival Time Estimate.

**3.1. Sampling from Statistical Time Series Models.** Since we are assuming a known functional form of the generative stochastic processes of our observed time series data, for both the noise and signal processes, we are able to easily generate random series of data after fitting the models  $\mathcal{M}_1$  and  $\mathcal{M}_2$ , as specified in Equations 2.1 and 2.2, and detailed in Equation 2.12.

**3.1.1. Sampling from Noise Process.** After fitting the noise model to the observed data for a particular station and sensor, we can then easily generate draws from the Gaussian distribution with the given mean and standard deviation values estimated:

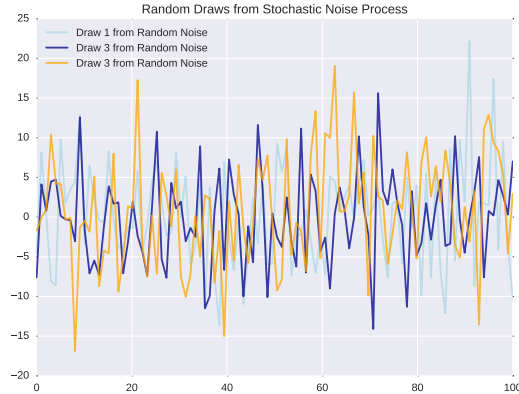


Fig. 3.1: Random Draws from Estimated Noise Process

The noise process can also follow a functional form that captures dependence -when appropriate. This would not cause any change to the algorithm itself, apart from swapping the likelihood of the white noise model for the dependence model.



**3.1.2. Sampling from Signal Process.** Just as easily as we are able to sample from the estimated Gaussian distribution for the noise process, we are also able to draw samples according to the estimated  $AR(p)$  process. This process has both a noise and signal component explicitly modeled, and it is clear to see a dependence in the following figure of an  $AR(2)$  process:

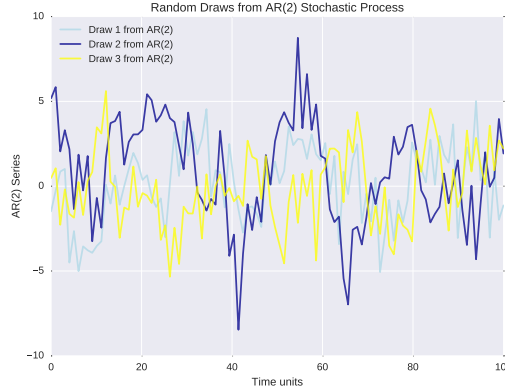


Fig. 3.2: Random Draws from Estimated Signal Process

Code to generate random draws from a stochastic  $AR(p)$  process is provided in the appendix of this document. The code generalizes the  $AR(p)$  to an  $ARMA(p,q)$  process, which is a time series model that explicitly incorporates an auto-regressive as well as moving average components, [2].

**3.2. A Monte Carlo Sampling Scheme.** Once we have established a method for sampling from both the noise and signal processes, we now have an immediate iterative method of estimating an *a posteriori* distribution of the Arrival Time Estimate.

Figure 3.3 illustrates how we simply make consecutive draws from each of the signal and noise processes and -for each pair of draws- we run the estimation algorithm of the Arrival Time detailed in Section 2 and illustrated in Figure 3.4.

In this manner, for each draw of the complete time series, we get a new estimate,  $k'$ , of the Arrival Time. As the two processes, noise and signal, get more similar, we will see confusion in our Arrival Time Estimation algorithm, giving more varied estimates: a heavier tailed distribution. As the two processes diverge from one another, we will see less variation in our Arrival Time estimates, giving a tighter distribution.

As seen in Figure 3.5, we have a complete posterior distribution of our Estimated Arrival Time, for further use in a downstream analysis of the seismic signal. A Kernel Density Estimate (KDE) is shown in Figure 3.5, as an approximation of the posterior distribution.

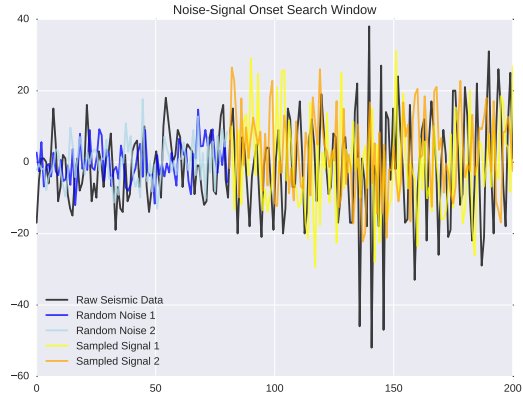


Fig. 3.3: Onset Search Window with samples drawn from  $\mathcal{M}_1$  and  $\mathcal{M}_2$

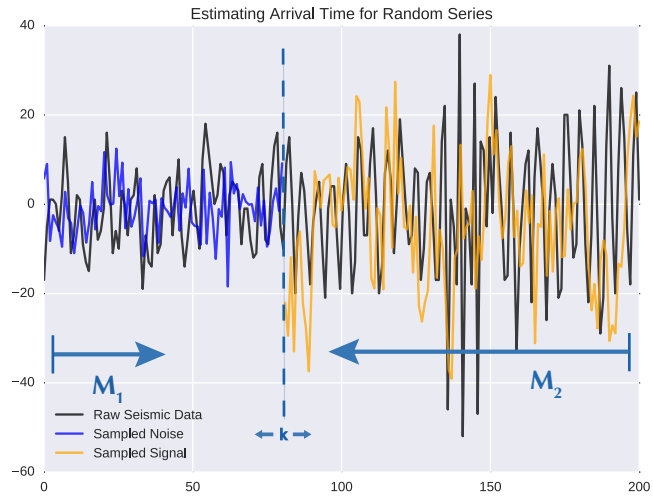


Fig. 3.4: Estimating Arrival Time for a Random Draw from Complete Process

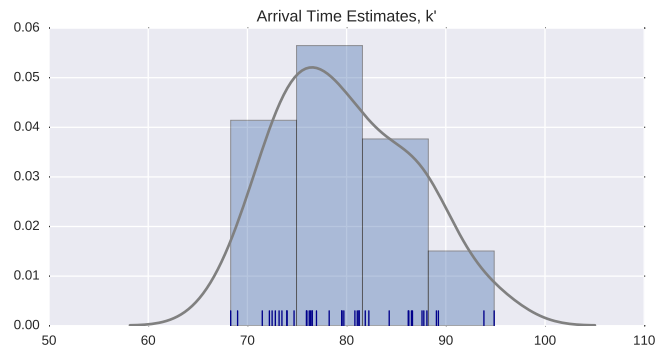


Fig. 3.5: Posterior Density Estimate of the Arrival Times,  $k'$

**4. Continued/Further Development.** While the development of the above estimation methodology, algorithms, and process is more or less complete, there is still quite a fair bit of refinement, validation, and the possibility of extensions to be explored in this work. Most of the immediately addressable topics include validation of the statistical models, calibration of the validated models, and extensions to the families of probability models used. These subdomains for future work are detailed in the following subsections.

**4.1. Validation of Statistical Time Series Models.** There are strong assumptions that need to be made when choosing the functional form of the statistical models,  $\mathcal{M}_1$  and  $\mathcal{M}_2$ , used to model the true, natural, generative stochastic processes. While motivated by physical processes their physical properties, the assumptions of White Noise and Auto-Regressive signal both need to be validated.

Noise could plausibly be of a functional form with a dependence structure. The true seismic signal could also be of a form not well-captured by an auto-regressive statistical model.

Of concern are things like particular devices at particular stations, as well as particular function forms for different arriving signals; i.e. ice sheet movement, seismic activity, human-induced explosions, etc. These concerns are also dealt with under the area of calibration, below.

**4.2. Calibration of Statistical Time Series Models.** While each of the models,  $\mathcal{M}_1$  and  $\mathcal{M}_2$ , and their parameters need to be estimated from the actual, observed data, they also need to be appropriately calibrated to their:

1. particular station
2. particular machine/sensor
3. variety of signal types; ie. ice sheet, seismic activity, explosions, etc.

It is quite feasible that each sensor records a different type of background noise and that the structure of such is distinct for each sensor. It is also plausible that each location has particularities to the data recorded. Perhaps different dependence structures need to be accounted for, and perhaps the signal-to-noise ratios are drastically different between stations. It is also fairly natural to assume that the form of the signals will be different depending on whether the waves be propagated due to natural or human-made causes.

Each of these points needs to be explicitly addressed and accounted for by any tool implementing the methodology detailed in this paper.

**4.3. Immediate Extensions to Auto-Regressive Time Series Models.** While the statistical models outlined in this paper were auto-regressive models, Equation 2.2, these models can easily be extended to include models of the moving average family of time series, as well. The **q'th order moving average** term in a time series model is a past error (multiplied by a coefficient), denoted  $\mathbf{MA}(\mathbf{q})$ :

$$Y_t = c + \epsilon_t + \sum_{i=1}^q \theta_i \epsilon_{t-i} \quad (4.1)$$

The generalization of Equations 2.2 and 4.1 is the ARIMA(p,d,q) model, which includes auto-regressive terms of lag  $p$ , moving average terms of lag  $q$ , and  $d$  differencing operations, [2]:

$$Y_t = c + \sum_{i=1}^p \phi_i Y_{t-i} + \sum_{i=1}^q \theta_i \epsilon_{t-i} + \epsilon_t \quad (4.2)$$

Python code (Python Software Foundation, <https://www.python.org/>), below, to simulate random draws from a stochastic time series of form **ARIMA(p,d,q)**, is also provided. This code relies on the Numpy [3] package in Python.

*Notice: an AR(2) process is equivalent to an ARIMA(2,0,0) time series.*

```

1  """
2  """
3  Random generation of Gaussian ARMA(p,q) time series.
4
5  phi:      An array of length p with the AR coefficients
6            (the AR part of the ARMA model).
7  theta:    An array of length q with the MA coefficients
8            (the MA part of the ARMA model).
9  sigma:    Standard deviation of the Gaussian noise.
10 n:        Length of the returned time-series.
11 burnin:   Number of data points that are going to be discarded
12           (the higher the better) to avoid dependence of the
13           ARMA time-series on the initial values.
14 """
15
16 from numpy import append, array
17 from numpy.random import normal
18 def ARMAgenerator(phi, theta, sigma, n, burnin=0, verbose=0):
19     l=max(len(phi), len(theta))
20     if(burnin==0):
21         burnin=10*l # Burn-in elements!
22     w=normal(0, sigma, n+burnin)
23     ARMA=array([])
24     s=0.0
25     l=max(len(phi), len(theta))
26     for i in range(n+burnin):
27         if(i<l):
28             ARMA=append(ARMA, w[i])
29         else:
30             s=0.0
31             for j in range(len(phi)):
32                 s=s+phi[j]*ARMA[i-j-1]
33             for j in range(len(theta)):
34                 s=s+theta[j]*w[i-j-1]
35             ARMA=append(ARMA, s+w[i])
36     if(verbose!=0):
37         print('Measured standard deviation: '
38               +str(sqrt(var(w[burnin:]))))
39     return ARMA[burnin:]

```

## REFERENCES

- [1] H. AKAIKE, A New Look at Statistical Model Identification, IEEE Transactions on Automatic Control, 19 (1974), pp. 716–723.
- [2] P. J. BROCKWELL AND R. A. DAVIS, *Time Series: Theory and Methods*, Springer, 1991.
- [3] E. JONES, T. OLIPHANT, P. PETERSON, ET AL., SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 2016-11-23].
- [4] D. WAHL, Programmer's Guide for the Detection and Feature Extraction Program, Science Applications International, SAIC-96 (1996), p. 1069.
- [5] ———, User's Manual for the Detection and Feature Extraction Program, Science Applications International, SAIC-96 (1996), p. 1098.

## OPTIMAL EXPERIMENTAL DESIGN USING A CONSISTENT BAYESIAN APPROACH

SCOTT N. WALSH\*, TIM M. WILDEY†, AND JOHN D. JAKEMAN‡

**Abstract.** We consider the utilization of a computational model to guide the optimal acquisition of experimental data to inform the stochastic description of model input parameters. Our formulation is based on the recently-developed *consistent* Bayesian approach for solving stochastic inverse problems, which seeks a posterior probability density that is consistent with the model and the data in the sense that the push-forward of the posterior (through the computational model) matches the observed density on the observations almost surely. Given a set of potential observations, our optimal experimental design (OED) seeks the observation, or set of observations, that maximizes the expected information gain from the prior probability density on the model parameters. We discuss the characterization of the space of observed densities and a computationally-efficient approach for rescaling observed densities to satisfy the fundamental assumptions of the consistent Bayesian approach. Numerical results are presented to compare our approach with existing OED methodologies using the classical/statistical Bayesian approach and to demonstrate our OED on a set of representative PDE-based models.

**1. Introduction.** Experimental data is often used to infer valuable information about parameters for models of physical systems. However, the collection of experimental data can be costly and time consuming. For example, exploratory drilling can reveal valuable information about subsurface hydrocarbon reservoirs, but each well can cost upwards of tens of millions of US dollars. In situations such as this, we can only afford to gather some limited amount of experimental data. Hence, we are motivated to design experiments in an optimal way, i.e., choose some limited amount of experimental data to maximize the value of the experiment.

We are primarily interested in the situation where experimental data is not yet available to guide the optimal experimental design (OED). Therefore, we can only seek to maximize the expected information gain, i.e., the average information gain over all possible realizations of a given set of experimental data. The determination of an OED is based largely on the computational model of the physical system. With no experimental data available, the physics and sensitivities of the computational model greatly impact our definition of the OED.

Determining optimal data to infer information about model input parameters is an emerging field of research. A general review of Bayesian experimental design is given in [7] and the Bayesian analogues of the so-called alphabetical design criteria are introduced and examined. In [8, 9] Bayesian OED is developed for physically-realistic and hence computationally-intensive models, which motivates the authors to explore the use of surrogate models to accelerate the computation of the expected information gain. In [3] the authors accelerate the computation of the expected information gain by approximating the posterior density with a truncated Gaussian density. In the context of OED for inference, analogues of the alphabetic optimality criterion for linear models have also been applied to nonlinear models [1, 2, 11]. In a recent work [6], an OED based on a measure theoretic approach for solving stochastic inverse problems is developed. This approach only uses local linear approximations of QoI maps and their singular values to approximate the OED.

The approach taken in this paper is similar to [8] in that we seek an OED that maximizes the expected information gain from a prior to a posterior over the set of possible observational densities. In [8], the authors use the classical/statistical Bayesian approach

---

\*University of Colorado Denver, scott.walsh@ucdenver.edu

†Sandia National Laboratories, tmwilde@sandia.gov

‡Sandia National Laboratories, jdjakem@sandia.gov

for stochastic inference (see e.g., [13]) whereby an error model is used to define a likelihood which is then used to compute the posterior. Our OED is based on the *consistent* Bayesian approach introduced in [5], which utilizes the push-forward of the prior to define a posterior density. This posterior is consistent with the model and the data in the sense that the push-forward of the posterior (through the computational model) matches the observed density almost surely. We direct the interested reader to [5] for a discussion on the differences between the consistent and classical Bayesian approaches.

The remainder of this paper is outlined as follows. In Section 2 we present the consistent Bayesian method for solving stochastic inverse problems. In Section 3 we discuss the information content of an experiment, and present our optimal experimental design formulation based upon expected information gain. During the process of defining the expected information gain of a given experimental design, care must be taken to ensure that model can predict all of our potential observed data. In Section 4 we discuss situations for which this assumption is violated and means for avoiding these situations. Numerical examples are presented in Section 5 and concluding remarks are provided in Section 6.

**2. A Consistent Bayes formulation for stochastic inverse problems.** In this section we present an overview of the consistent Bayes formulation of the stochastic inverse problem. This derivation of the posterior differs from the statistical Bayesian counterpart (see e.g. [13]) in that the posterior satisfies a consistency requirement, i.e., the posterior is consistent with the model and the observed data. In Section 3 we present numerical results for a simple nonlinear system to illustrate this consistency requirement.

**2.1. Notation, Assumptions, and a Stochastic Inverse Problem.** Let  $M(Y, \lambda)$  denote a deterministic model with solution  $Y(\lambda)$  that is an implicit function of model parameters  $\lambda \in \Lambda \subset \mathbb{R}^n$ . The set  $\Lambda$  represents the largest physically meaningful domain of parameter values, and, for simplicity, we assume that  $\Lambda$  is compact. In practice, modelers are often only concerned with computing a relatively small set of quantities of interest (QoI),  $\{Q_i(Y)\}_{i=1}^m$ , where each  $Q_i$  is a real-valued functional dependent on the model solution  $Y$ . Since  $Y$  is a function of parameters  $\lambda$ , so are the QoI and we write  $Q_i(\lambda)$  to make this dependence explicit. Given a set of QoI, we define the QoI map  $Q(\lambda) := (Q_1(\lambda), \dots, Q_m(\lambda))^T : \Lambda \rightarrow \mathcal{D} \subset \mathbb{R}^m$  where  $\mathcal{D} := Q(\Lambda)$  denotes the range of the QoI map.

Assume  $(\Lambda, \mathcal{B}_\Lambda, \mu_\Lambda)$  and  $(\mathcal{D}, \mathcal{B}_\mathcal{D}, \mu_\mathcal{D})$  are measure spaces. We assume  $\mathcal{B}_\Lambda$  and  $\mathcal{B}_\mathcal{D}$  are the Borel  $\sigma$ -algebras inherited from the metric topologies on  $\mathbb{R}^n$  and  $\mathbb{R}^m$ , respectively. The measures  $\mu_\Lambda$  and  $\mu_\mathcal{D}$  are volume measures.

We assume that the QoI map  $Q$  is at least piecewise smooth implying that  $Q$  is a measurable map between the measurable spaces  $(\Lambda, \mathcal{B}_\Lambda)$  and  $(\mathcal{D}, \mathcal{B}_\mathcal{D})$ . For any  $A \in \mathcal{B}_\mathcal{D}$ , we then have

$$Q^{-1}(A) = \{\lambda \in \Lambda \mid Q(\lambda) \in A\} \in \mathcal{B}_\Lambda, \quad \text{and} \quad Q(Q^{-1}(A)) = A.$$

Furthermore, given any  $B \in \mathcal{B}_\Lambda$ ,

$$B \subseteq Q^{-1}(Q(B)), \tag{2.1}$$

although we note that in most cases  $B \neq Q^{-1}(Q(B))$  even when  $n = m$  (see Figure 2.1 where  $n = 2$  and  $m = 1$  for an illustration).

The final assumption is that an observed probability measure,  $P_\mathcal{D}^{\text{obs}}$ , is given on  $(\mathcal{D}, \mathcal{B}_\mathcal{D})$ , which can be described in terms of an observed probability density,  $\pi_\mathcal{D}^{\text{obs}}$ . The stochastic inverse problem is then defined as determining a probability measure,  $P_\Lambda$ , described as a



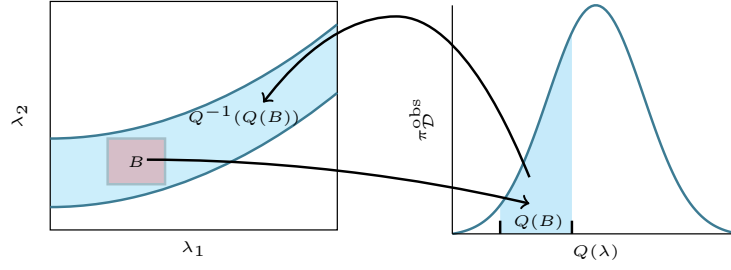


Fig. 2.1: Illustration of the mapping  $Q : \Lambda \rightarrow \mathcal{D}$ . Note that  $Q$  maps  $B$  to a particular region in  $\mathcal{D}$  ( $Q(B)$ ), and while the inverse image of this set, given by  $Q^{-1}(Q(B))$ , contains  $B$ , other points in  $\Lambda$  may also map to  $Q(B)$ .

probability density,  $\pi_\Lambda$ , such that, the push-forward measure agrees with  $P_{\mathcal{D}}^{\text{obs}}$ . We use  $P_{\mathcal{D}}^{Q(P_\Lambda)}$  to denote the push-forward of  $P_\Lambda$  through  $Q(\lambda)$ , i.e.,

$$P_{\mathcal{D}}^{Q(P_\Lambda)}(A) = P_\Lambda(Q^{-1}(A)).$$

for all  $A \in \mathcal{B}_{\mathcal{D}}$ . Using this notation, a solution to the stochastic inverse problem is defined formally as follows:

**DEFINITION 2.1 (Consistency).** *Given a probability measure  $P_{\mathcal{D}}^{\text{obs}}$  described as a density  $\pi_{\mathcal{D}}^{\text{obs}}$  with respect to  $\mu_{\mathcal{D}}$  on  $(\mathcal{D}, \mathcal{B}_{\mathcal{D}})$ , the stochastic inverse problem seeks a probability measure  $P_\Lambda$  described as a probability density  $\pi_\Lambda$  with respect to  $\mu_\Lambda$ , on  $(\Lambda, \mathcal{B}_\Lambda)$ , such that the subsequent push-forward measure induced by the map,  $Q(\lambda)$ , satisfies*

$$P_\Lambda(Q^{-1}(A)) = P_{\mathcal{D}}^{Q(P_\Lambda)}(A) = P_{\mathcal{D}}^{\text{obs}}(A), \quad (2.2)$$

for any  $A \in \mathcal{B}_{\mathcal{D}}$ . We refer to any probability measure  $P_\Lambda$  that satisfies Eq. 2.2 as a **consistent** solution to the stochastic inverse problem.

Clearly, a consistent solution may not be unique, i.e., there may be multiple probability measures that are consistent in the sense of Definition 2.1. A unique solution may be obtained by imposing additional constraints or structure on the stochastic inverse problem. In this paper, such structure is obtained by incorporating prior information to construct a unique Bayesian solution to the stochastic inverse problem.

**2.2. A Bayesian solution to the stochastic inverse problem.** Following the Bayesian philosophy [14], we introduce a finite *prior* probability measure  $P_\Lambda^{\text{prior}}$  described as a probability density  $\pi_\Lambda^{\text{prior}}$  on  $\Lambda$  with respect to  $\mu_\Lambda$ . The prior probability measure encapsulates the existing knowledge about the uncertain parameters.

Assuming that  $Q$  is at least measurable, then the prior probability measure on  $\Lambda$ ,  $P_\Lambda^{\text{prior}}$ , and the map,  $Q$ , induce a push-forward measure  $P_{\mathcal{D}}^{Q(\text{prior})}$  on  $\mathcal{D}$ , which is defined for all  $A \in \mathcal{B}_{\mathcal{D}}$ ,

$$P_{\mathcal{D}}^{Q(\text{prior})}(A) = P_\Lambda^{\text{prior}}(Q^{-1}(A)). \quad (2.3)$$

We utilize the following expression for the posterior,

$$P_\Lambda^{\text{post}}(B) := \begin{cases} P_\Lambda^{\text{prior}}(B) \frac{P_{\mathcal{D}}^{\text{obs}}(Q(B))}{P_{\mathcal{D}}^{Q(\text{prior})}(Q(B))}, & \text{if } P_\Lambda^{\text{prior}}(B) > 0, \\ 0, & \text{otherwise,} \end{cases} \quad (2.4)$$

which we describe in terms of a probability density given by

$$\pi_{\mathbf{\Lambda}}^{\text{post}}(\lambda) = \pi_{\mathbf{\Lambda}}^{\text{prior}}(\lambda) \frac{\pi_{\mathcal{D}}^{\text{obs}}(Q(\lambda))}{\pi_{\mathcal{D}}^{Q(\text{prior})}(Q(\lambda))}, \quad \lambda \in \mathbf{\Lambda}. \quad (2.5)$$

We note that if  $\pi_{\mathcal{D}}^{Q(\text{prior})} = \pi_{\mathcal{D}}^{\text{obs}}$ , i.e., if the prior solves the stochastic inverse problem, then the posterior density will be equal to the prior density.

It was recently shown in [5] that the posterior given by Eq. 2.4 defines a consistent probability measure using a *contour*  $\sigma$ -algebra and when interpreted as an iterated integral of Eq. 2.5 defines a probability measure on  $(\mathbf{\Lambda}, \mathcal{B}_{\mathbf{\Lambda}})$  that is consistent in the sense of Definition 2.1, i.e., the push-forward of the posterior matches the observed probability measure almost surely. Approximating the posterior density using the consistent Bayesian approach only requires an approximation of the push-forward of the prior probability on the model parameters, which is fundamentally a forward propagation of uncertainty. While numerous approaches have been developed in recent years to improve the efficiency and accuracy of the forward propagation of uncertainty using computational models, we only consider the most basic of methods, namely Monte Carlo sampling, to sample from the prior. We evaluate the computational model for each of the samples from the prior and use a standard kernel density estimator to approximate the push-forward of the prior. Given the approximation of the push-forward of the prior, we can evaluate the posterior at any point  $\lambda \in \mathbf{\Lambda}$  if we have  $Q(\lambda)$ . Thus, we can either construct an approximation of the posterior or generate samples from the posterior using straightforward rejection sampling.

In practice, we prefer to use data that is sensitive to the parameters since otherwise it is difficult to infer useful information about the uncertain parameters. Specifically, if  $m \leq n$  and the Jacobian of  $Q$  is defined a.e. in  $\mathbf{\Lambda}$  and is full rank a.e., then the push forward volume measure  $\mu_{\mathcal{D}}$  is absolutely continuous with respect to the usual Lebesgue measure [5].

For the rest of this work we maintain the following assumptions needed to produce a unique consistent solution to the stochastic inverse problem:

**(A1)** We have a mathematical model and a description of our prior knowledge about the model input parameters,

**(A2)** The data exhibits sensitivity to the parameters a.e. in  $\mathbf{\Lambda}$ , hence, we use the Lebesgue measure  $\mu$  as the volume measure on the data space,

**(A3)** The observed density is absolutely continuous with respect to the push forward of the prior, i.e., the model can reach all of the observed data.

In the remainder of this work, we focus on quantifying the value of these posterior densities. We employ a standard technique, the Kullback-Leibler divergence [15], to measure the information gained about the parameters from the prior to the posterior. We compute the expected information gain of a given set of QoI (a given experimental design), and then determine the OED to deploy in the field.

The consistent Bayesian approach described in this section and the statistical Bayesian approach typically used to solve stochastic inverse problems are similar in the sense that both approaches use prior information to determine a posterior probability density on model inputs given observed data on QoIs. However, the two approaches are fundamentally different in the way the problem is formulated. The statistical Bayesian approach introduces stochasticity through a likelihood model, often called an error model, and defines the posterior as a conditional probability density. This introduction of the likelihood transforms a deterministic mapping between parameters and QoI into a stochastic map. Samples from the posterior can be generated using a Markov Chain Monte Carlo (MCMC) method without directly constructing the posterior. On the other hand, the consistent Bayesian uses

the push-forward of the prior to define a posterior density on the parameters that satisfies the consistency requirement. Given the push-forward of the prior, the posterior can be evaluated directly, which opens up opportunities for methods other than MCMC and even allows us to explore the effect that varying the observed density has on the posterior which we exploit for efficient optimal experimental design.

**3. The information content of an experiment.** We assume we are given a model and some prior knowledge of the model input parameters. A posterior that is different from the prior is considered to be informative about the parameters, and posteriors that are more different from the prior are considered more informative. We quantify this *information gain*, as our knowledge of the parameters changes from the prior to a given posterior, using the Kullback-Leibler divergence [15]. Then, we discuss a simple nonlinear model with two different QoI that produce very different posteriors. Lastly, we define the *expected information gain* for the map  $Q$  as the average information gain over a specific space of possible observed densities.

**3.1. Information gain: Kullback-Leibler divergence.** Suppose we are given a description of the uncertainty on the observed data in terms of a probability density  $\pi_{\mathcal{D}}^{\text{obs}}$ . This produces a unique solution to the stochastic inverse problem,  $\pi_{\Lambda}^{\text{post}}$ . In [5] it was shown that  $\pi_{\Lambda}^{\text{prior}}$  and  $\pi_{\Lambda}^{\text{post}}$  are absolutely continuous with respect to the Lebesgue measure  $\mu_{\Lambda}$ . From [15] it follows that the Kullback-Leibler divergence (information gain), denoted  $I_Q$ , is given by

$$I_Q(\pi_{\Lambda}^{\text{prior}} : \pi_{\Lambda}^{\text{post}}) := \int_{\Lambda} \pi_{\Lambda}^{\text{post}} \log \left( \frac{\pi_{\Lambda}^{\text{post}}}{\pi_{\Lambda}^{\text{prior}}} \right) d\mu_{\Lambda}, \quad (3.1)$$

where we choose the volume measure,  $\mu_{\Lambda}$ , to be a scaled Lebesgue measure such that  $\mu_{\Lambda}(\Lambda) = 1$ . This choice of  $\mu_{\Lambda}$  impacts the scaling of  $\pi_{\Lambda}^{\text{prior}}$  and  $\pi_{\Lambda}^{\text{post}}$  which in turn produces an information gain that is independent of the measure of the parameter space, allowing us to compare  $I_Q$  values between applications.

Note that because  $\pi_{\Lambda}^{\text{prior}}$  is fixed,  $I_Q$  is simply a function of the posterior

$$I_Q(\pi_{\Lambda}^{\text{prior}} : \pi_{\Lambda}^{\text{post}}) = I_Q(\pi_{\Lambda}^{\text{post}}), \quad (3.2)$$

and from Eq. 2.5 the posterior is a function of the observed density. Therefore, we write  $I_Q$  as a function of the observed density,

$$I_Q(\pi_{\Lambda}^{\text{post}}) = I_Q(\pi_{\mathcal{D}}^{\text{obs}}). \quad (3.3)$$

The observation that  $I_Q$  is a function of only  $\pi_{\mathcal{D}}^{\text{obs}}$  allows us to define the expected information gain in Section 3.3 based on a specific space of observed densities.

**3.2. A simple nonlinear system.** Consider the following 2-component nonlinear system of equations with two parameters introduced in [4]:

$$\begin{aligned} \lambda_1 x_1^2 + x_2^2 &= 1 \\ x_1^2 - \lambda_2 x_2^2 &= 1 \end{aligned}$$

The first QoI is the second component, i.e.,  $q_1(\lambda) = x_2(\lambda)$ . The parameter ranges are given by  $\lambda_1 \in [0.79, 0.99]$  and  $\lambda_2 \in [1 - 4.5\sqrt{0.1}, 1 + 4.5\sqrt{0.1}]$  which are chosen as in [4] to induce an interesting variation in the QoI. We assume the observed density on  $q_1$  is a truncated normal distribution with mean 0.3 and standard deviation of 0.01, see Figure 3.1 (right).

An analytical solution to this stochastic inverse problem is not available, so we generate 40,000 samples from the uniform prior and use a kernel density estimator (KDE) to construct an approximation to the resulting push forward density. Then we use Eq. 2.5 to construct an approximation to the posterior density on a  $200 \times 200$  uniform grid, see Figure 3.1 (left), and a simple accept/reject algorithm to generate a set of samples from the posterior (Figure 3.1). We propagate this set of samples from the posterior through the model and approximate the resulting push forward of the posterior density using a KDE. In Figure 3.1 (right) we see the push forward of the posterior agrees quite well with the observed density.

Notice the support of the posterior lies in a relatively small region of the parameter space. The information gain from this posterior is  $I_{q_1}(\pi_{\mathcal{D}}^{\text{obs}}) = 2.015$ . Next, we consider a different QoI to use in the inverse problem, and compare the support of its posterior to the one we just observed.

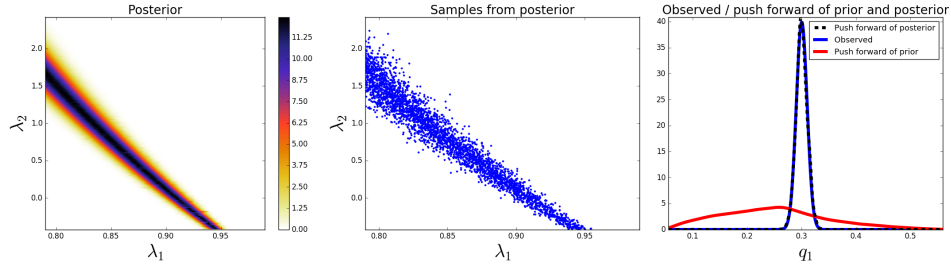


Fig. 3.1: (left) Approximation of the posterior density obtained using the data  $q_1$  which gives  $I_{q_1}(\pi_{\mathcal{D}}^{\text{obs}}) = 2.015$ , (middle) a set of samples from the posterior, (right) and a comparison of the observed density on  $q_1$  with the push forward densities of the prior and the posterior.

Consider a second QoI,

$$q_2(\lambda) = x_1.$$

We assume the observed density on  $q_2$  is a truncated normal distribution with mean 1.015 and standard deviation of 0.01. We approximate the push forward density and the posterior using the same 40,000 samples and again generate a set of samples from the posterior and propagate these samples through the model to approximate the push forward of the posterior, see Figure 3.2.

Although both  $q_1$  and  $q_2$  have the same standard deviation in their observed densities, clearly the two QoI produce very different posterior densities. The posterior corresponding to data from  $q_2$  has a much larger region of support within the parameter space compared to that of the posterior corresponding to  $q_1$ . This is quantified with the information gain from this posterior  $I_{q_2}(\pi_{\mathcal{D}}^{\text{obs}}) = 0.466$ . Given these two maps,  $q_1$  and  $q_2$ , and the specified observed data on each of these data spaces, the data  $q_1$  is more *informative* of the parameters than the data  $q_2$ .

Next, we consider using the data from both  $q_1$  and  $q_2$ ,  $Q : \Lambda \rightarrow (q_1, q_2)$ , with the same means and standard deviations as specified above. Again, we approximate the push forward density and the posterior using the same 40,000 samples (Figure 3.3).

With the information from both  $q_1$  and  $q_2$  we see a substantial decrease in the support of the posterior density. Intuitively, the support of the posterior using both  $q_1$  and  $q_2$  is the support of the posterior using  $q_1$  intersected with the support of the posterior using  $q_2$ . This is quantified in the information gain of this posterior  $I_Q(\pi_{\mathcal{D}}^{\text{obs}}) = 2.98$ .

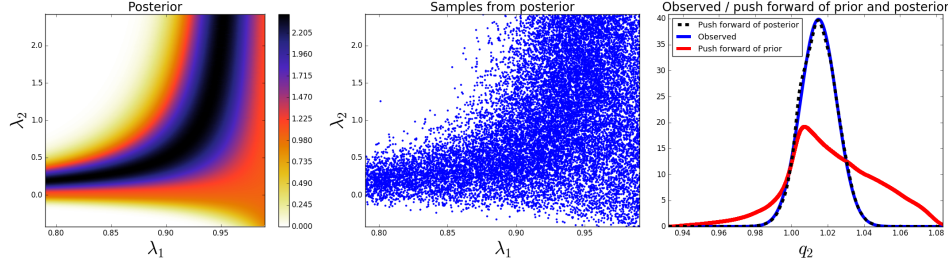


Fig. 3.2: (left) Approximation of the posterior density obtained using  $q_2$  which gives  $I_{q_2}(\pi_{\mathcal{D}}^{\text{obs}}) = 0.466$ , (middle) a set of samples from the posterior, (right) and a comparison of the observed density on  $q_2$  with the push forward densities of the prior and the posterior.

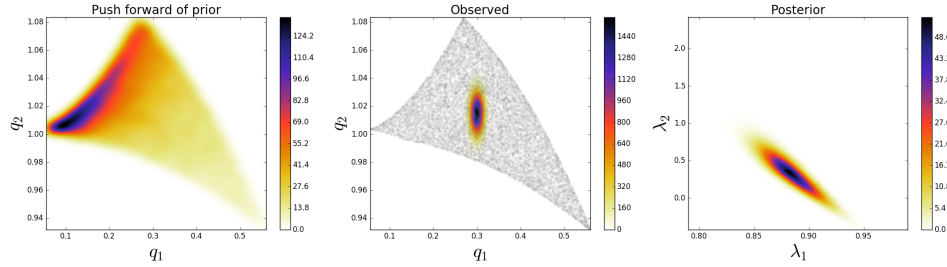


Fig. 3.3: (left) The approximation of the push forward of the prior, (middle) the exact observed density on  $(q_1, q_2)$ , (right) the approximation of the posterior density using both  $q_1$  and  $q_2$  which gives  $I_Q(\pi_{\mathcal{D}}^{\text{obs}}) = 2.98$ .

In the scenario in which we can afford to gather data on both  $q_1$  and  $q_2$ , we benefit greatly in terms of reducing the uncertainties on the model input parameters. However, suppose we could only afford to gather one of these QoI in the field. Based on the information gain from each posterior,  $q_1$  is more informative about the parameters than  $q_2$ . However, consider a scenario in which the observed data has different means in both  $q_1$  and  $q_2$ . Due to the nonlinearities of the maps, it is not necessarily true that  $q_1$  is still more informative than  $q_2$ . If we do not know the mean of the data for either  $q_1$  or  $q_2$ , then we want to determine which of these QoI we *expect* to produce the most informative posterior.

**3.3. Expected information gain.** Our goal is to quantify the *expected* information gain of a given experimental design. Physical data is not yet available because we are seeking to determine the optimal data to collect. Hence, our only resources are the computational data from the model of the physical system,  $Q$ , and the prior knowledge of the parameters,  $\pi_{\Lambda}^{\text{prior}}$ . We reiterate, we do not have data available to define  $\pi_{\mathcal{D}}^{\text{obs}}$ . Therefore, our goal is to define the expected information gain based on *potential*  $\pi_{\mathcal{D}}^{\text{obs}}$ . Let  $\mathcal{O}$  denote the space of densities over  $\mathcal{D}$ . We want to define the expected information gain as some kind of average over this density space in some meaningful way. However, this is far too general of a space to use to define the expected information gain. This space includes densities that are very unlikely to be observed in reality. Therefore, we restrict  $\mathcal{O}$  to be a space more representative of densities that may be observed in reality.

With no experimental data available to specify an observed density on a single QoI, we assume the density is a truncated Gaussian with a standard deviation determined by some estimate of the measurement instrument error. With Gaussians of (possibly) varying standard deviations specified for each QoI, this defines the shape of the observed densities we consider. We let  $\mathcal{O}_{\mathcal{D}}$  denote the space of all densities of this shape centered in  $\mathcal{D} = Q(\Lambda)$ ,

$$\mathcal{O}_{\mathcal{D}} = \left\{ N(q, \sigma^2) : q \in \mathcal{D} \right\}, \quad (3.4)$$

where  $N(q, \sigma^2)$  is a Gaussian function with mean  $q$  and standard deviation  $\sigma$ . More details of this definition of  $\mathcal{O}_{\mathcal{D}}$  are addressed in Section 4. We can easily generalize our description of  $\mathcal{O}$ . For example, we could also consider the standard deviation of the observed data to be uncertain, in which case we would also average over some interval of possible values for  $\sigma$ . However, in this work we only vary the center of the Gaussian densities.

REMARK 1. *We can restrict  $\mathcal{O}$  in other ways as well. For example, if we expect the uncertainty in each QoI to be described by a uniform density, then we define the restriction on  $\mathcal{O}$  accordingly. This choice of shape for the observed density space is largely dependent on the application. The only limitation is we require the measure specified on the observed density space reflects the push forward measure on the data space,  $\mu_{\mathcal{D}}$ , as described below. In Section 5.3 we describe one approach for defining a restricted observed density space where the observed density of each QoI has a Gaussian profile and the standard deviations are functions of the magnitudes of each QoI.*

The restriction of possible  $\pi_{\mathcal{D}}^{\text{obs}}$  to this specific space of densities allows us to represent each density uniquely with a single point  $q \in \mathcal{D}$ . Based on our prior knowledge of the parameters and the sensitivities of the map  $Q$ , the model informs us that some data are more likely to be observed than other data, this is seen in the plot of  $\pi_{\mathcal{D}}^{Q(\text{prior})}$  in Figure 3.3 (middle). This implies we do not want to average over  $\mathcal{D}$  with respect to  $\mu$  or  $\mu_{\mathcal{D}}$ , but rather with respect to the push forward of the prior on  $\mathcal{D}$ ,  $P_{\mathcal{D}}^{Q(\text{prior})}$ . This respects the sensitivity information provided by the model and weights the data appropriately. We define the *expected information gain*, denoted  $E(I_Q)$ , as just described,

$$E(I_Q) := \int_{\mathcal{D}} I_Q(q) \pi_{\mathcal{D}}^{Q(\text{prior})}(q) d\mu = \int_{\mathcal{D}} I_Q(q) dP_{\mathcal{D}}^{Q(\text{prior})}. \quad (3.5)$$

From Eq. 3.1,  $I_Q$  itself is defined in terms of an integral. The expanded form for  $E(I_Q)$  is then an iterated integral,

$$E(I_Q) = \int_{\mathcal{D}} \int_{\Lambda} \pi_{\Lambda}^{\text{post}}(\lambda; q) \log \left( \frac{\pi_{\Lambda}^{\text{post}}(\lambda; q)}{\pi_{\Lambda}^{\text{prior}}(\lambda)} \right) d\mu_{\Lambda} dP_{\mathcal{D}}^{Q(\text{prior})}, \quad (3.6)$$

where we make explicit that  $\pi_{\Lambda}^{\text{post}}$  is a function of the observed density and, by our restriction of the space of observed densities in Eq. 3.4, therefore a function of  $q \in \mathcal{D}$ . We utilize Monte Carlo sampling to approximate the integral in Eq. 3.5 as described in Algorithm 3.1.

REMARK 2. *Algorithm 3.1 appears to be a computationally-expensive procedure since it requires solving  $M$  stochastic inverse problems and, as noted in [5], approximating  $\pi_{\mathcal{D}}^{Q(\text{prior})}$  can be expensive. However, we note that we only need to compute this approximation once, as each  $I_Q$  in Step 4 of Algorithm 3.1 is computed using the same prior and map  $Q$  and, therefore, the same  $\pi_{\mathcal{D}}^{Q(\text{prior})}$ . In other words, the fact that the consistent Bayes method only requires approximating the push-forward of the prior implies that this information can be*

**Algorithm 3.1** Approximating the Expected Information Gain of an Experiment

1. Given a set of samples from the prior density:  $\lambda^{(i)}$ ,  $i = 1, \dots, N$ ;
2. Given a set of samples from the push forward density:  $q^{(j)}$ ,  $j = 1, \dots, M$ ;
3. Given the values of the posterior densities  $\pi_{\Lambda}^{\text{post}}(\lambda; q^{(j)})$  at :  $\lambda^{(i)}$ ,  $i = 1, \dots, N$ ;
4. For  $j = 1, \dots, M$  approximate  $I_Q(q^{(j)})$ :

$$I_Q(q^{(j)}) \approx \frac{1}{N} \sum_{i=1}^N \pi_{\Lambda}^{\text{post}}(\lambda^{(i)}; q^{(j)}) \log \left( \frac{\pi_{\Lambda}^{\text{post}}(\lambda^{(i)}; q^{(j)})}{\pi_{\Lambda}^{\text{prior}}(\lambda^{(i)})} \right)$$

5. Approximate  $E(I_Q)$ :

$$E(I_Q) \approx \frac{1}{M} \sum_{j=1}^M I_Q(q^{(j)})$$

used to approximate posteriors for different observed densities without requiring additional model evaluations. This significantly improves the computational efficiency of the consistent Bayesian approach in the context of OED.

**3.4. Defining the OED.** With these conceptual and computational details regarding data that extends beyond the range of the model addressed, we define the OED of a physics based model given prior information on the parameters, a space of potential experimental designs, and a generic description of the uncertainties for each QoI.

**DEFINITION 3.1 (OED).** *Let  $\mathcal{Q}$  represent the design space, i.e., the space of all possible experimental designs, and  $Q^z \in \mathcal{Q}$  be a specific design. Then the OED is the  $Q^z \in \mathcal{Q}$  that maximizes the expected information gain,*

$$Q^{\text{opt}} := \arg \max_{Q^z \in \mathcal{Q}} E(I_{Q^z}). \quad (3.7)$$

In general the design space  $\mathcal{Q}$  can either continuous or discrete. Our focus in this paper is on the utilization of the consistent Bayesian methodology within the OED framework, so we do not explore different approaches for solving the optimization problem given by Definition 3.1 and simply find the optimal design over a discrete set of candidate designs.

Consistent Bayesian inference is well suited to finding OED in continuous design spaces. Typically OED based upon statistical Bayesian methods uses Markov Chain Monte Carlo (MCMC) methods to characterize the posterior distribution. MCMC methods do not provide a functional form for the posterior but rather only provide samples from the posterior. Consequently gradient-free or stochastic gradient-based optimization methods must be used to find the optimal design. In contrast consistent Bayesian inference provides a functional form for the posterior which allows the use of more efficient gradient based optimizers. Exploring the use of more efficient continuous optimization procedures will be the subject of future work.

**4. Infeasible data.** The OED procedure proposed in this manuscript is based upon consistent Bayesian inference, which requires that the push-forward measure induced by the prior and the model *dominates* the observed measure, i.e., any event that we observe with non-zero probability will be predicted using the model and prior with non-zero probability (assumption A3). During the process of computing  $E(I_Q)$ , it is possible that we will violate this assumption. Specifically depending on the mean and variance of the observational



density, we can encounter  $\pi_{\mathcal{D}}^{\text{obs}} \in \mathcal{O}_{\mathcal{D}}$  such that  $\int_{\mathcal{D}} \pi_{\mathcal{D}}^{\text{obs}} d\mu < 1$ , i.e., support of  $\pi_{\mathcal{D}}^{\text{obs}}$  extends beyond the range of the map  $Q$ , see Figure 4.2 (middle). In this section we discuss the causes of infeasible data and options for avoiding infeasible data when estimating an optimal experimental design.

**4.1. Infeasible data and consistent Bayesian inference.** When inferring model parameters using consistent Bayesian inference the major cause for infeasible data is that the model being used to estimate the OED is inadequate. That is the deviation between the computational model and reality is large enough to prohibit the model from predicting all of the observational data. The deviation between the model prediction and the observational data is often referred to as model structure error and can often be a major source of uncertainty. This is an issue of most if not all inverse parameter estimation problems [10]. Recently there has been a number of attempts to quantify this error [12], however such approaches are beyond the scope of this paper. In the following we will assume that the model structure error does not prevent the model from predicting all the observational data.

**4.2. Infeasible data and optimal experimental design.** To estimate an approximate OED we must quantify the *expected* information gain of a given experimental design (see Section 3.3). The expectation is over all possible normal observation densities with mean  $q \in \mathcal{D}$  and variance  $\sigma$ , defined by the set (3.4). When the support of  $\mathcal{D}$  is bounded these densities may produce infeasible data. The effect of this violation increases as  $q$  approaches the boundary of  $\mathcal{D}$ .

To remedy this violation of (A3) we must modify the set of observational densities. In this paper we choose to normalize  $\pi_{\mathcal{D}}^{\text{obs}}$  over  $\mathcal{D}$ . We redefine the observed density space  $\mathcal{O}_{\mathcal{D}}$  so that (A3) holds for each density in the space,

$$\mathcal{O}_{\mathcal{D}} = \left\{ \frac{N(q, \sigma^2)}{C_q} : q \in \mathcal{D} \right\}, \quad (4.1)$$

where  $N(q, \sigma^2)$  is a Gaussian function with mean  $q$  and standard deviation  $\sigma$ , and  $C_q$  is the integral of  $N(q, \sigma^2)$  over  $\mathcal{D}$  with respect to the Lebesgue measure on  $\mathcal{D}$ ,

$$C_q = \int_{\mathcal{D}} N(q, \sigma^2) d\mu. \quad (4.2)$$

**4.3. A nonlinear model with infeasible data.** Consider the nonlinear model introduced in Section 3.2. Suppose the observed density on  $q_1$  is a truncated normal distribution with mean 0.3 and standard deviation of 0.04. In this one-dimensional data space, this observed density is absolutely continuous with respect to the push forward of the prior on  $q_1$ , see Figure 4.1 (left). Next, suppose the observed density on  $q_2$  is a truncated normal distribution with mean 0.982 and standard deviation of 0.01. Again, in this new one-dimensional data space, this observed density is absolutely continuous with respect to the push forward of the prior on  $q_2$ , see Figure 4.1 (right). Both of these observed densities are dominated by their corresponding push forward densities, i.e., the model can reach all of the observed data in each case.

However, consider the data space defined by *both*  $q_1$  and  $q_2$  and the corresponding push forward and observed densities on this space, see Figure 4.2. Clearly, this combined observed density is *not* absolutely continuous with respect to the push forward density on  $(q_1, q_2)$ , i.e., the support of  $\pi_{\mathcal{D}}^{\text{obs}}$  extends beyond the support of  $\pi_{\mathcal{D}}^{Q(\text{prior})}$ . Referring to Eq. 4.1, we normalize this observed density over  $\mathcal{D}$ , see Figure 4.2 (right). Now that the new observed density obeys the assumptions needed, we could solve the stochastic inverse problem as described in Section 2.



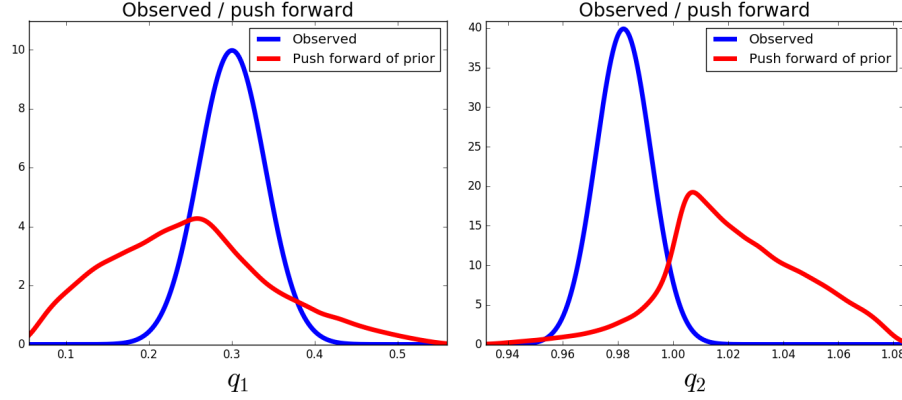


Fig. 4.1: (left) The push forward and observed densities on  $q_1$ , (right) and the push forward and observed densities on  $q_2$ . Notice the support of both of the observed densities is contained within the range of the model, i.e., the observed densities are absolutely continuous with respect to their corresponding push forward densities.

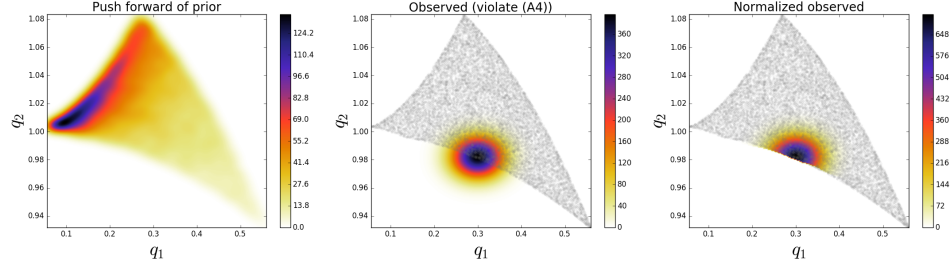


Fig. 4.2: (left) The push forward of the prior for the map  $Q : \Lambda \rightarrow (q_1, q_2)$  introduced in Section 3.2, (middle) the shaded area is the range of  $Q$ , we see the support of the observed density extends beyond the range of the map, (right) the normalized observed density that does not extend beyond the range of the map.

**4.4. Computational considerations.** Next, we address the computational issue of normalizing  $g(q, \sigma)$ , i.e.,  $\pi_{\mathcal{D}}^{\text{obs}}$ , over  $\mathcal{D}$ . From the plot of  $\pi_{\mathcal{D}}^{Q(\text{prior})}$  in Figure 4.2 (left) it is clear the data space may be a complex region. Normalizing  $\pi_{\mathcal{D}}^{\text{obs}}$ , as in Figure 4.2 (right), over  $\mathcal{D}$  would be computationally expensive. Fortunately, the consistent Bayesian approach provides a means to avoid this expense. Note that from Eq. 2.4 we have,

$$P_{\Lambda}^{\text{post}}(\Lambda) = P_{\Lambda}^{\text{prior}}(\Lambda) \frac{P_{\mathcal{D}}^{\text{obs}}(Q(\Lambda))}{P_{\mathcal{D}}^{Q(\text{prior})}(Q(\Lambda))}, \quad (4.3)$$

where  $P_{\Lambda}^{\text{prior}}(\Lambda) = P_{\mathcal{D}}^{Q(\text{prior})}(Q(\Lambda)) = 1$  which implies,

$$P_{\Lambda}^{\text{post}}(\Lambda) = P_{\mathcal{D}}^{\text{obs}}(Q(\Lambda)). \quad (4.4)$$

Therefore, normalizing  $\pi_{\mathcal{D}}^{\text{obs}}$  over  $\mathcal{D}$  is equivalent to solving the inverse problem and then normalizing  $\tilde{\pi}_{\Lambda}^{\text{post}}$  (where we use the tilde over  $\pi$  to indicate this function does not integrate

to 1 because we have violated (A3)) over  $\mathbf{\Lambda}$ . Although  $\mathbf{\Lambda}$  may not always be a generalized rectangle, (A1) implies we have a clear definition of  $\mathbf{\Lambda}$  and therefore can efficiently integrate  $\tilde{\pi}_{\mathbf{\Lambda}}^{\text{post}}$  over  $\mathbf{\Lambda}$  and then normalize  $\tilde{\pi}_{\mathbf{\Lambda}}^{\text{post}}$  by

$$\pi_{\mathbf{\Lambda}}^{\text{post}} = \frac{\tilde{\pi}_{\mathbf{\Lambda}}^{\text{post}}}{\int_{\mathbf{\Lambda}} \tilde{\pi}_{\mathbf{\Lambda}}^{\text{post}} d\mu_{\mathbf{\Lambda}}}. \quad (4.5)$$

**5. Numerical examples.** In this section we consider several models of physical systems. The first example, a linear model, is a bit abstract. We assume we have some model of a physical system where the QoI are linear functions of the parameters. Second, we consider a more concrete example, a stationary convection-diffusion model with a single uncertain parameter and QoI that are not necessarily linear with respect to the parameters. Third we consider a contaminant transport model with a two-dimensional parameter space and QoI

In each example, we have a parameter space  $\mathbf{\Lambda}$ , a set of possible QoI, and a specified number of QoI we can afford to gather during the experiment. This in turn defines a design space  $\mathcal{Q}$  and we let  $Q^z \in \mathcal{Q}$  represent a single experimental design and  $\mathcal{D}^z = Q^z(\mathbf{\Lambda})$  the corresponding data space. For each experimental design, we let  $\sigma^z$  represent the standard deviations defined by the uncertainties in each QoI that compose  $Q^z$ ,  $\mathcal{O}_{\mathcal{D}^z}$  represent the observed density space,  $\pi_{\mathcal{D}^z}^{\text{obs}} \in \mathcal{O}_{\mathcal{D}^z}$  represent an arbitrary observed density, and  $\pi_{\mathbf{\Lambda}}^{\text{post},z}$  represent the corresponding posterior density.

The first example has a discrete design space, and therefore we simply select the OED. The second and third examples have continuous design spaces, so we approximate the OED by selecting the OED from a large set of candidate designs. This approach was chosen because it is much more efficient to perform the forward propagation of uncertainty using random sampling only once and to compute all of the candidate measurements for each of these random samples. Alternatively, one could pursue a continuous optimization formulation, which would require a full forward propagation of uncertainty for each new design, but this was beyond the focus of this paper.

**5.1. A linear map.** Consider the following set of linear maps,

$$\begin{aligned} q_1(\lambda) &= 0.5\lambda_1 + 0.5\lambda_2 \\ q_2(\lambda) &= 2.5\lambda_1 + 0.5\lambda_2 \\ q_3(\lambda) &= -0.2\lambda_1 + 0.3\lambda_2, \end{aligned}$$

where we choose  $\lambda \in \Lambda = [0, 1]^2$ . Each of these QoI represents some kind of measurement of a physical system and  $\mathbf{\Lambda}$  represents two uncertain model input parameters. Suppose we can only afford to gather two of the three proposed QoI. This defines our design space  $\mathcal{Q}$  to be a discrete space with three possible maps  $\mathcal{Q} = \{Q^a, Q^b, Q^c\}$ , where

$$\begin{aligned} Q^a &= (q_1, q_2) \\ Q^b &= (q_1, q_3) \\ Q^c &= (q_2, q_3). \end{aligned}$$

Our goal is to choose the optimal map (OED) from  $\mathcal{Q}$ . To make this decision, we must specify the measurement uncertainties on each component. We let each of the measurement uncertainties be truncated normal distributions with standard deviation of 0.1. Therefore,

Design	$E(I_{Q(z)})$
$Q^a$	2.231
$Q^c$	2.126
$Q^b$	1.017

Table 5.1: The expected information gain for each of the three experimental designs.

the observed density spaces are defined as follows,

$$\begin{aligned}\mathcal{O}_{\mathcal{D}^a} &= \left\{ \frac{N(q, (\sigma^a)^2)}{C_q} : q \in \mathcal{D}^a \right\} \\ \mathcal{O}_{\mathcal{D}^b} &= \left\{ \frac{N(q, (\sigma^b)^2)}{C_q} : q \in \mathcal{D}^b \right\} \\ \mathcal{O}_{\mathcal{D}^c} &= \left\{ \frac{N(q, (\sigma^c)^2)}{C_q} : q \in \mathcal{D}^c \right\},\end{aligned}$$

where  $\sigma^a = \sigma^b = \sigma^c = (0.1, 0.1)$ . We choose a uniform prior and note that, because each component is linear, we can compute the push forwards of the prior exactly by,

$$\begin{aligned}\pi_{\mathcal{D}^{(a)}}^{Q^{(a)}(\text{prior})}(q) &= \frac{1}{\mu(\mathcal{D}^a)} \text{ for all } q \in \mathcal{D}^a \\ \pi_{\mathcal{D}^{(b)}}^{Q^{(b)}(\text{prior})}(q) &= \frac{1}{\mu(\mathcal{D}^b)} \text{ for all } q \in \mathcal{D}^b \\ \pi_{\mathcal{D}^{(c)}}^{Q^{(c)}(\text{prior})}(q) &= \frac{1}{\mu(\mathcal{D}^c)} \text{ for all } q \in \mathcal{D}^c.\end{aligned}$$

We generate 40,000 samples from the uniform prior, and for each  $Q^z \in \mathcal{Q}$ , this set of samples is used to approximate both  $I_{Q^z}$  and subsequently  $E(I_{Q^z})$  in Algorithm 3.1. To visualize the process outlined in Algorithm 3.1 we focus on the experimental design  $Q^a = (q_1, q_2)$ . Figure 5.1 (top) shows four of the 40,000  $\pi_{\mathcal{D}^{(a)}}^{\text{obs}} \in \mathcal{O}_{\mathcal{D}^a}$  used to approximate  $E(I_{Q^a})$  and their corresponding posterior densities  $\pi_{\Lambda}^{\text{post},(a)}$  (bottom). The information gain varies for each potential  $\pi_{\mathcal{D}^{(a)}}^{\text{obs}}$  and the expected information gain is the average over all of the computed information gains.

In Figure 5.2 we choose  $\pi_{\mathcal{D}^z}^{\text{obs}}$  to be centered at  $Q^z([0.5, 0.5])$  so we can observe the posteriors of the different experimental designs with limited boundary effects. In Table 5.1 we show the results of the approximated expected information gain for the three possible experimental designs. Design  $Q^a$  produces the maximum expected information gain and, therefore, is the OED.

## 5.2. Stationary convection-diffusion: uncertain source amplitude.

**5.2.1. Problem setup.** In this section we consider a stationary convection diffusion model on a square domain:

$$\begin{cases} -D\nabla^2 u + \nabla \cdot (\mathbf{v}u) = S, & x \in \Omega, \\ \nabla u \cdot \mathbf{n} = 0, & x \in \Gamma_N \subset \partial\Omega, \\ u = 0, & x \in \Gamma_D \subset \partial\Omega, \end{cases} \quad (5.1)$$

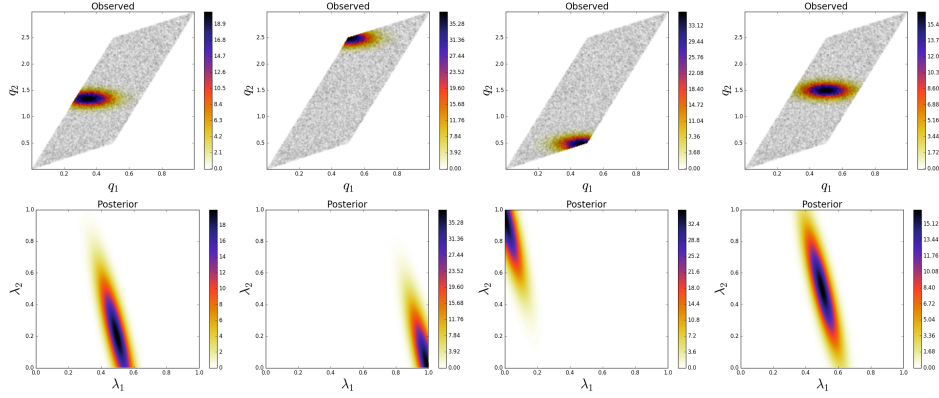


Fig. 5.1: A series of potential  $\pi_{\mathcal{D}(a)}^{\text{obs}}$  (top) and their corresponding  $\pi_{\Lambda}^{\text{post},(a)}$  (bottom). Information gain (from left to right) : 2.163, 2.658, 2.607, 1.929. For this map  $E(I_{Q^a}) = 2.231$ .

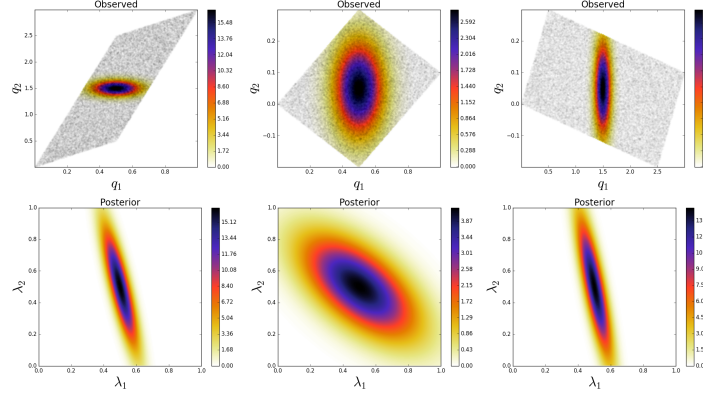


Fig. 5.2: (bottom) Posterior densities centered in  $\Lambda$  for each of the experimental designs (from left to right)  $Q^a, Q^b, Q^c$ . Corresponding information gain (from left to right) : 1.939, 0.565, 1.876. (top) The corresponding observed densities.

with

$$S(x) = A \exp\left(-\frac{\|x_{\text{src}} - x\|^2}{2h^2}\right)$$

where  $\Omega = [0, 1]^2$ ,  $u$  is the concentration field, the diffusion coefficient  $D = 0.01$ , the convection vector  $\mathbf{v} = [1, 1]$ , and  $S$  is a Gaussian source with the following parameters:  $x_{\text{src}}$  is the location,  $A$  is the amplitude,  $h$  is the width. We impose homogeneous Neumann boundary conditions on  $\Gamma_N$  (right and top boundaries) and homogeneous Dirichlet conditions on  $\Gamma_D$  (left and bottom boundaries). For this problem, we choose  $x_{\text{src}} = [0.5, 0.5]$ , and  $h = 0.05$ . We let  $A$  be uncertain within  $[50, 150]$ , thus the parameter space for this problem is  $\Lambda = [50, 150]$ . Hence, our goal is to gather some limited amount of data that provides the best information about the amplitude of the source, i.e., reduces our uncertainty in  $A$ .

To approximate solutions to the PDE in Eq. 5.1 given a source amplitude  $A$ , we use a finite element discretization with continuous piecewise bilinear basis functions defined on a

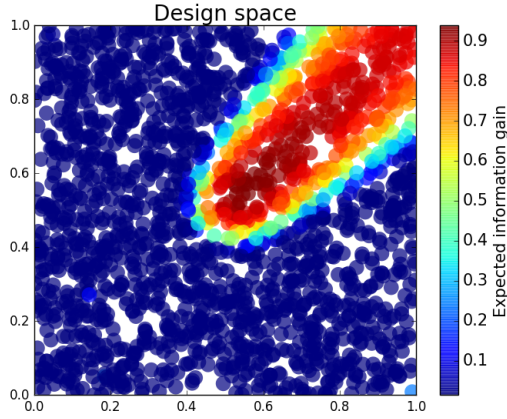


Fig. 5.3: (left) The expected information gain over the design space (which is  $\Omega$  in this example). Notice the higher values in the center of the domain and towards the top right (in the direction of the convection vector from the location of the source), this is consistent with our intuition. (right) The expected information gain for each of the top 6 experimental designs.

uniform ( $25 \times 25$ ) spatial grid.

**5.2.2. Results.** We assume that we have limited resources for gathering experimental data; specifically, we can only afford to place one sensor in the domain to gather a single concentration measurement. Our goal is to place this single sensor in  $\Omega$  to maximize the expected information gained about the amplitude of the source. We discretize  $\Omega$  using 2,000 uniform random points which produces a design space with 2,000 possible experimental designs. For this problem, we let the uncertainty in each QoI be described by a truncated Gaussian profile with a fixed standard deviation of 0.1. This produces observed density spaces,  $\mathcal{O}_{\mathcal{D}^z}$ , as described in Eq. 4.1.

We generate 5,000 uniform samples from the prior and simulate measurements of each QoI for each of these 5,000 samples. For each experimental design, we calculate  $E(I_{Q^z})$  using Algorithm 3.1 and plot  $E(I_{Q^z})$  as a function of the discretized design space in Figure 5.3 (left). Notice the expected information gain is greatest near the center of the domain (near the location of the source) and in the direction of the convection vector away from the source. This result matches intuition, as we expect data gathered in regions of the domain that exhibit sensitivity to the parameters to produce high expected information gains. In Figure 5.4 (right) we show the top 6 experimental designs and corresponding  $E(I_{Q^z})$ .

**5.3. Time-dependent diffusion: uncertain source location.** In this section, we compare results from a statistical Bayesian formulation of OED to the formulation described in this paper. Specifically, we consider the model in [8] where the author uses a Bayesian framework for OED to determine the optimal placement of a single sensor that maximizes the expected information about the location of a contaminant source.

**5.3.1. Problem setup.** We consider a contaminant transport model on a square domain:

$$\begin{cases} \frac{\partial u}{\partial t} = \nabla^2 u + S, & x \in \Omega, t > 0, \\ \nabla u \cdot \mathbf{n} = 0, & x \in \partial\Omega, t > 0, \\ u = 0, & x \in \Omega, t = 0. \end{cases} \quad (5.2)$$

with

$$S(x) = \begin{cases} \frac{s}{2\pi h^2} \exp\left(-\frac{\|x_{src} - x\|^2}{2h^2}\right), & \text{if } 0 \leq t < \tau, \\ 0, & \text{if } t \geq \tau, \end{cases}$$

where  $\Omega = [0, 1]^2$ ,  $u$  is the space-time concentration field, we impose homogeneous Neumann boundary conditions along with a zero initial condition, and  $S$  is a Gaussian source with the following parameters:  $x_{src}$  is the location,  $s$  is the intensity,  $h$  is the width, and  $\tau$  is the shutoff time. For this problem, we choose  $s = 2.0$ ,  $h = 0.05$ , and  $\tau = 0.3$ . We let  $x_{src}$  be uncertain within  $[0, 1]^2$ , thus the parameter space for this problem is  $\mathbf{\Lambda} = [0, 1]^2$ . Hence, our goal is to gather some limited amount of data that provides the best information about the location of the source, i.e., reduces our uncertainty in  $x_{src}$ . To approximate solutions to the PDE in Eq. 5.2 given a location of  $S$ , i.e., a given  $x_{src}$ , we use a finite element discretization with continuous piecewise bilinear basis functions defined on a uniform  $(25 \times 25)$  spatial grid and backward Euler time integration with a set size  $\Delta t = 0.004$  (100 time steps).

**5.3.2. Results.** We assume that we have limited resources for gathering experimental data; specifically, we can only afford to place one sensor in the domain and can only gather a single concentration measurement at time  $t = 0.24$ . Our goal is to place this single sensor in  $\Omega$  to maximize the expected information gained about the location of the contaminant source. For simplicity, we discretize  $\Omega$  using an  $11 \times 11$  regular grid of points which produces a design space with 121 possible experimental designs, i.e.,  $|\mathcal{Q}| = 121$ . We let the uncertainty in each QoI be described by a Gaussian profile with a standard deviation that is a function of the magnitude of the QoI, i.e.,

$$\sigma_i = 0.1 + 0.1|q_i| \text{ for } i = 1 \dots M, \quad (5.3)$$

where  $M$  is the dimension of the data space. This produces observed density spaces,  $\mathcal{O}_{\mathcal{D}^z}$ , that consist of truncated Gaussian functions with varying standard deviations,

$$\mathcal{O}_{\mathcal{D}^z} = \left\{ \frac{N(q, (\sigma(q))^2)}{C_q} : q \in \mathcal{D}^z \right\}. \quad (5.4)$$

We generate 5,000 uniform samples from the prior and simulate measurements of each QoI for each of these 5,000 samples. For each experimental design, we use this data to calculate  $E(I_{Q^z})$  using Algorithm 3.1 and plot  $E(I_{Q^z})$  as a function of the discretized design space in Figure 5.4 (left). Notice the expected information gain is greatest near the corners of the domain and smallest near the center, this is consistent with [8]. In Figure 5.4 (right) we show the top 6 experimental designs and corresponding  $E(I_{Q^z})$ . In Figure 5.5 we consider three different posteriors computed using data from the OED, i.e., data gathered by a sensor placed in the bottom left corner of the domain, where each posterior corresponds to a different possible location of the source. We see varying levels of information gain in these three scenarios, reiterating the point that we choose the OED based on the average of these information gains,  $E(I_Q)$ .

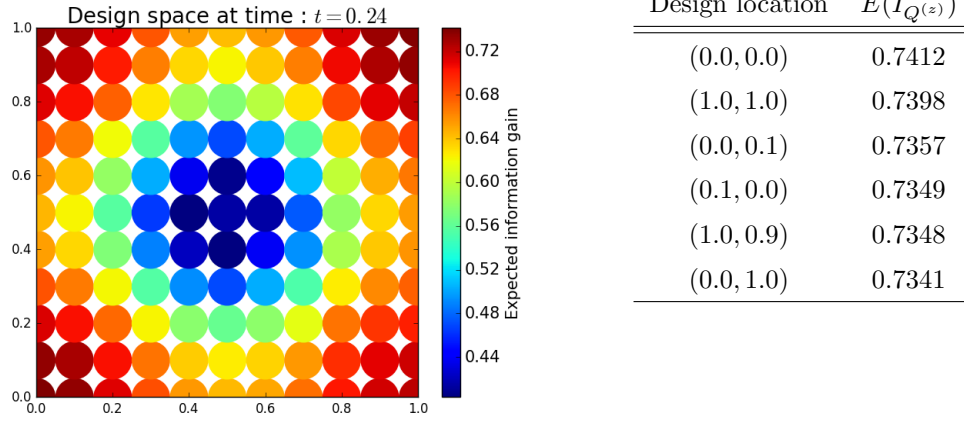


Fig. 5.4: (left) The expected information gain over the design space (which is  $\Omega$  in this example). Notice the higher values in the corners and the general trend are consistent with [8]. (right) The expected information gain for each of the top 6 experimental designs.

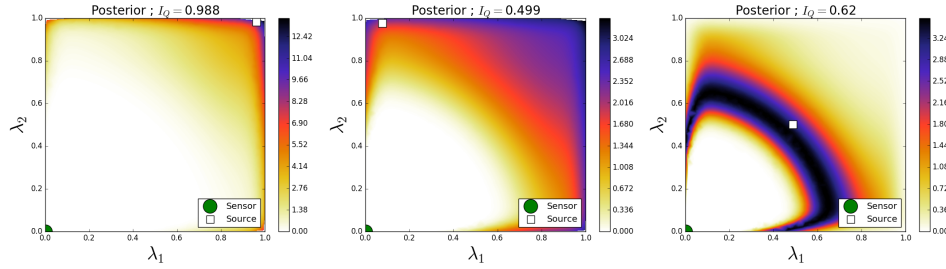


Fig. 5.5: Posteriors using the OED for three realizations of the location of the source. Notice the information gain changes substantially for each posterior, however, this experimental design produces the maximum average information gain,  $E(I_Q)$ , over all possible locations of the source.

REMARK 3. Although many of the results in this section seem to match our intuition about which measurement locations should produce high expected information gains, results from this method are not always consistent with intuition. In particular, we have found that our results can depend on our choice of the variance in the observed densities  $\sigma$ . If  $\sigma$  is chosen to be large relative to the range of a data space, then the posteriors produced as we average over  $\mathcal{O}_{\mathcal{D}}$  are all nearly the same and potentially produce unusually high information gains when the observed densities having substantial support over regions of the data space with very small probability (very small values of the push-forward of the prior). Another way to think of this is the push-forward densities have high entropy and because  $\sigma$  is large  $\pi_{\mathcal{D}}^{\text{obs}}$  is very close to uniform and this produces posterior densities with high information gains. If  $\sigma$  is chosen to be small relative to the range of the data space, we do not encounter this issue because we are integrating over  $\mathcal{D}$  with respect to the push-forward measure so most of our potential observed data lies in high probability regions of the data space.

**6. Conclusions.** In this manuscript, we developed an OED formulation based on the recently-developed *consistent* Bayesian approach for solving stochastic inverse problems. We discussed a characterization of the space of observed densities and a computationally-efficient approach for rescaling observed densities to satisfy the requirements of the consistent Bayesian approach. We also gave several intuitive numerical examples to demonstrate the OED procedure.

## REFERENCES

- [1] A. ALEXANDERIAN, N. PETRA, G. STADLER, AND O. GHATTAS, A-Optimal Design of Experiments for Infinite-Dimensional Bayesian Linear Inverse Problems with Regularized  $\ell_0$ -Sparsification, *SIAM J. Sci. Comput.*, 36 (2014), pp. A2122–A2148.
- [2] ———, A Fast and Scalable Method for A-Optimal Design of Experiments for Infinite-dimensional Bayesian Nonlinear Inverse Problems, *SIAM J. Sci. Comput.*, 38 (2016), pp. A243–A272.
- [3] F. BISETTI, D. KIM, O. KNIO, Q. LONG, AND R. TEMPONE, Optimal Bayesian Experimental Design for Priors of Compact Support with Application to Shock-Tube Experiments for Combustion Kinetics, *Int. J. Numerical Methods in Engineering*, 108 (2016), pp. 136–155.
- [4] J. BREIDT, T. BUTLER, AND D. ESTEP, A Computational Measure Theoretic Approach to Inverse Sensitivity Problems I: Basic Method and Analysis, *SIAM J. Numer. Analysis*, 49 (2012), pp. 1836–1859.
- [5] T. BUTLER, J. JAKEMAN, AND T. WILDEY, A Consistent Bayesian Formulation for Stochastic Inverse Problems Based on Push-Forward Measures. Submitted to *SIAM J. Sci. Comput.*, 2016.
- [6] T. BUTLER, M. PILOSOV, AND S. WALSH, Optimal Experimental Design: A Measure-Theoretic Approach. In Prep, 2016.
- [7] K. CHALONER AND I. VERDINELLI, Bayesian Experimental Design: A Review, *Institute of Mathematical Statistics*, 10 (1995), pp. 273–304.
- [8] X. HUAN, Numerical Approaches for Sequential Bayesian Optimal Experimental Design, (2015).
- [9] X. HUAN AND T. MARZOUK, Simulation-Based Optimal Bayesian Experimental Design for Nonlinear Nsystems, *Journal of Computational Physics*, 232 (2013), pp. 288–317.
- [10] M. C. KENNEDY AND A. O’HAGAN, Bayesian Calibration of Computer Models, *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63 (2001), pp. 425–464.
- [11] Q. LONG, M. SCAVINO, R. TEMPONE, AND S. WANG, Fast Estimation of Expected Information Gains for Bayesian Experimental Designs Based on Laplace Approximations, *Computer Methods in Applied Mechanics and Engineering*, 259 (2013), pp. 24 – 39.
- [12] K. SARGSYAN, H. N. NAJM, AND R. GHANEM, On the Statistical Calibration of Physical Models, *International Journal of Chemical Kinetics*, 47 (2015), pp. 246–276.
- [13] A. M. STUART, Inverse Problems: A Bayesian Perspective, *Acta Numerica*, 19 (2010), pp. 451–559.
- [14] A. TARANTOLA, *Inverse Problem Theory and Methods for Model Parameter Estimation*, SIAM, 2005.
- [15] T. VAN ERVEN AND P. HARREMOES, Renyi Divergence and Kullback-Leibler Divergence, *IEEE Transactions on Information Theory*, 60 (2014), pp. 3797–3820.



## Applications

Articles in this section discuss the application of computational techniques to simulate physical systems.

*Barone and Perego* use Albany to implement a unified enthalpy formulation for polythermal glaciers, which is an improvement over standard cold-ice models, and apply this model to predict temperature and porosity distributions of the Greenland ice sheet.

*Castillo, Mitchell, and Bond* apply data analytics methods to simulated microstructures of weld joints generated by the kinetic Monte Carlo code SPPARKS. Using two-point spatial cluster statistics, chord length distributions, windowing strategies, and Principle Component Analysis, they begin to quantify complex microstructures and significantly reduce the dimensions of the resulting data. The methods are described and demonstrated on a set of weld microstructures for which one process parameter, the weld speed, is varied. Performing analyses and dimensional reduction techniques presented here are an important first steps toward the ultimate goal of discovering material process-structure relationships.

*Decolvenaere and Mattsson* develop a test suite designed to be used in validation studies for codes that implement density functional theory. They focus on experimental data from transition metal elements and binaries, and specific binaries are selected with the goal of including a wide variety of physics, which may be captured with varying degrees of fidelity by different functionals, while keeping the number of computations to a minimum. They apply the suite to eight commonly-used functionals and demonstrate that no one functional captures all the physics present in transition metal elements and binaries.

*Fisher and Robinson* give an exact analytic solution to a forced Burger's equation using the method of characteristics. They numerically compute quantities of interest that are integrals over two-dimensional domains of space-time, including domains crossing the shock and different characteristic regions. These results are compared with numerical results from Drekar.

*Hong and Perego* use finite element based simulations to assess two phenomenological models for ion crowding, which are improvements on the Poisson Boltzmann model. Ion size effects are evaluated by comparing simulation results to results based on classical density functional theory, and the models are then applied to the problem of colloid mobility in a microchannel.

*Kalmbach and Chance* investigate a possible mechanism for phase precession in hippocampal place cells and whether an animal's age has an impact on place cell firing. They analyze rat electrophysiology data from a previously-published study using data clustering, curve fitting, and phase range analysis to find evidence that place cell firing in the CA1 region of the hippocampus is driven by two distinct components.

*Keller, Silling, and Du* perform molecular dynamics simulations that are in a time-dependent peridynamic framework that includes a thermal source term and allows coarse graining by constraining the average displacements of groups of atoms. They determine the conditions on thermal forcing magnitude, constraints, number of atoms per bin, and  $\delta$ -horizons for which the method is stable and converges to the correct solution, and perform numerical experiments to demonstrate the behavior of the method in different regimes.

*Meredith, Siefert, and Kramer* propose a method to compute the effective electrical conductivity tensor of a computational cell containing multiple materials, given the planar interfaces separating the different material regions. They demonstrate that this method is superior to volume averaging in the case where the electric field is tangent to all the material interfaces. They also propose a method for computing the element-centered current density from the edge-centered electric field using Ohm's law, rather than from the face-centered

magnetic field using Ampere's law. They demonstrate that the Ohm's law method results in second-order convergence, which is an improvement over the first-order convergence for achieved by the Ampere's law method.

*Pasetto and Littlewood* implement the semi-Lagrangian reproducing kernel particle method (RKPM) in Sierra / Solid Mechanics. As with Lagrangian RKPM, this meshfree method avoids the problems of mesh distortion and inversion often encountered with finite element approaches. Semi-Lagrangian RKPM goes further by rebuilding kernels, shape functions, and nodal neighbor lists at every time step. This approach is ideal for challenging computational mechanics problems involving large deformations, material damage, fracture, and material fragmentation. They describe their implementation, and present a simulation of a wave propagation in a bar as verification problem.

*Thompson, Matzen, and Armenta* test the effectiveness of pupillometry at measuring cognitive load by comparing eye tracker and electroencephalography measurements taken while subjects performed visual and cognitive tasks of varying difficulties.

*Wang, Quach, and Rothganger* propose an approach for file fragment classification based on sparse coding and recurrent neural networks with long short-term memory that does not require the use of hand-engineered features. They demonstrate this approach by classifying byte patches from common file types and in most cases achieve a prediction accuracy that is better than for existing methods.

J.B. Carleton

M.L. Parks

December 15, 2016

## IMPLEMENTATION OF ENTHALPY MODEL FOR POLYTHERMAL GLACIERS

ALESSANDRO BARONE\* AND MAURO PEREGO†

**Abstract.** Polythermal glaciers contain both cold ice and temperate ice where liquid water is present. Standard thermodynamic cold-ice models approximate the energy balance by a differential equation for the temperature variable. However, such models do not account for the portion of latent heat of liquid water within temperate ice, so they are not energy-conserving when temperate ice is present. We develop a computational model to describe temperature and liquid water fraction of glaciers, based on work by Schoff and Hewitt, 2016. They can be determined from the enthalpy of the ice sheets, so a unified enthalpy formulation will be described. The model accounts for the possibility of gravity-driven drainage allowing to predict more physical water content. The scheme is implemented in Albany and numerical results for the Greenland ice sheet are presented.

**1. Introduction.** In this work, we will focus on the computational modeling of temperature and liquid water content in polythermal glaciers. They contain both cold ice, below its pressure melting temperature, and temperate ice, which is made of a solid/liquid mixture at the melting temperature. Temperate regions are present mainly near the bed, where dissipation caused by viscous creep acting as a heat source allows regions of temperate ice to form.

Polythermal conditions are present in a huge variety of glaciers. For instance, the Greenland and Antarctic ice sheets feature cold ice in the bulk and a temperate layer near the bed. Models able to take into account such thermal structures are an important component of ice-sheet simulations because the rheology of ice depends strongly on temperature and water content. This motivates the development of thermomechanically coupled problem.

Thermodynamic models of ice flow have already been presented in literature [6]. Such models are called cold-ice models since they do not take into account the fully energy content of temperate ice, so they are not energy-conserving.

Some polythermal methods present in the literature split the ice domain into disjoint cold and temperate regions and model the temperature and the water content separately, applying Stefan matching conditions at the cold-temperate interface [4]. Since the implementation of such models is troublesome, we introduce a thermodynamic model using a unified enthalpy formulation able to take into account the presence of temperate ice. We will follow the approach proposed in [11].

The key ingredient of this theory is a constitutive law for water transport in the temperate ice. This mechanism might be modeled either by Darcy's law or by a diffusive flux. We will adopt the strategy that uses the diffusive flux. Moreover, in these regions, due to ice-water density difference, meltwater may also flow through the solid ice. We will model this mechanism by adding a gravity-driven drainage term. This will allow us to obtain more reliable results since disregarding the drainage mechanism leads to unreasonable levels of liquid water fraction.

The current report is organized as follows. In Section 2, we lay out the Stokes model and its first-order approximation for ice velocity. Then, we introduce the enthalpy model equations including boundary conditions. In Section 3, we describe the numerical implementation of the method in Albany/FELIX, a parallel, scalable and robust finite-element based solver for the first-order Stokes momentum balance equations for ice flow [9],[12]. Finally, in Section 4, we report on the results of the numerical simulations on Greenland ice sheet.

---

\*Department of Math&CS, Emory University, Atlanta, GA, alessandro.barone@emory.edu

†Sandia National Laboratories, Albuquerque, NM, mperego@sandia.gov

## 2. Mathematical model.

**2.1. Stokes model.** Glacier ice is generally modeled as an incompressible, heat-conducting, non-Newtonian fluid behaving as a Stokes flow under the action of gravity. The Reynolds number for typical ice-sheet flows are extremely low and inertial terms can be neglected due to the slow movement of the ice. The Stokes model is given by

$$\begin{cases} -\nabla \cdot \sigma = \rho \mathbf{g} \\ \nabla \cdot \mathbf{u} = 0, \end{cases} \quad (2.1)$$

where  $\mathbf{u}$  denotes the ice velocity,  $\sigma$  the Cauchy stress tensor,  $\rho$  the ice density, and  $\mathbf{g}$  the gravitational acceleration. For the sake of simplicity, we do not consider curved domains, so  $\mathbf{g}$  is assumed to be directed as the  $z$  axis, namely  $\mathbf{g} = (0, 0, -\|\mathbf{g}\|)$ .

The Cauchy stress tensor can be split into a deviatoric and isotropic part:

$$\sigma = \tau - pI = 2\mu\dot{\epsilon} - pI, \quad (2.2)$$

where  $\tau$  is the deviatoric stress tensor,  $p$  the ice pressure,  $I$  the identity tensor,  $\mu$  the ice effective viscosity and  $\dot{\epsilon}$  the strain-rate tensor defined as

$$\dot{\epsilon} = \frac{1}{2}(\nabla \mathbf{u} + \nabla \mathbf{u}^T). \quad (2.3)$$

The nonlinear rheology of ice sheets is commonly expressed by the Glen's law, relating the effective viscosity  $\mu$  and the effective strain rate  $\dot{\epsilon}_e$

$$\mu = \frac{1}{2}A^{-\frac{1}{n}}\dot{\epsilon}_e^{\left(\frac{1-n}{n}\right)} = \frac{1}{2}A^{-1}\tau_e^{1-n}, \quad (2.4)$$

where the effective stress  $\tau_e$  and  $\dot{\epsilon}_e$  are given by

$$\dot{\epsilon}_e = \sqrt{\frac{1}{2} \sum_{ij} \dot{\epsilon}_{ij}^2} \quad \text{and} \quad \tau_e = \sqrt{\frac{1}{2} \sum_{ij} \tau_{ij}^2}. \quad (2.5)$$

Moreover,  $n$  is the flow rate exponent and  $A$  follows an Arrhenius relationship

$$A = A(T) = \begin{cases} A_0 \exp\left(-\frac{Q_0}{RT}\right) & T < 263.15 \text{ K} \\ A_1 \exp\left(-\frac{Q_1}{RT}\right) & T \geq 263.15 \text{ K} \end{cases}, \quad (2.6)$$

where  $T$  is the temperature measured in Kelvin,  $R$  the universal gas constant,  $Q_0$  and  $Q_1$  are activation energies,  $A_0$  and  $A_1$  material constants.

*Stokes boundary conditions.* At the ice-atmosphere interface, a stress-free boundary condition is prescribed, i.e.,  $\sigma \cdot \mathbf{n} = \mathbf{0}$ , where  $\mathbf{n}$  denotes the unit vector normal to the ice sheet boundary and outward-pointing.

At the ice-bedrock interface, we consider either no-slip condition ( $\mathbf{u} = \mathbf{0}$ ) or sliding boundary conditions, i.e.,  $\mathbf{u} \cdot \mathbf{n} = 0$  and  $(\sigma \cdot \mathbf{n} + \beta \mathbf{u})_{||} = \mathbf{0}$ , where  $\beta$  is the basal sliding coefficient and the operator  $(\cdot)_{||}$  denotes the tangential projection onto the basal surface.

**2.2. First-order model.** Due to the high computational burden associated with the solution of the Stokes equations over 3D domains, such as Greenland or Antarctica, several approximate models have been introduced in the literature.

In this work, we consider the FO model, an first-order accurate approximation to the Stokes model (2.1) relying on the fact that the aspect ratio between the characteristic thickness and the characteristic horizontal extent of an ice sheet is very small. This simplification

leads to a three-dimensional elliptic problem in the variables  $u$  and  $v$ , the components of the velocity along the  $x$  and  $y$  directions, respectively. The FO model reads

$$\begin{cases} -\nabla \cdot (2\mu \dot{\epsilon}_1) = -\rho g \frac{\partial s}{\partial x} \\ -\nabla \cdot (2\mu \dot{\epsilon}_2) = -\rho g \frac{\partial s}{\partial y}, \end{cases} \quad (2.7)$$

where  $s \equiv s(x, y)$  denotes the upper surface boundary:  $\Gamma_s \equiv \{(x, y, z) \in \mathbb{R}^3 | z = s(x, y)\}$  and

$$\dot{\epsilon}_1 = \begin{bmatrix} 2\dot{\epsilon}_{xx} + \dot{\epsilon}_{yy} \\ \dot{\epsilon}_{xy} \\ \dot{\epsilon}_{xz} \end{bmatrix} \quad \text{and} \quad \dot{\epsilon}_2 = \begin{bmatrix} \dot{\epsilon}_{yx} \\ \dot{\epsilon}_{xx} + 2\dot{\epsilon}_{yy} \\ \dot{\epsilon}_{yz} \end{bmatrix}. \quad (2.8)$$

Further details about the derivation of this model and its numerical implementation can be found in [7],[12].

**2.3. Enthalpy model.** We assume that the FO model coupled with the conservation of mass accurately models the temperate ice deformation. This is not strictly true since, in case of temperate ice, the ice is actually a two-phase mixture of ice and pore water. This means that the porous ice matrix may be permeable, and the incompressibility assumption does not hold anymore due to the drainage of water from the ice. So, the expression in (2.1) should become (see [11] for more details)

$$\nabla \cdot \mathbf{u} = -\nabla \cdot \mathbf{j} - \frac{\rho_w - \rho}{\rho_w \rho} m, \quad (2.9)$$

where  $\mathbf{j} = \phi(\mathbf{u}_w - \mathbf{u})$  is the relative water flux,  $\phi$  the porosity (water content),  $\rho_w$  the water density, and  $m$  the internal melting rate (rate of mass conversion from ice to water).

Nevertheless, ignoring the ice-water density difference, the water flux is expected to be small compared to the ice flux so that the right hand side in (2.9) may be reasonably ignored.

The conservation of energy is given by

$$-k\Delta T + \rho c \mathbf{u} \cdot \nabla T = \tau_{ij} \dot{\epsilon}_{ij} - \rho_w L m, \quad (2.10)$$

where  $k$  is the ice thermal conductivity,  $c$  the ice heat capacity and  $L$  the latent heat. As done for the Stokes model, we focus on the steady version of the equation. The summation convention is used for repeated indices.

If the temperature  $T$  is below the pressure melting point  $T_m(p)$ , then  $m$  is zero. On the other hand, if the temperature reaches  $T_m(p)$ , the temperate ice is porous with a non-zero porosity  $\phi$ . Moreover, since in temperate ice the temperature is basically constant, (2.10) can be used to determine the melting rate  $m$ , which provides the source term to the moisture equation

$$\mathbf{u} \cdot \nabla \phi + \nabla \cdot \mathbf{j} = m. \quad (2.11)$$

(2.10) and (2.11) are usually combined into a single equation that holds for both cold and temperate ice. These equations can be unified by defining the enthalpy as

$$h = \rho c(T - T_0) + \rho_w L \phi, \quad (2.12)$$

where  $T_0$  is a fixed reference temperature (here we take  $T_0 = 265\text{ K}$  for not having the enthalpy negative almost everywhere). Temperature and porosity can be determined as follows

$$T = \begin{cases} T_0 + h/\rho c & \phi = \begin{cases} 0 & h < \rho c(T_m(p) - T_0) \\ [h - \rho c(T_m(p) - T_0)]/(\rho_w L) & h \geq \rho c(T_m(p) - T_0) \end{cases} \end{cases} \quad (2.13)$$

We point out that  $h < \rho c(T_m(p) - T_0)$  holds in case of cold ice, whereas  $h \geq \rho c(T_m(p) - T_0)$  denotes the temperate ice enthalpy.

Therefore, combining the energy and the moisture equation (2.10) and (2.11), we can easily introduce an enthalpy equation of an ice sheet

$$\mathbf{u} \cdot \nabla h + \nabla \cdot \mathbf{q} = \tau_{ij} \dot{\epsilon}_{ij}, \quad \mathbf{q} = \begin{cases} -k \nabla T & h < \rho c(T_m(p) - T_0) \\ -k \nabla T_m(p) + \rho_w L \mathbf{j} & h \geq \rho c(T_m(p) - T_0) \end{cases} \quad (2.14)$$

Enthalpy is transported by advection and conduction in the cold ice, and by advection and water transport in the temperate regions.

However, in order to close the thermal model and compute the temperature and the porosity, we need to introduce a model for the water flux  $\mathbf{j}$ . A standard approach relies on the assumption that the water transport is diffusive, so

$$\mathbf{j} = -\nu \nabla \phi, \quad (2.15)$$

for some coefficient  $\nu$ .

In this case, (2.14) is simply an advection-diffusion equation with variable diffusivity depending on the value of the enthalpy  $h$ .  $\nu$  is typically taken to be smaller than  $k/\rho c$ . This term may be interpreted as a regularization of the case  $\nu = 0$  implying that water is simply advected with the ice. This choice of  $\nu$  leads to a hyperbolic equation on the temperate regions that allows the enthalpy to be discontinuous at the cold-temperate interfaces. We want to prevent such discontinuities since they may introduce complications in the numerical methods.

Since this model may result in unreasonable levels of water in the ice, we aim at incorporating a term which takes into account the drainage of meltwater from the ice. Indeed, since liquid water is denser than ice, liquid water may flow through the ice porous matrix. Therefore, following [11], we add a gravity-driven drainage term in (2.15) yielding

$$\mathbf{j} = -\nu \nabla \phi + \frac{k_0 \phi^\alpha (\rho_w - \rho) \mathbf{g}}{\eta_w}, \quad (2.16)$$

where  $k_0$  is a permeability factor and  $\eta_w$  the viscosity of the water.

*Enthalpy boundary conditions.* Assuming the temperature field at the surface  $T_{surf}$  to be known a priori, a Dirichlet boundary condition is imposed at the ice-atmosphere interface

$$h_{surf} = \rho c(T_{surf} - T_0). \quad (2.17)$$

It is reasonable to assume the ice at surface to be cold, so  $\phi = 0$ .

At the ice-bedrock interface, heat flux  $\mathbf{q}$  is supplied to the basal surface by the geothermal flux  $\mathbf{G}$ . Moreover, frictional heating arises from sliding of the ice,  $F_b = \beta |\mathbf{u}|^2$ . In case of cold ice, the heat flux entering the base of the glacier is merely given by geothermal and frictional heating

$$\mathbf{q} \cdot \mathbf{n} = \mathbf{G} \cdot \mathbf{n} + F_b. \quad (2.18)$$

This leads to a Neumann boundary condition for the enthalpy  $h$  in (2.14)

$$-\frac{k}{\rho c} \nabla h \cdot \mathbf{n} = \mathbf{G} \cdot \mathbf{n} + F_b. \quad (2.19)$$

Throughout, we assume that  $\mathbf{G} \cdot \mathbf{n}$  is constant and equal to  $G$ .

On the other hand, in case of temperate ice, we need to take into account the fact that the ice is actually a mixture of solid/liquid water. Therefore, as explained in [3], two different boundary conditions are imposed depending on the basal water layer thickness  $\eta_b$ . In case of positive thickness, ice melts due to the heat flux entering from the bedrock. This process is taken into account by computing the basal melt rate  $-M_b$

$$-M_b = \frac{F_b + G}{\rho_w L (1 - \frac{\rho_w}{\rho} \phi)}, \quad (2.20)$$

where  $L$  is the latent heat of fusion.  $-M_b$  is basically the rate at which hydrological storage delivers liquid water to the base of the glacier. Of course,  $-M_b = 0$  in case of cold ice.

Since the basal melt rate calculation balances the energy flux, we apply a homogeneous Neumann boundary condition for  $h$  in (2.14)

$$-\frac{k}{\rho c} \nabla h \cdot \mathbf{n} = 0 \quad \text{if } \eta_b > 0. \quad (2.21)$$

On the other hand, if  $\eta_b = 0$ , this implies that  $T = T_m(p)$  and  $\phi = 0$ . Thus, we apply a Dirichlet boundary condition for  $h$  in (2.14)

$$h = \rho c (T_m(p) - T_0) \quad \text{if } \eta_b = 0. \quad (2.22)$$

In this work, for the sake of ease, we assume the thickness of the basal water layer  $\eta_b$  to be always positive, implying that we impose a homogeneous Neumann boundary condition in any temperate region. This is because the Dirichlet boundary condition (2.22) may be applied on a subset of DOFs that can change at each iteration of the nonlinear solver making its implementation cumbersome.

However, in a follow up work, we plan to improve the model by including (2.22) in the computational model.

**2.4. Computation of vertical velocity.** After computing the ice velocity  $\mathbf{u} = (u, v)$  from the FO model, the velocity along the  $z$ -direction  $w$  follows from mass conservation with zero vertical velocity at the base

$$w = w|_{z=b} - \int_b^z \left( \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right) d\tilde{z}, \quad (2.23)$$

where  $b$  is the bedrock elevation.

The basal melt rate as well as the drainage contribution affects the vertical velocity at the base. Therefore,  $w$  is computed as follows

$$w = -M_b + \frac{k_0 \phi^\alpha (\rho_w - \rho) \mathbf{g}}{\eta_w} \Big|_{z=b} - \int_b^z \left( \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right) d\tilde{z}. \quad (2.24)$$

**2.5. Hydrostatic approximation.** We use the hydrostatic pressure approximation

$$p(z) = \rho g(h - z) + p_{atm}, \quad (2.25)$$

where  $h$  is the surface elevation and  $p_{atm}$  is the atmospheric pressure. By recall the Clausius-Clapeyron relation

$$\frac{\partial T_m(p)}{\partial p} = -v, \quad (2.26)$$

where  $v$  is constant, the pressure melting temperature is given by

$$T_m(p) = 273.15 K - vp. \quad (2.27)$$

**3. Implementation.** The enthalpy formulation presented in the previous section has been implemented in Albany/FELIX, an unstructured grid, parallel, scalable and robust finite element solver for the first-order Stokes equation. The code is written in C++ and implemented in Albany (see [9],[12]), an open-source multiphysics analysis package based on the Trilinos multiphysics framework (<http://trilinos.sandia.gov>).

The PDEs defined in (2.7) and (2.14) and the associated boundary conditions are discretized using the classical Galerkin finite element method FEM. In particular, we resort to a P1 finite element discretization (see [7] for more details on the weak formulation of the first-order equations). The PDE equations are described by a set of evaluation kernels, whose evaluation is managed by the Phalanx package. Then, all that is required in Albany is to code the residual of the PDE equations.

Once the system of nonlinear algebraic equations for the ice velocities and the enthalpy is created following discretization by the FEM, the Trilinos package NOX is used to solve it with Newton's method. The Sacado package of Trilinos is in turn used to compute an analytic Jacobian matrix at each iteration of Newton's method with automatic differentiation. When the convergence of Newton's method is not monotonic, the solver uses a backtrack algorithm which halves the Newton increment until monotonicity is restored. The tolerance  $\text{tol} = 1 \times 10^{-2}$  is used. The inner linear Jacobian system is solved by Belos with block GMRES ( $\text{tol} = 1 \times 10^{-3}$ ) and ILU additive Schwartz preconditioner (available in IFPACK).

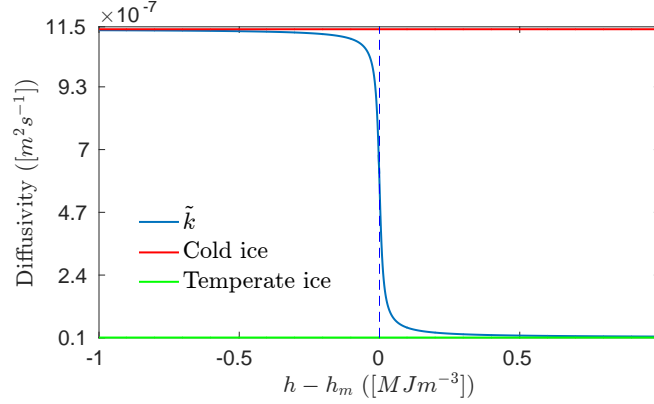
As pointed out in the previous section, the diffusivity parameter in (2.14) depends on the energy content of the glacier. In literature, it is reasonable to take the diffusivity  $\nu$  of temperate ice 100 times smaller than the diffusivity of cold ice  $k/\rho c$ . Therefore, especially at the cold-temperate interface, the discontinuity on the value of the diffusivity may undermine the convergence of the nonlinear solver. By running some preliminary simulations, we have noticed that the nonlinear solver does not converge. To overcome this issue, a smooth function is used to smooth out the discontinuity leading us to solve (2.14) with an unique diffusivity value  $\tilde{\nu}$  defined by

$$\tilde{\nu} = -\frac{k/\rho c - \nu}{\pi} \left( \tilde{\alpha} \arctan(h - h_m) \right) + \frac{k/\rho c - \nu}{2}, \quad (3.1)$$

where  $\tilde{\alpha}$  is a tuning parameter on the smoothness of the function and  $h_m = \rho c(T_m(p) - T_0)$  the pressure melting enthalpy. The colder the ice, e.g.,  $h \ll h_m$ , the closer  $\tilde{\nu}$  is to  $k/\rho c$ , whereas, if  $h \gg h_m$  (temperate ice),  $\tilde{\nu}$  tends to  $\nu$ . We might use some fixed value for the tuning parameter  $\tilde{\alpha}$ , e.g.,  $\tilde{\alpha} = 100$ , but even this choice does not improve the performances of the nonlinear solver.

Therefore, the strategy proposed is to treat  $\tilde{\alpha}$  as a continuation parameter in the so called homotopy method (Algorithm 1, see [2] for more details). This approach aims at



Fig. 3.1: Approximation of diffusivity discontinuity for  $\tilde{\alpha} = 100$ 


---

**Algorithm 1:** Homotopy continuation on tuning parameter  $\tilde{\alpha}$  in  $\tilde{\nu}$ 


---

 set  $\gamma = 0.1, h^0 = h_0$  and  $i = 0$ 
**while**  $\gamma \leq 1$  **or**  $i == \text{max \# iter}$  **do**

 Set  $\tilde{\alpha} = 10^{(-2+20\gamma)/9}$  and define  $\tilde{\nu}$  by (3.1)

 Set  $i = i + 1$ 

 Solve (2.14) with initial guess  $h^{i-1}$  using Newton's method to obtain  $h^i$ 

 Increase  $\gamma$  using a homotopy natural continuation method

**end while**


---

solving a sequence of problems corresponding to a sequence of tuning parameters  $\tilde{\alpha}_i$  with  $0 < \tilde{\alpha}_i < \tilde{\alpha}_{i+1}$ , until  $\tilde{\alpha}$  is sufficiently close to 100. We focus on a natural continuation procedure, where the final solution at one value of the continuation parameter is used as the initial guess for the subsequent nonlinear problem. The continuation algorithm relies on a homotopy parameter  $\gamma$  used to generate the sequence  $\{\tilde{\alpha}_i\}$ ,  $i = 1, 2, \dots$ , where  $\tilde{\alpha}_i = 10^{(-2+20\gamma)/9}$ . Starting with  $\gamma = 0.1$  (corresponding to  $\tilde{\alpha}_0 = 1$ ), the nonlinear solver converges from a trivial initial guess. The continuation algorithm will determine the next value of  $\gamma$  using an adaptive step size control, and will backtrack and attempt a smaller parameter step if the nonlinear solve at some step fails to converge (for more details, see [2] and [10]). The procedure stops either when  $\gamma = 1$  or a maximum number of steps is reached. Increasing the value of  $\tilde{\alpha}$ , the nonlinear solver fails to converge, therefore we set  $\tilde{\alpha} = 100$ . The correspondent approximation of the discontinuity is shown in Fig. 3.1. The homotopy continuation algorithm is implemented in the LOCA package [10].

Another issue we need to face is related to the discontinuity in the boundary conditions (2.19) and (2.21) at the basal surface for the enthalpy model. We impose either a homogeneous or nonhomogeneous Neumann boundary condition depending on the thermal state of the ice. Performing numerical simulations, we have noticed that this makes the nonlinear solver struggle to converge. Therefore, we smooth out the discontinuity using the same strategy explained above and (2.19) and (2.21) are replaced by an unique Neumann

$v$	$7.9 \times 10^{-8} \text{ K Pa}^{-1}$	$n$	3
$\rho$	$916 \text{ kg m}^{-3}$	$A_0$	$1.3 \times 10^7 \text{ Pa}^{-3} \text{ yr}^{-1}$
$\rho_w$	$1000 \text{ kg m}^{-3}$	$A_1$	$6.26 \times 10^{22} \text{ Pa}^{-3} \text{ yr}^{-1}$
$k$	$2.1 \text{ W m}^{-1} \text{ K}^{-1}$	$Q_0$	$6 \times 10^4 \text{ J mol}^{-1}$
$c$	$2009 \text{ J kg}^{-1} \text{ K}^{-1}$	$Q_1$	$1.39 \times 10^5 \text{ J mol}^{-1}$
$L$	$3.34 \times 10^5 \text{ J kg}^{-1}$	$R$	$8.314 \text{ J mol}^{-1} \text{ K}^{-1}$
$\nu$	$1.1 \times 10^{-8} \text{ m}^2 \text{ s}^{-1}$	$\alpha$	2
$g$	$9.8 \text{ m}^2 \text{ s}^{-1}$	$k_0$	$10^{-12} \text{ m}^2$
$p_{atm}$	$1.01 \times 10^6 \text{ Pa}$	$\eta_w$	$1.8 \times 10^{-3} \text{ Pa s}$
$G$	$0.05 \text{ W m}^{-2}$		

Table 3.1: Values of parameter used for numerical simulations.

boundary condition

$$\begin{aligned}
 -\frac{k}{\rho c} \nabla h \cdot \mathbf{n} &= a(\mathbf{G} \cdot \mathbf{n} + F_b), \\
 a &= -\frac{1}{\pi} \left( \tilde{\alpha} \arctan(h - h_m) \right) + \frac{1}{2},
 \end{aligned} \tag{3.2}$$

where  $a$  is tuned by Algorithm 1.

As pointed out in [12], Glen's law effective viscosity (2.4) is not defined when  $\mathbf{u}$  is a rigid movement or exactly 0. This may lead to poor convergence of the nonlinear solver since the initial guess for  $\mathbf{u}$  is often taken as uniform or 0. A regularization parameter  $\delta > 0$ ,  $\delta \ll 1$  is added to the sum of the strain rates in the effective strain rate term in (2.4), leading to

$$\mu_\delta = \frac{1}{2} A^{-\frac{1}{n}} (\dot{\epsilon}_e^2 + \delta)^{\left(\frac{1-n}{2n}\right)}, \text{ where } \lim_{\delta \rightarrow 0} \mu_\delta = \mu. \tag{3.3}$$

However, even using some small, fixed value for  $\delta$ , e.g.,  $\delta = 10^{-10}$ , Newton's method either converges very slowly or fails to converge. This issue is tackled using the same homotopy continuation procedure showed in Algorithm 1. The only difference is that  $\tilde{\alpha}$  is replaced by  $\delta$  and, instead of  $\tilde{\alpha} = 10^{(-2+20\gamma)/9}$ , we impose  $\delta = 10^{-10\gamma}$  (see [12] for more details).

We test the model and the numerical discretization using Greenland Ice Sheet (GIS) geometry. Ice sheet simulations with gravity-driven drainage may be sensitive to the value of the permeability factor  $k_0$ . For GIS, there is no common agreement on an appropriate value of  $k_0$  since it depends on grain size and impurity content which may vary within the glacier.  $k_0 = 10^{-12} \text{ m}^2$  seems to be a reasonable value (see [1]).

A realistic basal friction coefficient field  $\beta$  was calculated by solving a deterministic inversion problem (see [8] for more details). In all of the simulations we have performed, the ice velocity is given. Solutions with fully coupled ice velocity have not been tested yet.

The values of the parameters used for numerical simulations are shown in Table 3.1.

Finally, since equation (2.14) is strongly advection-dominated, the numerical discretization is stabilized by means of the strongly consistent SUPG (StreamLine Upwind Petrov-Galerkin) stabilization method [5].

**4. Results.** We present results from 3D GIS uniform quadrilateral meshes of different resolutions (see Fig. 4.1). The coarse mesh has 32 km resolution and the number of degrees of freedom  $\#DOF = 10824$ , whereas the fine mesh has 16 km resolution and  $\#DOF = 41322$ .

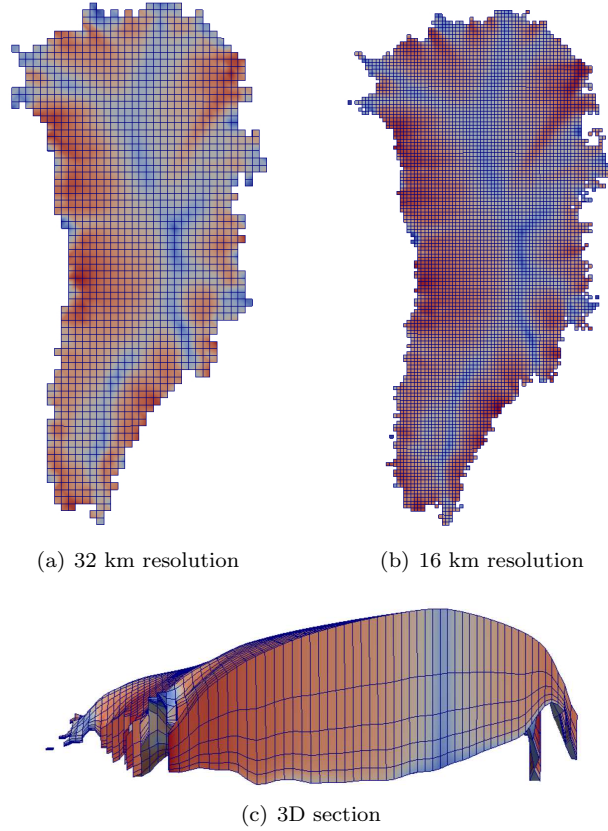


Fig. 4.1: Examples of GIS uniform quadrilateral meshes. Z axis scale is enlarged 100 times. Color contours represent the ice velocity

The 3D meshes were generated by extruding quadrilateral meshes into hexahedral meshes having layers denser close to the bedrock (see Fig. 4.1(c)). The number of layers considered in this work is equal to 5.

As mentioned earlier, ice velocity ( $[m\ yr^{-1}]$ ), basal friction coefficient  $\beta$  ( $[kPa\ yr\ m^{-1}]$ ) and the temperature at the surface ( $[K]$ ) are taken as given and they are displayed in Fig. 4.2.

Fig. 4.3 shows a comparison between the performances of homotopy algorithm and Newton's method with backtracking algorithm for the coarse mesh.

As for the fine mesh, the nonlinear solver does not converge for  $\tilde{\alpha} > 75$ , thus the continuation method is set up in such a way that  $\gamma = 1$  corresponds to  $\tilde{\alpha} = 75$ . Moreover, since in Albany an exporter for nodal fields has not been implemented yet, nodal fields were converted to cell fields, exported, and visualized in Paraview.

First, we show the results from the enthalpy formulation obtained with the 32 km resolution mesh. From Fig. 4.4, we point out that, since  $T_m(p) \approx 273K$ , the model predicts

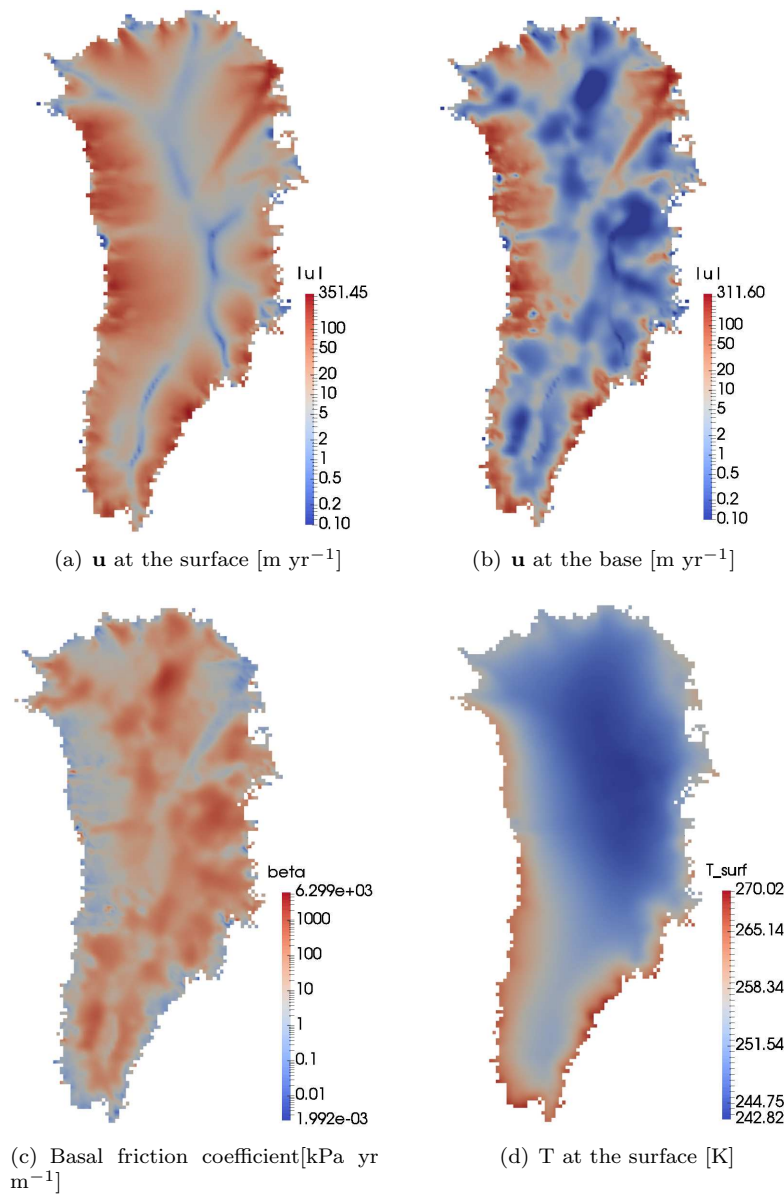


Fig. 4.2: Prescribed data

a majority of the basal area in the center to be cold, whereas the ice is temperate at the edge of the base. Furthermore, by comparing the temperature with the ice velocity at the base, temperate regions match those regions where the velocity magnitude is higher. This is reasonable since higher velocity magnitudes lead to higher basal friction heating that increases the enthalpy and so the temperature of the ice sheet.

Observations do not support a particular upper bound on the porosity  $\phi$  (water fraction), but values above 3% are rare. By looking at Fig. 4.4(b), the model predicts reasonable values

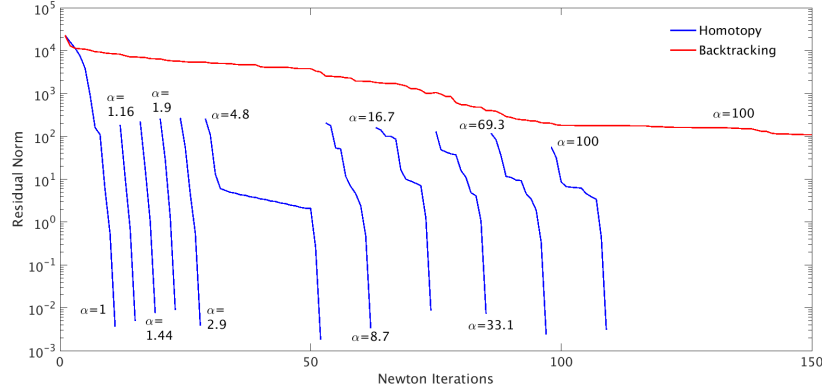


Fig. 4.3: Convergence of homotopy algorithm and Newton's method with backtracking

of  $\phi$ , and the water content seems to be consistent with the predicted temperature field and the velocity at the base. However, we obtain non trivial values of  $\phi$  in some areas of cold ice. This non-physical result may be related to the approximations introduced by (3.1) and (3.2). Further investigation is needed to improve the performances of both the homotopy algorithm and the nonlinear solver.

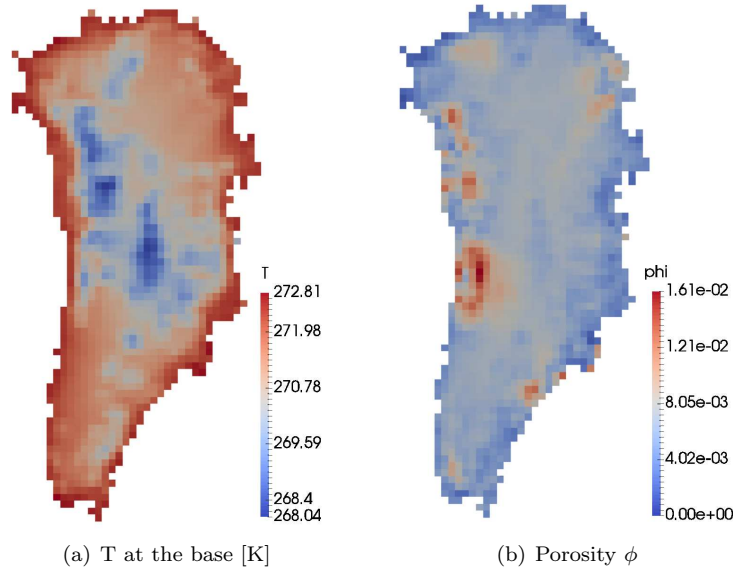


Fig. 4.4: Prediction of  $T$  and  $\phi$  at the basal surface for the coarse mesh

The claim that an enthalpy formulation would raise the liquid water fraction to abnormal levels without any drainage term is justified by Fig. 4.5. It shows the predicted water content when we disregard the gravity-driven drainage term. We notice that  $\phi$  is above 15%, providing a further confirmation of the fact that a reliable enthalpy model needs to

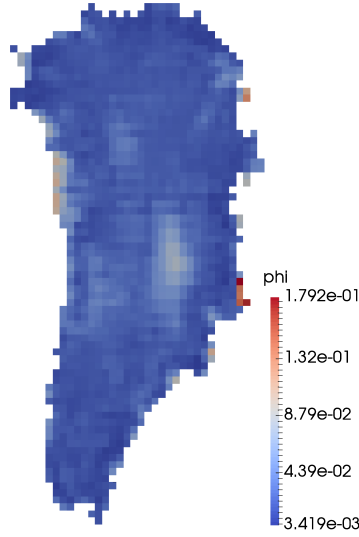


Fig. 4.5:  $\phi$  at the basal surface without drainage of water

take into account the drainage of meltwater from ice.

With the aim of obtaining more accurate results, we have tested the model on the fine mesh. Computed fields at the base are shown in Fig. 4.6.  $\phi$  features slightly larger but still reasonable values. However, the model still predicts non trivial water fraction in cold ice as noticed when using the coarser mesh. This needs to be fixed in the future in order to provide more reliable results. The predicted temperature field is similar to Fig. 4.4 and basal melt rate  $M_b$  is consistent with porosity and velocity fields.

For the sake of visualization, we plot the computed field  $h - h_m(p)$  in Fig. 4.6(b). By recalling the approximation of the diffusivity discontinuity shown in Fig. 3.1, the basal areas colored in blue (where  $h - h_m(p) < -0.5 \text{ MJ m}^{-3}$ ) are cold and those where  $h - h_m(p) > 0.5 \text{ MJ m}^{-3}$  are temperate, whereas we should be more careful in analyzing the thermal status of ice in the remaining areas.

Finally, Fig. 4.7 shows  $\phi$  in 2D vertical sections. Only the basal layer features non trivial amount of meltwater, whereas the ice is cold in the other layers. According to the literature [3], it is reasonable to expect small areas of temperate ice in the bulk of the ice sheet. This may be due to the assumption of positive thickness of the basal water layer in the treatment of the boundary condition at the base for temperate ice. Moreover, a refinement along the  $z$  direction using more layers may provide more physical results.

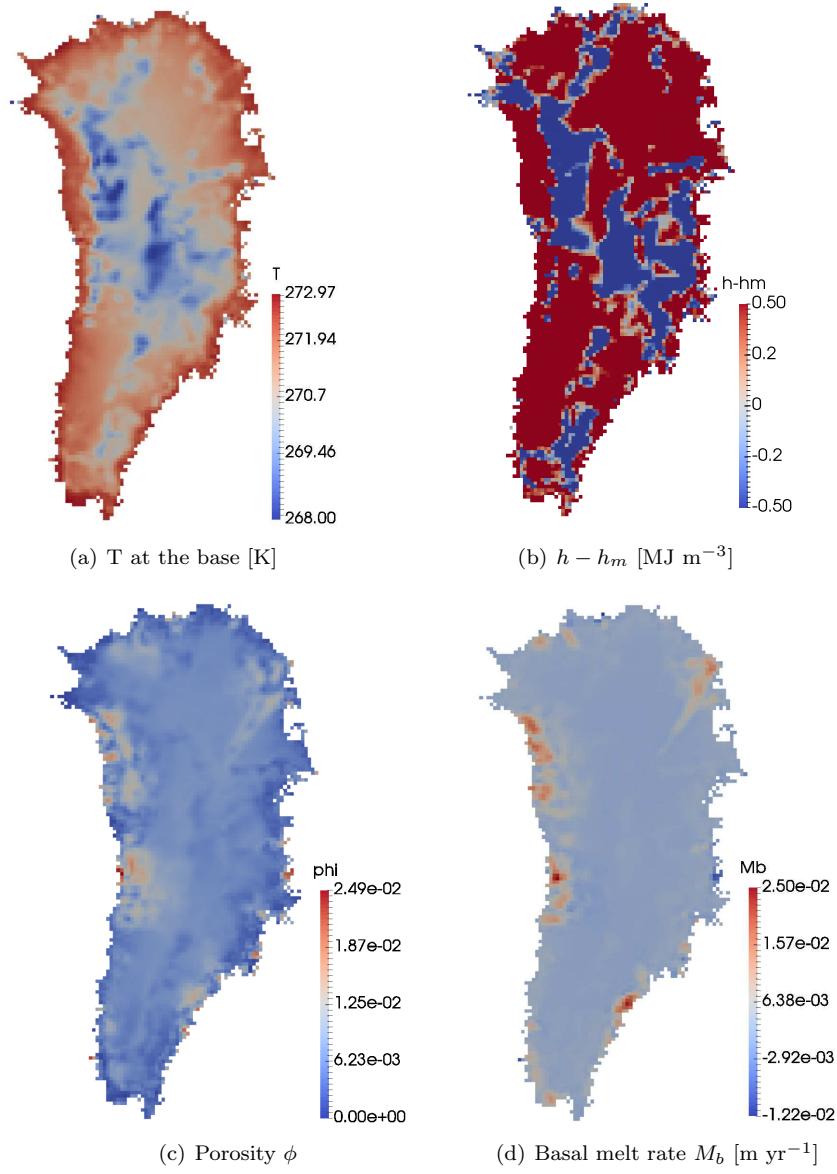


Fig. 4.6: Computed fields at the basal surface for the fine mesh

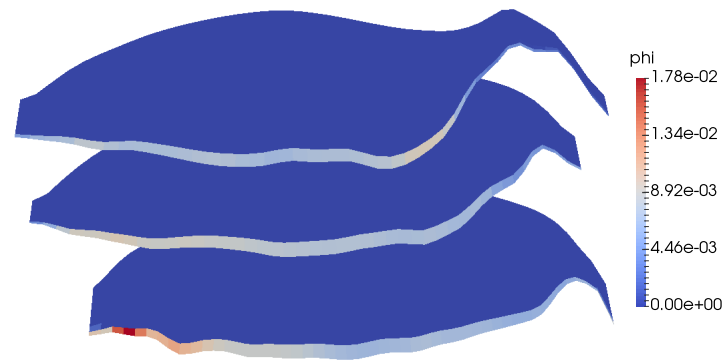


Fig. 4.7:  $\phi$  in three different 2D vertical sections



**5. Conclusions.** In this work, we have presented a method for modeling the temperature and the water content of polythermal ice sheets. An enthalpy formulation of such ice masses is more realistic than cold-ice methods since it is energy-conserving and allows to take into account the presence of liquid water in ice.

The possibility of meltwater raises modeling questions that are not present in cold-ice methods. These issues include choices about temperate ice rheology, boundary conditions at the ice-bedrock interface and drainage of meltwater from ice. A proper modeling of drainage mechanism plays a key role in determining the reliability of a thermodynamic model. We have taken care of this issue by using a gravity-driven drainage proposed by Hewitt and Schoof. The advantage of including such a term is that it allows for more physical liquid water content, whereas disregarding this term leads to abnormal levels of porosity.

A finite element discretization of the problem has been implemented in Albany. The huge discontinuity of the diffusivities of cold and temperate ice leads to poor performance of the nonlinear solver. This issue has been faced by introducing some approximations involving the use of a continuation method. It allows us to smooth the discontinuity and speed up the convergence of the nonlinear solver. Simplified boundary conditions at the base were considered to avoid implementation-related difficulties expected in Albany. The discontinuity affecting the boundary conditions at the basal surface was tackled by using the continuation method as well.

Numerical results on Greenland ice sheet have been shown using two meshes with different resolutions. We showed that the model provides reasonable temperature and water content fields in both cases. However, the implementation of the model needs to be improved. The approximation introduced by using the continuation algorithm may still have a negative impact on the accuracy of the results. Therefore, this issue as well as the assumption of a positive thickness of the basal meltwater layer needs to be tackled in order to provide more reliable results. For this purpose, we aim at improving the computational model by including (2.22).

Future work will also consist of comparing the behavior of this model with other enthalpy formulations present in literature and testing it on finer GIS meshes as well as Antarctica geometries. Future efforts will also be devoted to improving the efficiency of the linear and nonlinear solver. This may allow us to introduce discontinuities in the model without relying on the continuation method.

Finally, we aim at investigating the performance of the fully thermodynamic model, coupling the enthalpy formulation with the FO model.

**Acknowledgments.** We thank Ian J Hewitt for fruitful discussions.

## REFERENCES

- [1] A. ADOLPH AND M. ALBERT, Gas Diffusivity and Permeability Through the Firn Column at Summit, Greenland: Measurements and Comparison to Microstructural Properties, *The Cryosphere*, 8 (2014), pp. 319–328.
- [2] E. L. ALLGOWER AND K. GEORG, *Introduction to Numerical Continuation Methods*, vol. 45, SIAM Classics in Applied Mathematics, 2003.
- [3] A. ASCHWANDEN, E. BUELER, C. KHROULEV, AND H. BLATTER, An Enthalpy Formulation for Glaciers and Ice Sheets, *Journal of Glaciology*, 58 (2012), pp. 441–457.
- [4] R. GREVE, A Continuum-Mechanical Formulation for Shallow Polythermal Ice Sheets, *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 355 (1997), pp. 921–974.
- [5] T. J. HUGHES, A. BROOKS, ET AL., A Theoretical Framework for Petrov-Galerkin Methods with Discontinuous Weighting Functions: Application to the Streamline-Upwind Procedure, *Finite elements in fluids*, 4 (1982), p. 47.

- [6] A. PAYNE, P. HUYBRECHTS, A. ABE-OUCHI, R. CALOV, J. FASTOOK, R. GREVE, S. MARSHALL, I. MARIAT, C. RITZ, L. TARASOV, ET AL., Results From the EISMINT Model Intercomparison: the Effects of Thermomechanical Coupling, *Journal of Glaciology*, 46 (2000), pp. 227–238.
- [7] M. PEREGO, M. GUNZBURGER, AND J. BURKARDT, Parallel Finite-Element Implementation for Higher-Order Ice-Sheet Models, *Journal of Glaciology*, 58 (2012), pp. 76–88.
- [8] M. PEREGO, S. PRICE, AND G. STADLER, Optimal Initial Conditions for Coupling Ice Sheet Models to Earth System Models, *Journal of Geophysical Research: Earth Surface*, 119 (2014), pp. 1894–1917.
- [9] A. G. SALINGER, R. A. BARTLETT, A. M. BRADLEY, Q. CHEN, I. P. DEMESHKO, X. GAO, G. A. HANSEN, A. MOTA, R. P. MULLER, E. NIELSEN, J. T. OSTIEN, R. P. PAWLOWSKI, M. PEREGO, E. T. PHIPPS, W. SUN, AND I. K. TEZAUR, Albany: Using Component-Based Design to Develop a Flexible, Generic Multiphysics Analysis Code, *Int. J. Multiscale Com. to appear*, (2016).
- [10] A. G. SALINGER, N. M. BOU-RABEE, R. P. PAWLOWSKI, E. D. WILKES, E. A. BURROUGHS, R. B. LEHOUCQ, AND L. A. ROMERO, LOCA 1.0 Library of Continuation Algorithms: Theory and Implementation Manual, Sandia National Laboratories, Albuquerque, NM, Technical Report No. SAND2002-0396, (2002).
- [11] C. SCHOOF AND I. J. HEWITT, A Model for Polythermal Ice Incorporating Gravity-Driven Moisture Transport, *Journal of Fluid Mechanics*, 797 (2016), pp. 504–535.
- [12] I. K. TEZAUR, M. PEREGO, A. G. SALINGER, R. S. TUMINARO, AND S. F. PRICE, Albany/FELIX: a Parallel, Scalable and Robust, Finite Element, First-Order Stokes Approximation Ice Sheet Solver Built for Advanced Analysis, *Geoscientific Model Development*, 8 (2015), pp. 1197–1220.

## USING AND DEVELOPING NOVEL DATA ANALYTICS FOR SYNTHETIC MICROSTRUCTURES GENERATED BY SPPARKS

ANDREW R. CASTILLO\*, JOHN A. MITCHELL†, AND STEPHEN D. BOND‡

**Abstract.** The primary goal of this work is to develop and implement analytical methods for quantitative analysis of synthetic microstructures generated by the Sandia kinetic Monte Carlo code SPPARKS [6]. This report documents progress to that end as well as incremental work towards larger aspirations of making process-structure-property connections [4] for complex processes such as welding and additive manufacturing.

### 1. Introduction.

**1.1. Motivation.** The primary goal of this work is to develop and implement analytical methods for quantitative analysis of digitized microstructures such as those generated by the Sandia kinetic Monte Carlo code SPPARKS [6] and experimentally obtained microstructures. The tools will be demonstrated by analyzing grain morphologies generated by kinetic Monte Carlo (KMC) techniques [7, 5] that predict grain growth during welding and normal curvature-driven grain growth. These models use key process related parameters, such as spatial and temporal temperature profiles associated with scanning and/or rastering a heat source across material, producing qualitatively realistic looking microstructures; raster speed and aspects of melt pool geometries are also key process parameters. These models are used in this report to generate synthetic microstructures that can be further analyzed for purposes of creating process-structure linkages and maps.

**1.2. Approach.** This report is on the development and application of microstructure characterization methods based on data analytics. The methods are demonstrated on microstructures generated by a KMC model for simulating normal, curvature-driven grain growth during welding using the SPPARKS software package. The normal grain growth simulations [2] result in equiaxed grains while the welding simulations [5] give highly elongated grains with curvature that are heterogeneous in size and shape as a function of distance from the weld. Methods from data analytics [4], such as chord length distributions, two-point statistics and principal component analyses (PCA) [1] are applied to the synthetically generated microstructures.

A brief outline of the report is given. Section 2 gives a basic introduction and description of the methods used in this report, including a description of the Potts [2] model for equiaxed grains and the weld model [5] for spatially heterogeneous and non-equiaxed microstructures; this section continues on to describe chord length calculations and two-point statistics for use in PCA. Section 3 demonstrates the application of data analytic techniques to two sets of microstructures and discusses the results. The last two sections of the report summarize results, make conclusions about the data analytic methods demonstrated, and describe future plans.

**2. Description of methods.** Development of methods to quantify locally-heterogeneous microstructures are required to make process-structure-property linkages. These linkages facilitate process optimization via data analytics with the goal of producing desirable microstructures. This section introduces and describes methods and their use in achieving these goals. Specifically, microstructure generation, measure of cluster statistics, measure of

---

\*George W. Woodruff School of Mechanical Engineering, jcastillo7@gatech.edu

†Sandia National Laboratories, jamitch@sandia.gov

‡Sandia National Laboratories, sdbond@sandia.gov

windowed chord length distributions, and principal component analysis (PCA) as a means of dimensionality reduction are described and demonstrated.

**2.1. Modeling grain growth during welding.** The Potts grain growth model [2] has been modified to simulate grain growth during welding. Grain growth with non-uniform grain boundary mobility due to temperature gradients are incorporated in the model as described previously [3]; here, the model is extended to incorporate melting and solidification of the melt pool accompanying welding. This model, which simulates grain growth during welding, is an application in SPPARKS [6, 5], an open-source code developed for simulation of microstructural evolution using Monte Carlo methods.

**2.2. Spatial statistics.** To quantify microstructures and build surrogate models that capture process-structure-property linkages, a proper statistical representation of the weld microstructure must be developed. The implementation of two-point statistics used in this work is an extension of previous implementations [4]. The methods used are termed *average grain correlation* and the *cluster statistic*. Both of these methods are dependent upon the convolution of individual grains to form a spatial statistic of size  $S$ , where  $S$  is the number of lattice sites in the microstructure, and differ from each other only by the method of normalization. The aim is to quantify heterogeneity of the microstructure across various process parameters. The cluster statistic describes the probability that two points separated by an offset vector  $r$ , lie within the same grain. Formally, this is

$$f(r) = \sum_n^N \sum_s^S \frac{m_s^n m_{s+r}^n}{S_R}, \quad (2.1)$$

where  $r$  is the chosen vector,  $s$  is a lattice site index,  $S$  is total number of lattices sites,  $S_R$  is the number of sites that accomodate  $s$  and  $s+r$ ,  $n$  is a region of interest with a total of  $N$  regions. In this work, each region is a grain, and  $m_s^n$  is the probability density for region  $n$  of site  $s$ ; in practice,  $m_s^n \in \{0, 1\}$  is a binary representation of region  $n$  of a microstructure for all sites  $s \in 0, 1, 2, \dots, S$ ; this is illustrated in Figure 2.1. Note that  $f$  has the following properties

$$f(0) = \sum_n^N \sum_s^S \frac{m_s^n m_s^n}{S} = \frac{S}{S} = 1, \quad \lim_{|r| \rightarrow \infty} f(r) = 0. \quad (2.2)$$

In this work each grain is considered separately in the computation of statistics, thus  $N$  is equal to the number of grains. The mean autocorrelation is calculated by normalizing equation 2.1 by  $N$  instead of  $S_R$ . Casting the metric as a mean implies that  $f(0)$  is equal to the average grain size across the whole microstructure. In its current form, this formulation is computationally expensive. To reduce computational cost, convolutions are performed in Fourier space; this is shown below.

$$F(m_s^n) = M_k^n = \sum_s^S m_s^n e^{\frac{2\pi i s k}{S}} \quad (2.3)$$

$$f(r) = \frac{\sum_n^N F^{-1}(M_k^{n*} M_k^n)}{S_R} \quad (2.4)$$

Computational cost can be further reduced by identifying a bounding box for each grain and performing convolutions over the reduced volume; treating the microstructure

and padding the boundary of the box must be carefully done so that spurious boundary effects are not introduced. The padded region along each coordinate axis is nominally twice the size of the original grain bounding box; the final padded region size along an axis is even or odd depending on whether the original full image contained an even or odd number of sites. This process is shown in Figure 2.2.

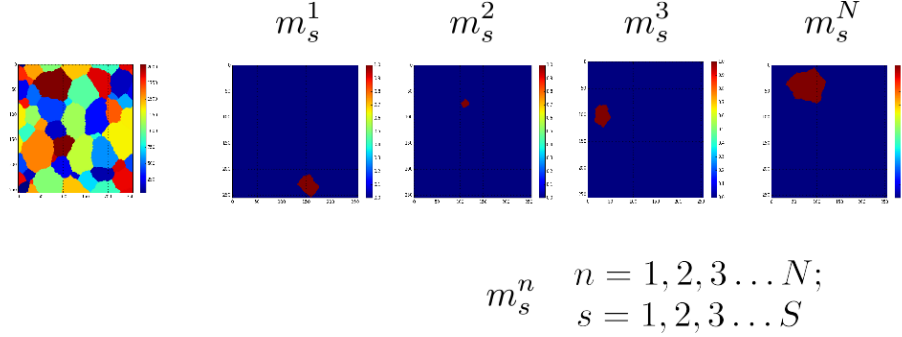


Fig. 2.1: Microstructure Discretization Scheme

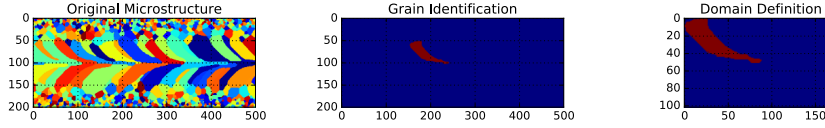


Fig. 2.2: Shrink Calculation Domain

It is important to note that after the reduction to this size, the resulting statistic must be padded, or reduced to the resulting size,  $S$ , of the final statistics. This method presents a global statistic that captures the heterogeneity of the grains throughout the microstructure. However the spatial dependence of grains is lost in this approach, making additional metrics necessary if an accurate microstructure synthesis from the given statistics are needed.

**2.3. Chord length distributions.** The implementation of chord length distributions forms an  $n$ -point statistic used to quantify weld microstructures. The challenge is to characterize regions of spatially homogeneous grains within the microstructure. The weld microstructures have periodic boundary conditions along the weld axis ( $y$ -axis) and non-periodic boundary conditions through the thickness ( $z$ -axis). Based on these conditions, the microstructures are partitioned into spatially homogeneous volumes bounded by the green regions shown in Figure 2.3.

These partitions are defined by “windows” on the  $x - z$  plane denoted by  $(dx, dz)$ . In this example, a window size of  $(15, 35)$  was chosen, which partitions the microstructure into 13 regions parallel to the weld axis. Using this method the chord length distribution for  $L_x$ ,  $L_y$  and  $L_z$  can be found for a given a region. The chord length is the length of a line segment within a grain along a given direction  $x$ ,  $y$  or  $z$ , and example chords are shown in Figure 2.4. As an example, the distribution for the  $y$  direction for a windowed region  $R$  is

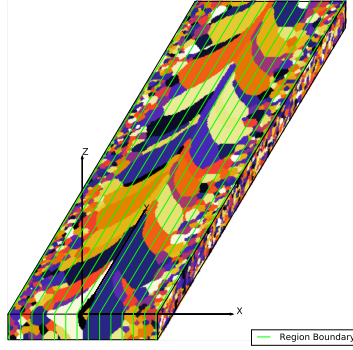


Fig. 2.3: Weld microstructure windowing for chord length distributions. The weld microstructure is partitioned into  $R$  regions to perform chord length calculations. The green lines denote the boundaries of a region.

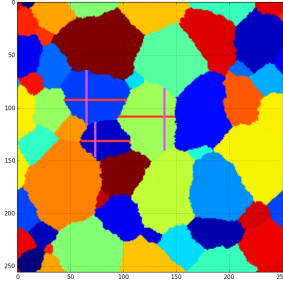


Fig. 2.4: Visualization of chordlengths for  $x$  direction (red) and  $y$  direction (purple) for an example microstructure

$$L_y(d) = \sum_{s=1}^{S_R} \frac{c_s^d}{S_R}; \quad d \in 1, 2 \dots D_y; \quad c_s^d \in \{0, 1\}, \quad (2.5)$$

where  $S_R$  is the total size of a windowed region,  $c$  is 1 if position  $s$  in the region of interest is inside a chordlength denoted by  $d$ , and otherwise is 0.  $D_y$  is given by the total number of sites along dimension  $y$  in this work. The chord length distribution implemented is the weighted chord length distribution, which gives the probability for a point in the microstructure being within a certain chord length along an axis.

**2.4. Principal component analysis (PCA).** PCA is a form of dimensionality reduction. Using PCA, each point in a large dimensional data set is approximated by its projection onto a much lower dimensional orthonormal basis. Surrogate models for process-structure-property linkages are constructed using a significantly reduced basis set derived from the PCA. PCA begins with a data matrix  $X$ , which is mean centered, and which has shape  $m \times n$ , where  $m$  is the number of microstructures (or observations) and  $n$  is the total size of the statistics applied to the microstructure. In this context, if only one

spatial statistic is considered, then  $n$  is the number of sites in the synthetically generated microstructure.

It is assumed that  $n$  is much larger than  $m$ . Using the reduced singular value decomposition, the data matrix is multiplicatively decomposed

$$X = USV^T = \sum_{i=1}^m S_i \vec{U}_i \vec{V}_i^t, \quad (2.6)$$

where  $S$  is a diagonal matrix whose diagonal values  $S_1 \geq S_2 \geq S_3 \geq \dots, S_m \geq 0$  are singular scalar values,  $\vec{U}_i$  are columns of matrix  $U$  which has shape  $m \times m$ , and  $\vec{V}_i^t$  are column vectors of  $V^t$  which has shape  $m \times n$ . Note that  $X$  can be approximated by

$$X_K = \sum_{i=1}^K S_i \vec{U}_i \vec{V}_i^t, \quad (2.7)$$

where  $K \ll m$ . This is the best rank  $K$  approximation as measured by the Frobenius norm  $\|\cdot\|_F$

$$\|X - X_K\|_F^2 = S_{K+1}^2 + \dots + S_m^2, \quad \|X\|_F^2 = S_1^2 + S_2^2 + \dots + S_m^2. \quad (2.8)$$

A ratio of the two norms provides a measure of how well  $X_K$  approximates  $X$ ; later in this report, scree plots are computed using the following formula, where the quality measure  $e$  varies between 0 and 1.

$$e = 1 - \frac{\|X - X_K\|_F^2}{\|X\|_F^2} \quad (2.9)$$

Once the decomposition above is computed, a row of  $X$ ,  $\vec{X}_j^t$ , can be projected onto the span of the first  $K$  right singular vectors of  $V$  using dot product  $\langle \cdot, \cdot \rangle$

$$\vec{X}_j^t \approx \langle \vec{X}_j^t, \vec{V}_1^t \rangle \vec{V}_1^t + \dots + \langle \vec{X}_j^t, \vec{V}_K^t \rangle \vec{V}_K^t. \quad (2.10)$$

The coefficients  $\langle \vec{X}_j^t, \vec{V}_1^t \rangle$  are known as the principal component scores.

**3. Demonstration calculations.** Organization of this section is split between cluster statistics on equiaxed grains, weld microstructure generation, application of cluster statistics to weld microstructures, PCA on cluster statistics, application of windowed chord length distributions to weld microstructures, and PCA on chord length distributions. The purpose of this section is two fold: to introduce these measures for the weld microstructure and display the information generated from the measures, and to perform PCA and demonstrate the viability of these statistics to build a process-structure-property linkage.

**3.1. Cluster statistics-equiaxed grains.** Cluster statistics capture the probability of the head and tail of a vector being within the same grain of a microstructure. Interpretation of this statistic is that the vector tail (center of statistic) is located inside a grain  $|r| = 0$  and hence the statistic value is 1; the vector head can take on any point in the microstructure and hence the statistic value(s) limit is 0 with increasing  $|r|$  since the grains are finite in size.

Equiaxed microstructures were simulated using the Potts grain growth model, and results of demonstration calculations are shown in Figure 3.1; this figure depicts two microstructures representing microstructural evolution at two different time steps from a grain

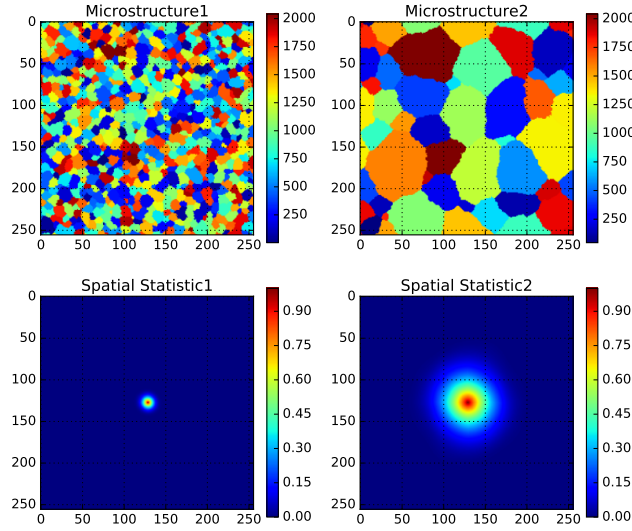


Fig. 3.1: Cluster statistics is applied to equiaxed microstructures. The resulting statistics exhibit a radial distribution which increases in size respective to the increase in average grain size

growth simulation. Visually, the microstructures appear to be equiaxed; using the cluster statistic defined in Section 2.2, this fact is quantified and captured by the near radial distribution of the statistical field. Comparing the statistic between the two microstructures, the cluster statistic captures increase in grain size as an increase in the diameter of the radial probability distribution; for the microstructure with larger grains, this reflects the increased probability for larger values of  $|r|$  being within the same grain.

**3.2. Cluster statistics-weld model.** The cluster statistics were applied to weld microstructures in an effort to capture spatial heterogeneity. A 3D weld microstructure with lattice dimensions of (200, 500, 35) was generated using the SPPARKS model described in Section 2.1. Twenty microstructures, with simulated weld velocities of 5, 6.25, 7.5, 9, and 12 ls/MCS, were used for the study. Units of the velocity are in lattice site per Monte Carlo Step (MCS). Representative microstructures are shown in Figures 3.2(a) through 3.2(e).

Figure 3.3 shows 2D spatial statistics for xy planar cuts at different levels along the z-axis of a weld microstructure; The weld axis and direction of applied weld (left to right) is apparent by the line of symmetry formed by grains.

Figure 3.4 shows zoomed in spatial statistics for Figure 3.3. There are significant differences between the statistics. Statistics 1 and 3 have “wings” that are more apparent than in statistic 2. The apparent wings are composed of elongated grains along the weld axis; elongated grains are more regularly distributed in cuts 1 and 3 than in 2; the central and circular aspect of the statistic arises from equiaxed grains at distances further from the weld axis in the heat affected zone. Differences in grain size and shape across these cuts are due to differences in temperature gradients. This spatial heterogeneity is expected and captured with the 2D cluster statistics. Further insight may be gleaned by extracting line statistics



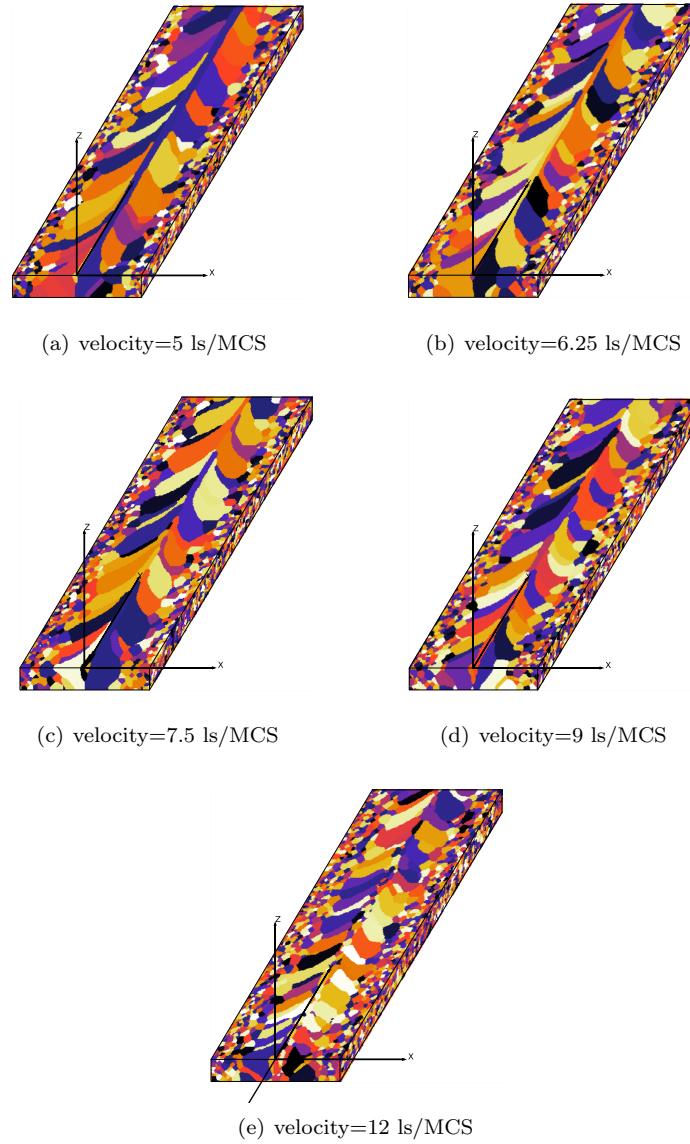


Fig. 3.2: Representative weld microstructures for the chosen process parameters are shown. Twenty microstructures for each weld velocity were simulated.

along particular directions (Figure 3.5).

The line statistics show a rudimentary analysis of the spatial statistics. An effective orientation for the wings is found and shown by the blue and green lines. This orientation is thought to be representative of the direction of average elongation during the weld process. Although they appear symmetric, they are not perfectly so. The average orientations given by the blue and green lines are 40.7 and 38.3 degrees from the weld axis respectively; this discrepancy is due to statistical variance and the number of elongated grains sampled. The

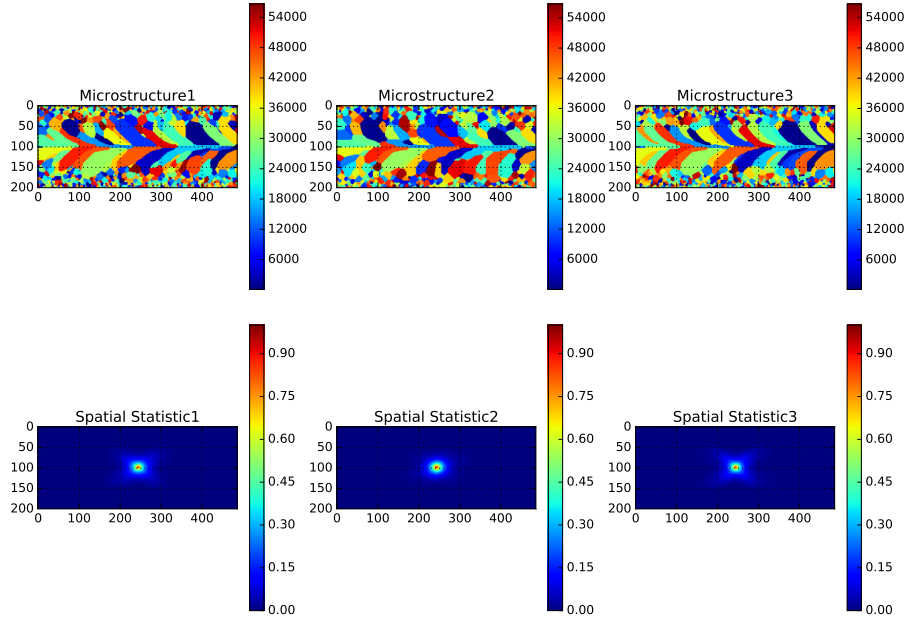


Fig. 3.3: Spatial statistics for planar cuts at  $z$  elevations through thickness of weld; cuts 1 and 2 are the closest and furthest from the top surface respectively; cut 3 is evenly spaced between cuts 1 and 2; images for cuts 1 through 3 are arranged from left to right respectively.

probability distributions are similar but diverge noticeably from the line statistic along the weld axis (orange line) at radii larger than 20 ls.

To more fully take advantage of the spatial statistics and demonstrate their utility, they are used to quantify differences in weld microstructures by varying a single process parameter while keeping all other parameters constant. In this case weld velocity was varied from 5 to 12 ls/MCS over 100 weld-simulations; microstructures at each velocity study are shown in Figures 3.2(a) through 3.2(e). Each starting microstructure was unique to ensure each simulation was statistically equivalent but had a distinct starting microstructure. Each run produced a 3 dimensional microstructure with dimensions of (200, 500, 35) for a total of 3,500,000 lattice sites. All other process parameters were kept constant. PCA, as defined in Section 2.4, was used to represent microstructures with reduced dimensionality. The results of PCA using cluster statistics are shown in Figures 3.6 and 3.7.

Visual inspection of Figures 3.6 and 3.7 show that PCA of microstructure statistics cluster by weld velocity. The principal scores for each microstructure: PC1, PC2, and PC3, are calculated from the dot product in Equation 2.10 using the singular vectors corresponding to the three largest singular values computed using the whole dataset. There is a clear trend in PC1 with weld velocity; as the weld process velocity decreases, PC1 increases. This distinct trend presents opportunities to build process-structure linkages. The *scree* plot shown in Figure 3.8 is found using Equation 2.9. The *scree* plot describes the total variance captured as a function of number of principle components. Viewing the *scree* plot, 89.9 percent of the variance is captured with 1 principal component and 96 percent with three. Further inspection of Figures 3.6 and 3.7 shows that the largest

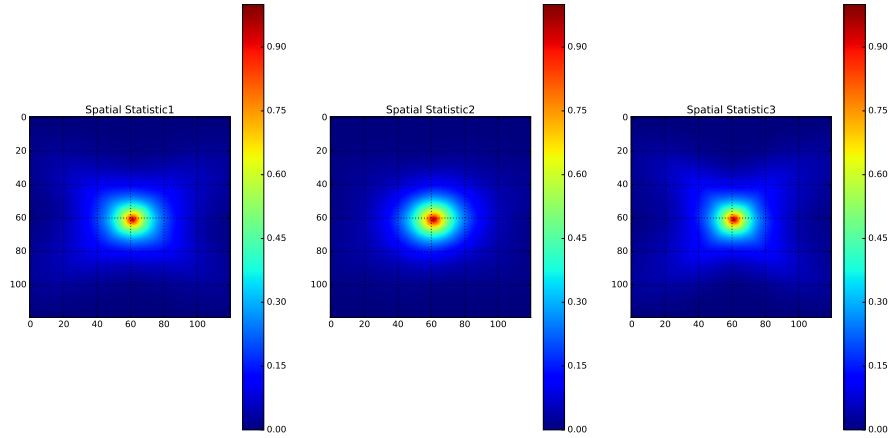


Fig. 3.4: Zoomed in spatial statistics from Figure 3.3

intravariance is exhibited by weld microstructures produced from a velocity of 5 lattice sites per Monte Carlo Step (ls/MCS). Conversely, the smallest intravariance appears to be exhibited by microstructures produced from a velocity of 12 ls/MCS. There appears to be a large spread over PC2 for microstructures ran at a velocity of 5 ls/MCS. This large spread begets the question of whether the dimensionality reduction may be skewed or dominated by the large intravariance exhibited by the 5 ls/MCS microstructures. This may play a role in the future when developing an inverse predictive model.

It has been shown that the application of the spatial statistics in conjunction with PCA has successfully reduced high dimensional data sets to much lower dimensionality. More importantly, with further development of these characterization tools, a quantitative relationship may be found which may enable quantifying the process-structure linkages. Developing predictive models relating process variables to microstructure based on PCA is promising. These models infer unknown process parameters given a microstructure for which a process-structure linkage has been built for similar microstructures. This is accomplished by projecting mean centered statistics of the given unknown microstructure into the existing PC space as shown by Equation 2.10. The PC scores from the unknown microstructure are then used to infer the given velocity, in this case by inverting the process-structure model. To perform the inversion, surrogate models for the PC scores as a function of the independent variable (such as velocity) are developed. The simplest of these models are polynomial fits for the mapping of PC components to the process parameters. The use of this approach assumes that any processes applied to the microstructure of interest are spanned by the family of microstructures used for building the initial linkage. In general, surrogate models can be produced by tracking the evolution of microstructures through sequential Monte Carlo Steps. By concentrating on resulting microstructures at various steps in time, process paths can be built in PC space. This type of model requires non-intersecting distinct process paths. Due to the heterogeneity of the weld process and microstructure in time, the initial approach to use resulting microstructures from chosen velocities was taken. Currently, the example microstructures are varying only a single process parameter, future work will require the mixing of various process parameters to ensure distinct clustering from resulting microstructures.

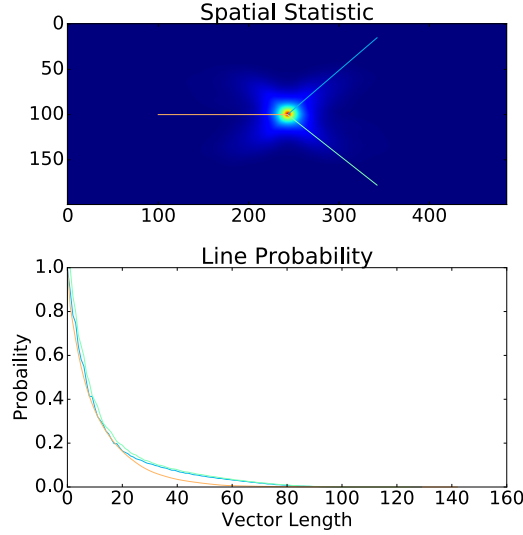


Fig. 3.5: Spatial statistics for the weld surface are shown in the top figure. Statistics are extracted from the orange, green and blue lines. The probabilities on the lines are shown in the bottom figure. The blue and green line diverge noticeably from the orange line around a vector length of 20.

**3.3. Chord length distribution-weld model.** Another approach to quantifying weld microstructures is the use of various chordal analyses. The aim is to perform calculations on spatially homogeneous regions. This is done by taking rectangular volumes spanning the whole length of the weld direction, or  $y$ -axis of the microstructures. These rectangular volumes are defined by window sizes along  $y$ -axis of the microstructure.

Using window sizes of  $(9, 35)$ , chord length distributions  $L_y$  and  $L_z$  along the  $y$  and  $z$  respectively are calculated. The partitioning of these regions is shown in Figure 3.9. The resulting chord length distributions,  $L_y$  and  $L_z$ , for each of the 22 regions are depicted in Figures 3.10, 3.11 and Figures 3.12, 3.13; note that regions span the entire length of the simulation domain along the  $y$  and  $z$ -axes. Each filled curve presents the chord length distribution of a region centered at the  $x$  position along the microstructure. The height of each curve denotes the probability of being within a chord length specified by the axis parallel to the distribution curve in a given region. Each region is specified by a different color. For these microstructures the outermost grains at the bounding simulation domain must be taken into consideration; in the following statistics, the outermost grains were used in calculation of the chord length distribution. This approach is taken since  $L_y$  and  $L_z$  included many regions in which only one or two grains were found across the whole dimension. In order to distinguish these cases the outermost grains are included, however the inclusion or rejection of chords produced by grains at the simulation bounds is still under consideration. The purpose of the windowed volumes in the chord length distribution calculation is to include a larger number of similar grains in order to produce a smooth distribution. If too small a volume is selected the distribution is sampled from a relatively small data set; if too large a volume is selected then heterogeneity is lost because chord

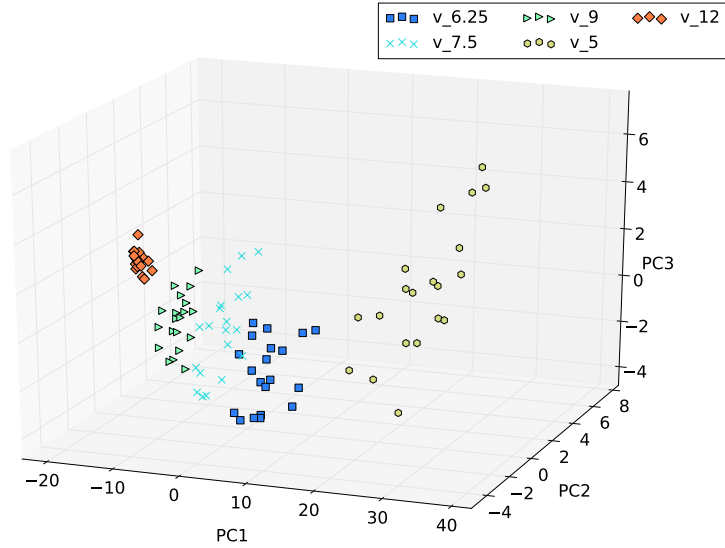


Fig. 3.6: Cluster statistics PCA: The first three PC scores of the cluster statistics for corresponding weld velocities are shown. Each point provides the PC scores for the cluster statistics of a microstructure where each color denotes a different weld velocity.

lengths representing many different grain sizes and shapes are lumped together.

Figures 3.10, 3.10 and 3.12, 3.13 show that the probability of being within a longer chord length grows as the weld axis is approached. The chord length variation also grows as the weld axis is approached, unless a region is dominated by grains that travel through the whole domain in the respective direction (Figures 3.12, 3.13). Further analysis is difficult due to the jagged distributions. There are smooth distributions in the outer regions where the majority of the grains are in the base material and are relatively equiaxed. As the weld axis is approached grains are larger and more heterogeneous, resulting in a smaller sample set leading to jagged distributions.

PCA is applied to the above chord length distributions as shown in Figure 3.14 and 3.15. In this case both distributions in the weld direction and through the thickness are used as inputs. Results show clustering similar to what is found using the spatial statistics. Again, there is a clear trend between the weld velocity and the first principle component. This distinct trend of increasing velocity with a decrease of PC1 makes apparent the opportunity to form process structure linkages. Comparing Figures 3.6 and 3.7 to Figures 3.14 and 3.15 we see the relative intravariance for different weld velocities are similar as well. However, the PC scores from the differing clusters with respect to velocity appear to overlap more using the chord length distributions compared to the cluster statistics. The amount of variance captured with the first principle component is 66.8 percent while three components capture 78.6 percent of the variance. The *scree* plot characterizing the amount of variance captured with the addition of principle components is shown in Figure 3.16. The PCA results from the cluster statistics appear to be more promising for establishing models in the lower dimensional PC space for the weld microstructures. Although the cluster statistics provide better results in this case, the technique used to find the chord length distribution may provide better results for other microstructures. Both methods are retained in order

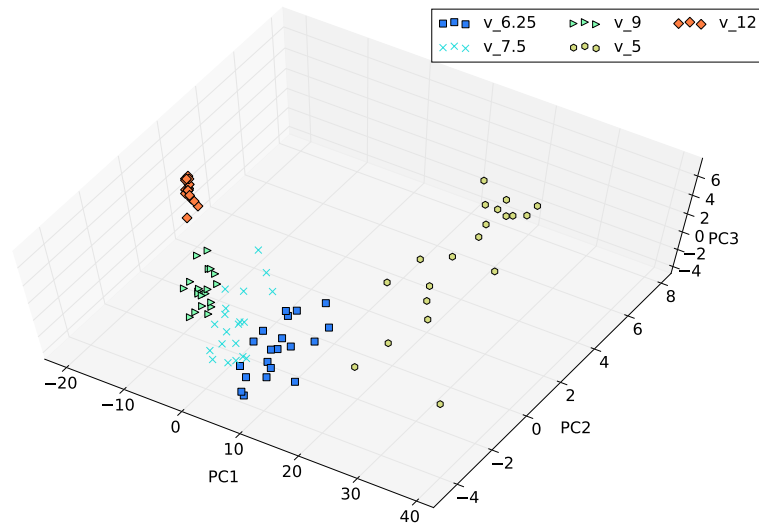


Fig. 3.7: Cluster statistics PCA: View 2

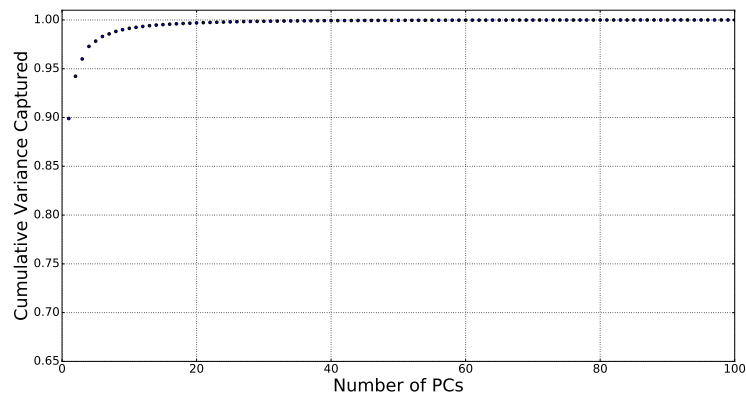


Fig. 3.8: Scree Plot for PCA of cluster statistics. The scree plot depicts the amount of expression, or variance, captured with the addition of principle components. Through the application of cluster statistics, 96 percent of the variance across statistics is captured using 3 principle components.

to provide more options in future material process-structure linkage problems.

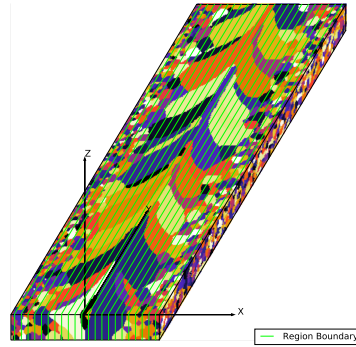


Fig. 3.9: Depiction of windowing used in examples; window size  $(9, 35)$ ; microstructure corresponds to weld velocity 7.5.

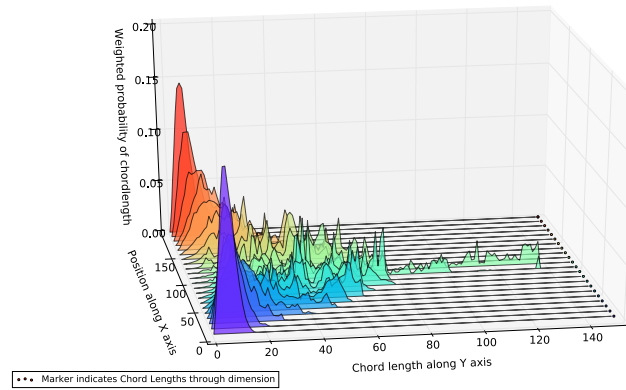


Fig. 3.10: Chord length probability along weld direction,  $L_y$ , weld velocity 7.5, View 1.

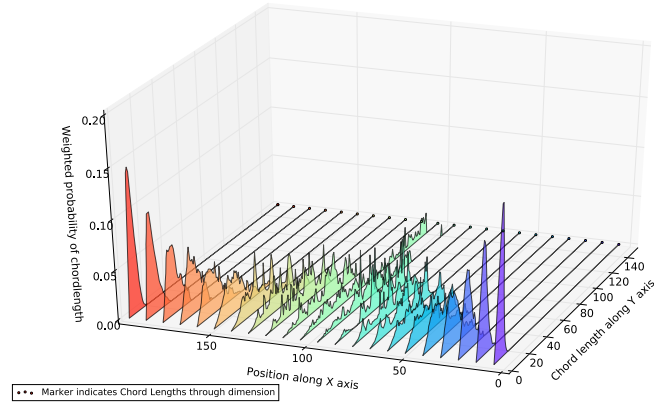


Fig. 3.11: Chord length probability along weld direction,  $L_y$ , weld velocity 7.5, View 2.

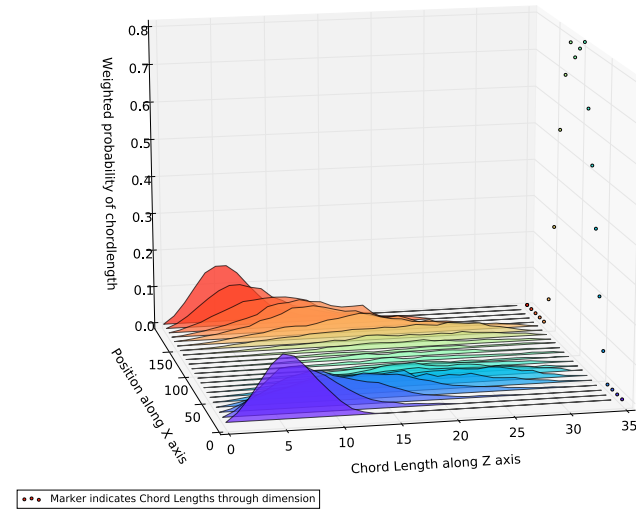


Fig. 3.12: Chord length probability along thickness direction,  $L_z$ , weld velocity 7.5, View 1.



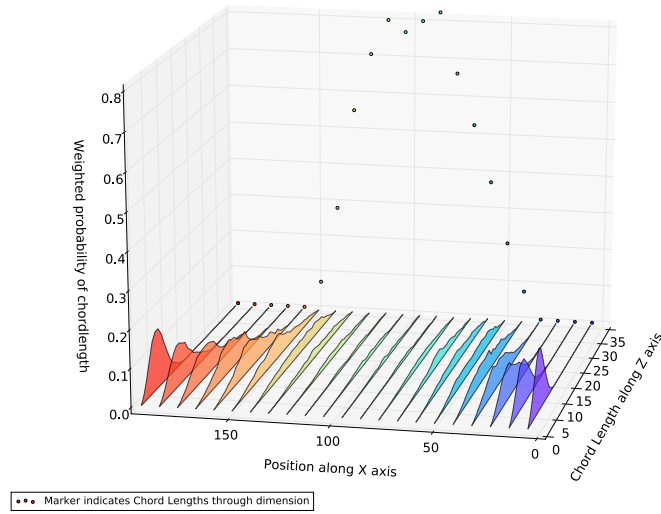


Fig. 3.13: Chord length probability along thickness direction,  $L_z$ , weld velocity 7.5, View 2.

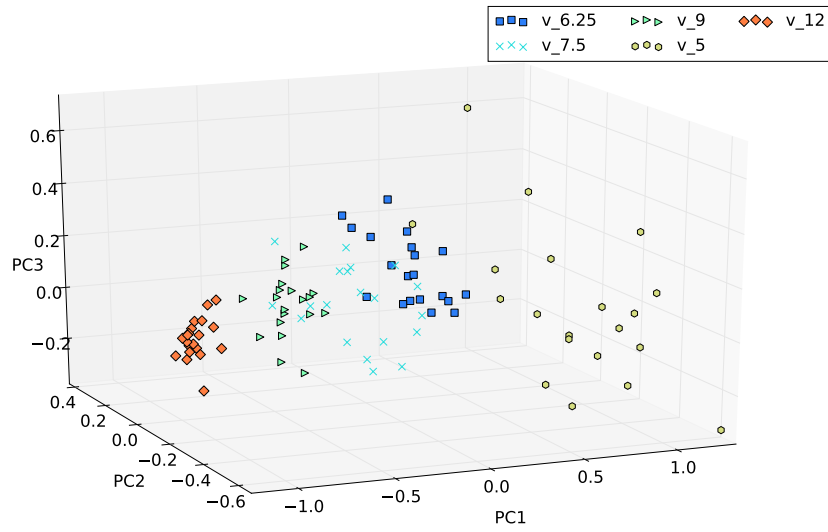


Fig. 3.14: PCA of Combined Chord Length Distributions: The first three PC scores of the Chord Length Distributions for corresponding weld velocities are shown. Each point provides the PC scores for the Chord Length Distribution of a microstructure where each color denotes a different weld velocity.

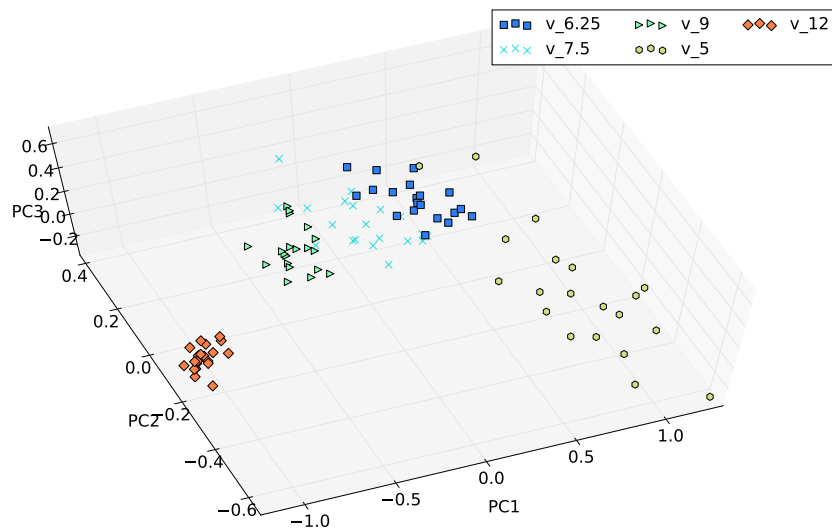


Fig. 3.15: PCA of Combined Chord Length Distributions: View 2

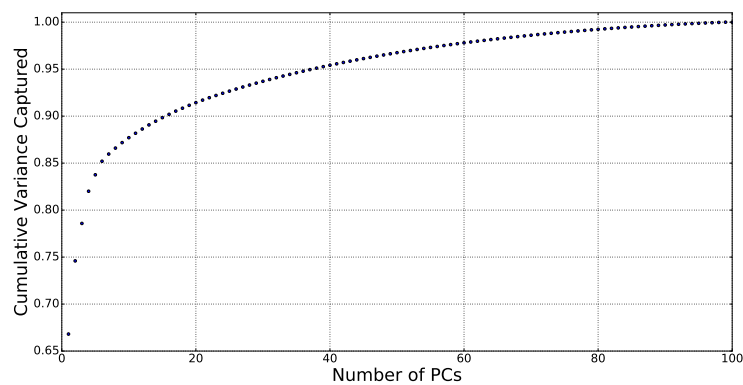


Fig. 3.16: Scree plot for PCA of Combined Chord Length Distributions. The scree plot depicts the amount of expression, or variance, captured with the addition of principle components. Through the application of the chord length distribution statistics, 78.6 percent of the variance across statistics is captured using 3 principle components.

**4. Conclusions.** This report documented analytical methods developed for quantitative characterization of digital microstructures such as those generated by the Sandia kinetic Monte Carlo code SPPARKS [6]. Kinetic Monte Carlo models used for grain growth during welding were used to generate microstructures for characterization. Methods for quantifying microstructures were two-point statistics, chord length distributions, specialized windowing strategies for capturing heterogeneities, and PCA for quantifying complex and varied microstructures using simple low dimensional metrics. These methods were demonstrated by applying them to microstructures generated by welding simulations, where all the process variables, except welding velocity, were held constant. While grain shapes were not specifically addressed, the PCA figures showed that grains of different shapes resulting from different weld velocities clustered. The approach used to generate two-point statistics combined all grains from the microstructure image, which in effect smeared heterogeneity; using this approach, some spatial heterogeneity information was lost; windowing strategies to group grains by the time-temperature histories they experienced during welding were more successful in extracting heterogeneity. PCA showed great promise. The variance captured by the first three principle components for the two-point statistics was greater than the variance captured using the chord length distributions. PCA scores obtained from welds formed at different velocities not only clustered but also exhibited a distinct trend of a decrease in the weld velocity process parameter with an increase of PC1.

**5. Planned future work.** Future work for building a surrogate model has been briefly addressed in the conclusions section. The main steps forward regard weld model development in microstructural evolution. This is focused on implementing algorithms in SPPARKS to accommodate constraints and augment grain growth with grain orientations. In order to correctly model orientations, parameters must be chosen in accordance with a material system. Initialized microstructures of a material system may have a preferred orientation. The majority of grain growth within the weld microstructure is controlled by epitaxial growth. Grain growth in SPPARKS focuses on the computation of the change in grain boundary energy; therefore misorientation appears to be a logical choice in modeling orientation. The grain boundary character is conventionally defined by 5 parameters describing the misorientation between neighboring grains and the grain boundary plane. The use of misorientation would be incorporated using boundary mobility and would be dependent upon temperature and the local misorientation relative to neighboring grains. Grain boundary mobility already exists as a parameter in the current grain growth model. The treatment of low and high angle grain boundaries may also need to be taken into consideration separately. For example low angle grain boundaries exhibit increases in grain boundary energy as misorientation increases. Conversely, high angle grain boundaries may be expected to have a constant, maximum grain boundary energy. The above is in no way a comprehensive road map to develop the algorithms necessary to develop microstructural evolution capabilities in SPPARKS but presents a starting point. The incorporation of orientations presents new venues of analysis involving orientation and misorientation distribution functions. The capability also allows extensions to other techniques, such as the implementation of crystal plasticity to extract mechanical properties from differing weld microstructures.

**Acknowledgments.** The authors wish to express appreciation to the following individuals whose support made this work possible: Jim Redmond (ASC P&EM), Anthony Geller (ASC P&EM Advanced Manufacturing), Jeremy Lechman (Industrial CRADA on predicting properties of heterogeneous materials), Daniel Clayton and Chris Jones (Radioisotope Power System Launch Safety), and Veena Tikare (manager, Multiscale Science).

## REFERENCES

- [1] O. ALTER, P. O. BROWN, AND D. BOTSTEIN, Singular Value Decomposition for Genome-Wide Expression Data Processing and Modeling, *Proceedings of the National Academy of Sciences of the United States of America*, 97 (2000), pp. 10101–10106.
- [2] C. C. BATTAILE, The Kinetic Monte Carlo Method: Foundation, Implementation, and Application, *Computer Methods in Applied Mechanics and Engineering*, 197 (2008), pp. 3386–3398.
- [3] A. GARCIA, V. TIKARE, AND E. HOLM, Three-Dimensional Simulation of Grain growth in a Thermal Gradient with Non-Uniform Grain Boundary Mobility, *Scripta Materialia*, 59 (2008), pp. 661–664.
- [4] S. R. KALIDINDI, *Hierarchical Materials Informatics, Novel Analytics for Materials Data*, Elsevier, 2015.
- [5] J. A. MITCHELL AND V. TIKARE, A Model for Grain Growth During Welding, In preparation, (2016).
- [6] S. PLIMPTON, C. BATTAILE, M. CHANDROSS, L. HOLM, V. TIKARE, G. WAGNER, E. WEBB, AND X. ZHOU, Crossing the Mesoscale No-Man’s Land via Parallel Kinetic Monte Carlo, Tech. Rep. SAND2009-6226, Sandia National Laboratories, October 2009.
- [7] T. RODGERS, J. MADISON, V. TIKARE, AND M. MAGUIRE, Predicting Mesoscale Microstructural Evolution in Electron Beam Welding, *JOM*, 68 (2016), pp. 1419–1426.

## DENSITY FUNCTIONAL THEORY APPLIED TO TRANSITION METAL ELEMENTS AND BINARIES: DEVELOPMENT, APPLICATION, AND RESULTS OF THE V-DM/17 TEST SET

ELIZABETH R. DECOLVENAERE\* AND ANN E. MATTSOON†

**Abstract.** Density functional theory (DFT) is undergoing a shift from a descriptive to a predictive tool in the field of solid state physics, heralded by a spike in “high-throughput” studies. However, methods to rigorously evaluate the validity and accuracy of these studies are lacking, raising serious questions when simulation and experiment disagree. In response, we have developed the V-DM/17 test set, designed to evaluate the ability of DFT’s various implementations to reproduce experimental results in periodic transition metal solids. Our test set evaluates 26 transition metal elements and 80 transition metal alloys across three physical observables: lattice constants, elastic coefficients, and formation energy of alloys. Whether or not a functional can accurately evaluate the formation energy offers key insights into whether the relevant physics are being captured in a simulation, an especially important question in transition metals where active *d*-electrons can thwart the accuracy of an otherwise well-performing functional. Our test set captures a wide variety of cases where the unique physics present in transition metal binaries can undermine the effectiveness of “traditional” functionals. By application of the V/DM-17 test set, we aim to better characterize the performance of existing functionals on transition metals, and to offer a new tool to rigorously evaluate the performance of new functionals in the future.

**1. Introduction.** In the last decade, density functional theory (DFT) has shifted in use from a primarily *descriptive* to an increasingly *predictive* tool in the field of solid state physics [42]. However, compared to the field of quantum chemistry, solid-state DFT has only the most rudimentary standards of verification, validation, and reproducibility [11] – vital aspects in the design and use of a predictive method. The shift in focus, from descriptive to predictive, has been heralded by a spike in high-throughput studies [39, 65, 5, 9, 12, 21, 38], which leverage increasingly powerful and inexpensive computing resources to churn through tens or hundreds of thousands of structures in search of novel materials, properties, or behavior. Unfortunately, without rigorous methods to evaluate the accuracy and precision of DFT-derived results, the reliability of any of these studies is questionable when experiments and simulation disagree [15, 48], as is often the case.

While several verification and validation studies have been performed on periodic systems [36, 37, 26, 72, 41, 24], these simulations all have compared various DFT codes against each other, and focused on lattice constants or volume, or were limited to a small and not necessarily representative set of pure or binary metals and oxides. The most popular high-throughput databases have been excellent at providing input files capable of reproducing published outputs [25, 60, 13], but usually concentrate on only the Perdew-Burke-Ernzerhof (PBE) [49, 50] functional. Individual studies in other publications, while sometimes encompassing a greater spread of functionals, often provide insufficient information on input parameters such that results are difficult to compare and impossible to reproduce. Regardless of the context, precious little attention has been given to situations where DFT and experiment have strong, fundamental disagreements.

To address the above issues, we have developed a test set designed to evaluate the ability of DFT’s implementations to reproduce experimental results in periodic transition metal (TM) solids. Our test set, V-DM/17<sup>1</sup>, evaluates 26 TM elements and 80 TM alloys across three physical observables: lattice constants, elastic coefficients, and formation energy of

\*University of California, Santa Barbara, elizabeth@umail.ucsb.edu

†Sandia National Laboratories, aematts@sandia.gov

<sup>1</sup>The test-set name was chosen in the style of quantum chemistry test sets, V coming from the software package VASP used to develop the set, DM for Decolvenaere and Mattsson as the two contributors, and 17 from the year of completion, 2017

alloys. Our enumeration of structures for the binary compounds takes a hybrid approach, blindly enumerating configurations containing up to six atoms in the symmetrically-distinct supercell, and also containing experimentally observed phases (when known) and similar supercell decorations. Overall, our full test set contains roughly 12,000 structures; we additionally aim to provide mini and micro test sub-sets for fast, computationally-cheap initial evaluations.

The V-DM/17 test set is designed to capture the wide variety of unique physics that arise in TM alloys, a consequence of the interaction of chemically active *d*-electrons [43]. These interactions are often ill-captured by present functionals [22], as most functionals are designed to focus on and accurately solve one “model” electron system or situation. For example, the local density approximation [28] (LDA) functional models the uniform electron gas. Some functionals, such as PBE for solids (PBESol) [54] and the Armiento and Mattsson functional (AM05) [3, 40], are modeled to solve systems with more “mixed” character, but these functionals still fail to capture the *simultaneous* highly localized and delocalized electron behavior that can arise in systems with active *d*- and *s*- electrons. Additionally, multiple elements and many alloys among the TMs have strong and sometimes complex magnetic characteristics [61]; our test set is designed to include representative cases of potential magnetic behaviors. Finally, late-period TMs have important chemical contributions from spin-orbit coupling [59, 66], a phenomena very poorly modeled by most common formulations of DFT [47]. Our test set considers multiple alloy systems where the presence or lack of spin-orbit coupling is critical in correctly predicting material properties.

**2. Simulation Methods.** All of our simulations were carried out using the Vienna Ab initio simulation package [29, 30, 32, 31] (VASP) with projector-augmented wave (PAW) pseudopotentials [6, 33]. Enumeration of superstructures, as well as management of high-throughput simulations, was handled by the CASM: A Clusters Approach to Statistical Mechanics [57, 64, 58, 16] software package. All calculations were performed at zero temperature and pressure, with spin polarization, using the interpolation formula of Vosko, Wilk, and Nusair [70], and with automatically-generated, gamma-centered *k*-point meshes [45]. Electronic self-consistency loops were converged to energy changes below  $10^{-3}$  meV, while ionic relaxations were converged to energy changes below 1 meV. The general simulation scheme consisted of two steps: first, repeated relaxation runs were performed until the simulation converged within 3 or fewer ionic steps, then, a final static run was performed with no changes allowed in cell geometry or ionic positions. This scheme was chosen to avoid problems introduced by possible non-isotropic changes in the basis set over a volume change during a single run. A quasi-Newton ionic relaxation scheme was used, RMM-DIIS [32], as implemented in VASP. During relaxation runs the method of Methfessel-Paxton [44] was used for electronic smearing. In most cases, the final static run switched to the tetrahedron method with Blöchl corrections [7] for smearing. Energy cutoffs, as well as *k*-point meshes, were chosen for each element and binary using the techniques described in Section 2.1. Specific functionals used, as well as special simulation schemes used with those functions, are described in Section 2.2.

**2.1. Convergence Criteria.** As formation energy forms such a large component of our test set metrics, ensuring numerical convergence is especially critical. Rather than aim for a particular *k*-point mesh density or energy cutoff in some global sense, each element was treated individually and converged, with respect to *k*-point mesh and energy cutoff, to within 1 meV. This was achieved in two stages:

First, the *k*-point mesh was optimized, using the fully-automatic *k*-point generation

scheme provided by VASP. For each element, a series of  $k$ -point lengths<sup>2</sup> were enumerated, and calculations were performed with energy cutoffs of 150% of the VASP default value. Each cell was fully relaxed, and then a static run was performed. When the energy difference between two subsequent  $k$ -point lengths was less than 1 meV and *remained* less than 1 meV at increasing  $k$ -point densities, this value was taken as the converged  $k$ -point length. Step sizes were of 1 angstrom, and the starting  $k$ -point length was 30 angstroms.

Next, using the  $k$ -point length previously determined, a series of energy cutoffs were enumerated starting from 150% of the VASP default cutoff for that element. Again, the starting geometry was that of the experimental primitive cell, and each energy cutoff was allowed to independently relax, and a final static run was used to collect data. When the energy difference between two subsequent energy cutoffs was less than 1 meV and *remained* less than 1 meV at increasing cutoffs, this value was taken as the converged energy cutoff. Step sizes were 5 eV.

In all cases, starting structures were taken from the Materials Project [25] database, using primitive cells corresponding to the lowest-temperature experimentally observed structure. For binary compounds, the higher of the two components'  $k$ -point length and energy cutoff was used. Both  $k$ -point and energy cutoff convergences were independently performed for each functional explored. While not used as a *convergence* criteria, the convergence of cell volume and lattice constant *at* the converged  $k$ -point and energy cutoff values was collected.

**2.2. Functionals Explored.** To demonstrate the wide variety of behaviors captured by our test set, we ran simulations across eight different functionals, detailed in table 2.1. Our purpose here was two-fold: our test set characterizes the behavior of these functionals in the case of transition metals, and the differences in behaviors across these functionals demonstrates the necessity of our test set. The functionals chosen represent a wide variety of techniques used in DFT in materials science and different rungs on the “Jacob’s ladder” [55] of functional accuracy. Specifically, we aimed to analyze the most commonly used functionals (PBE and PBEsol), in addition to a representative of the local density approximation, historically the most commonly used generalized gradient approximation functionals, a subsystem functional, and meta-generalized gradient approximation functionals with historically good performance for transition metals.

All functionals were fully and self-consistently converged with regards to both electronic and ionic degrees of freedom (lattice constants and atom positions). For PZ, PW91, BLYP, PBE, and PBEsol, simulations were carried out in a straightforward manner using the techniques described at the start of this section. For AM05, rTPSS, and M06-L, each configuration first underwent a single relaxation run using PBE. Then, the resulting wavefunctions were used as the starting condition for the chosen functional<sup>3</sup>. For AM05 and M06-L, the chosen algorithms (All/Damped) were incompatible with tetrahedral smearing, and so the final static run was also performed using the smearing method of Methfessel-Paxton. The GW-labeled PAW-PBE potentials from the v.54 set [67] were used in all simulations except for PZ, where the GW-labeled PAW-LDA potentials were used instead.

We have specifically chosen not to analyze any DFT+U [2] techniques. The DFT+U functionals, which employ a Hubbard-U correction to the on-site coulomb interaction term, add an additional one or two degrees of freedom to our elements and binaries, respectively.

<sup>2</sup>A  $k$ -point “length” generates a  $k$ -point mesh along reciprocal direction  $\hat{i}$  with  $\max(1, l_k |\vec{b}_i| + 0.5)$  spacings, where  $l_k$  is the  $k$ -point length, and  $\vec{b}_i$  is the reciprocal lattice vector along  $\hat{i}$ .

<sup>3</sup>Specifically, a single run was performed using the PBE functional, and then the WAVECAR containing the Kohn-Sham wavefunctions was copied over for use as a starting condition for the next run, this one using the listed functional.

Name	Type	Algo
PZ [56]	Local Density Approximation	Normal
PW91 [51, 52]	Generalized Gradient Approximation	Normal
BLYP [4, 35]	Generalized Gradient Approximation	Normal
PBE [49, 50]	Generalized Gradient Approximation	Normal
PBESol [54]	Generalized Gradient Approximation	Normal
AM05 [3, 40]	Sub-System Functional	All/Damped
rTPSS [63, 53]	Meta-Generalized Gradient Approximation	Normal
M06-L [71]	Meta-Generalized Gradient Approximation	All/Damped

Table 2.1: Table of functionals used to test the V-DM/17 test set, indicating the functional’s common name (usually an acronym), the general category of the functional, and the electronic convergence algorithm (according to VASP’s tags) used with that functional.

Additionally, the U’s are semi or totally empirical, usually chosen to match up some experimental parameter (often the band gap) to a simulation result. While some self-consistent methods exist to determine the U *in-situ* [34, 10, 1], these approaches are computationally involved and therefore beyond the scope of our analysis.

We have also neglected to include any hybrid-type functionals, owing primarily to the massively increased computational cost associated with hybrids. Not only can hybrid functionals be thousands of times slower and have significant memory requirements [62, 8, 17], but additionally, results for transition metals in the past have been poor or aphysical [26, 17, 22]. Finally, for both DFT+U and hybrid functionals, these techniques result in energies that are no longer purely *functionals* of the electron density, and hence, these techniques are not “true” Kohn-Sham DFT.

While the GW PAWs include scalar-relativistic corrections, we have not made any attempts to include additional relativistic terms in our analyses. While all calculations were performed using spin-polarization, the magnetic moments calculated were both collinear and decoupled from the crystal structure. Although for some TM binaries non-collinear magnetic effects (e.g., in Ni-Fe [61]) or spin-orbit coupling contributions (e.g., in Co-Pt [46]) can be important, the computational difficulty of performing these calculations in VASP puts them beyond the reach of our current high-throughput infrastructure.

**3. The V-DM/17 Test Set.** The V-DM/17 test set consists of 26 single-element TM systems and 80 TM binaries, and contains approximately 12,000 structures. Our choice of systems represents roughly one quarter of the potential space spanned by all possible TM binaries, offering a compromise between completeness and computational efficiency. While use of the entire structure space is recommended, we will in the future provide guidelines for “mini” and “micro” versions of the test set that consider decreasingly sized subsets of increasing significance. While significantly larger than the test sets traditionally found in quantum chemistry, the efficiency of periodic codes for “simple” bulk systems (i.e., small unit cells), as well as the increasing availability of cheap but powerful computational resources, ensures our test set remains accessible and justifiable in use.



**3.1. Single-Component Systems.** Of the potential space of transition metals, we have chosen to include elements in periods 4–6 and groups 3–11, excluding Lutetium. Lu is often considered a lanthanide instead of a transition metal; furthermore, the  $f$ -electrons in Lu actively participate in determining the electronic structure [18], putting it outside the scope of this test-set. Group 12, containing zinc, cadmium, and mercury, was also excluded as these elements contain completely filled  $d$ -shells and therefore behave significantly differently than other metals in their respective periods. For the 26 remaining elements, the experimentally observed lowest-temperature crystal structure was used to select the primitive cell, with geometry inputs acquired from the Materials Project database. These primitive cells were used for all convergence tests, as well as inputs for enumeration for binary mixtures.

**3.2. Binary Systems.** Combinatorially,  $\binom{26}{2} = 325$  binary mixtures of our selected TMs exist; from this we selected a subset of 80 according to several criteria. We wished our test-set to explore as many different electronic structure phenomena as possible while utilizing the minimum number of required calculations, both counting by number of binaries, and total number of structures. Our constraints, in order of priority, were then:

1. Each element is present in 6–7 binaries
2. Each element shares a binary with 2–3 elements from each period
3. No element shares a binary with two elements in the same period and in adjacent groups
4. Each element participates in at least one binary where it is known to form an ordered mixed phase, and one where it is known to form no ordered mixed phases
5. Each element shares a binary with at least one element from each space group

Constraints 1, 2, and 3 were strictly adhered to; constraint 4 was adhered to as a fortunate consequence of constraints 1–3, and constraint 5 was adhered to when possible. For elements like Mn with a unique space group for the primitive cell, due to constraints 1 and 2, only some elements could share a binary with Mn. The resulting binaries chosen, as well as a qualitative view of *all* possible binaries with experimental mixing behavior indicated, is shown in Figure 3.1.

For enumerating structures, if both elements in the binary shared a space group, the element with the lower atomic number was used to prototype the enumeration. If both elements did not share a space group, two enumerations were performed, one using each primitive cell. Enumeration was carried out by constructing all symmetrically-distinct supercells containing up to 6 atoms, and decorating those supercells with all possible symmetrically-distinct configurations. This scheme resulted in 137 unique configurations for all structures enumerated from one-atom primitive cells, and 63 unique configurations for all structures enumerated from two-atom supercells. If one or more known ordered phases formed in a binary, that ordered phase was also included in the enumeration, as well as any alternate decorations of the underlying supercell if the supercell contained 10 or fewer atoms. Otherwise, only the exact, experimentally-observed structure was enumerated. If the ordered phase did not share a space group with either of the elemental primitive cells, this process was carried out separately as a third enumeration. In the case of Mn, owing to the relatively large size of the primitive cell (32 atoms), a body-centered-cubic primitive cell was used for enumeration of Mn binaries, though the true primitive cell was used as a chemical reference.

**3.3. Experimental Data.** Experimental data was derived from a variety of resources, though the authors have attempted to use as few *different* sources as possible to ensure regularity of the results. Owing to the large nature of this study, several databases were used, including Materials Project [25], the ASM Alloy Phase Diagram Database [69], and the Springer Materials [68] search tool. When multiple sources existed, the most modern experimental source was used; this choice is not a statement of distrust in older results, but

Element		Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Cu	Y	Zr	Nb	Mo	Tc	Ru	Rh	Pd	Ag	Hf	Ta	W	Re	Os	Ir	Pt	Au
Period		4	4	4	4	4	4	4	4	4	5	5	5	5	5	5	5	5	5	6	6	6	6	6	6	6	6
Group		3	4	5	6	7	8	9	10	11	3	4	5	6	7	8	9	10	11	4	5	6	7	8	9	10	11
	SG																										
Sc	4	3																									
Ti	4	4																									
V	4	5																									
Cr	4	6																									
Mn	4	7																									
Fe	4	8																									
Co	4	9																									
Ni	4	10																									
Cu	4	11																									
Y	5	3																									
Zr	5	4																									
Nb	5	5																									
Mo	5	6																									
Tc	5	7																									
Ru	5	8																									
Rh	5	9																									
Pd	5	10																									
Ag	5	11																									
Hf	6	4																									
Ta	6	5																									
W	6	6																									
Re	6	7																									
Os	6	8																									
Ir	6	9																									
Pt	6	10																									
Au	6	11																									

P6<sub>3</sub>/mmc P6<sub>3</sub>/mmm Im3m I43m Fm3m

Element Space Groups

Immiscible No Ordered Phases Ordered Phases Sigma Phase

Mixed Phases

X

Selected

Fig. 3.1: Table of transition element binaries, ordered by period then group, with the space group (SG) of the lowest-temperature stable solid listed. Binaries selected for inclusion in the V-DM/17 test set are indicated by purple X's. In each cell the experimentally-observed mixing behavior of the two alloys is given by color: red indicates a miscibility gap, orange indicates a random solid solution, green indicates the presence of ordered phases, while blue indicates that only a sigma (quasi-ordered) phase is formed.

only is meant to serve as an objective and simple selection criteria.

A summary of the experimental techniques utilized for each parameter is given in Table 3.1. As the individual measurements used in this study were taken from a wide variety of sources, a general guideline as to that measurement's precision is given, rather than a specific error estimate. For lattice constants and elastic coefficients, the precision is expressed as  $\frac{\Delta i}{i} \times 100\%$  where  $i$  is some property, whereas for formation enthalpy a more direct estimate of  $\Delta i$  can be made.

Some consideration must be given in comparing experimental measurements taken at *finite* temperatures, versus DFT simulations performed at zero kelvin. Thermal expansion effects, as well as zero-point phonon energies, can modify lattice constants, elastic coef-

Parameter	Technique	Precision
Lattice Constant	Powder Diffraction	$< 0.1\%$ [23]
Elastic Coeff.	Continuous Wave Ultrasonic Interferometry	$< 0.1\%$ [14]
Formation Enthalpy	High Temperature Direct Reaction Calorimetry	$\pm 10$ meV [27]

Table 3.1: Table of experimental techniques used to acquire the various parameters used as a standard for the test-set. Precision estimates are taken from general analysis of the technique or of the instruments used, and not from individual experiments.

ficients, and formation enthalpies. In the case of lattice constants, these corrections are smaller than 1% of the value and strictly decrease the lattice constant [37, 20], and so have been neglected. Determining elastic coefficient corrections requires pressure derivatives of the coefficients, which are lacking in experimental information [37], and so have also been neglected. Finally, the order of the corrections to the formation enthalpies are on the order of 10 meV [37, 26], which is within the precision of our experimental technique, and so can also be neglected.

**4. High-Throughput Results.** At the present moment, 33,309 single-element convergence calculations and 146,888 production binary calculations have been performed across 8 functionals. These calculations represent full convergence calculations for 25 of the 26 individual elements (Mn is still in-progress for several functionals), and approximately one half of the production runs for the binaries. Completion of all calculations, numbering approximately 300,000 individual VASP runs, is projected for February of 2017. Failures that could not be automatically recovered from have occurred in less than 1% of simulations, however, this still represents hundreds of calculations that must be inspected and restarted by-hand.

**4.1. Single-Component Systems.** Convergence statistics for the completed elements can be found in Tables 4.1 and 4.2, showing  $k$ -point grids and energy cutoffs resulting in energy differences  $\leq 1$  meV, respectively. Convergence simulations were primarily straightforward, however, in several cases (e.g., Cu, Zr, Co, Fe), the number of bands used had to be manually increased after several warnings and errors were encountered. Additionally, for Y, the GW potentials resulted in non-converging electronic self-consistency loops, and so were subsequently replaced by the non-GW versions of those potentials. In the majority of cases, the smallest  $k$ -point mesh density tested (corresponding to a length of 30 angstroms in the fully automatic scheme) already achieved 1 meV convergence, however, in a few cases (especially in periods 9 and 10), a significantly greater number of  $k$ -points had to be used. For energy cutoffs, 1.5 times the default value provided by VASP usually proved sufficient. Unsurprisingly, AM05, rTPSS, and M06-L required both the densest  $k$ -point meshes and the highest energy cutoffs; these convergence needs are explained by the finer-grained information required to correctly determine subsystem parameters and kinetic energy densities, respectively.

**4.2. Binary Systems.** For binary systems, in addition to the difficulties encountered in single elements, a new challenge was introduced in the form of highly unstable structures. Since the enumeration technique we employed relied solely on symmetry (to produce unique structures), some generated structures proved to be unstable. These structures would, through relaxation, reconfigure into either a new structure or another existing structure.

Element	PZ	PW91	BLYP	PBE	PBESol	AM05	rTPSS	M06-L
Ag	20×20×20	13×13×13	13×13×13	13×13×13	20×20×20	30×30×30	20×20×20	25×25×25
Au	13×13×13	13×13×13	13×13×13	13×13×13	13×13×13	33×33×33	13×13×13	25×25×25
Co	20×20×11	19×19×10	19×19×10	19×19×10	20×20×11	16×16×9	26×26×14	22×22×12
Cr	16×16×16	15×15×15	15×15×15	15×15×15	16×16×16	29×29×29	16×16×16	28×28×28
Hf	11×11×6	12×12×7	12×12×7	12×12×7	12×12×7	14×14×8	12×12×7	11×11×6
Ir	24×24×24	24×24×24	24×24×24	24×24×24	24×24×24	30×30×30	24×24×24	33×33×33
Mo	16×16×16	15×15×15	15×15×15	16×16×16	16×16×16	27×27×27	16×16×16	22×22×22
Nb	14×14×14	14×14×14	14×14×14	14×14×14	14×14×14	25×25×25	14×14×14	22×22×22
Ni	21×21×21	21×21×21	24×24×24	21×21×21	21×21×21	32×32×32	24×24×24	24×24×24
Os	15×15×8	15×15×8	15×15×8	16×16×9	16×16×9	29×29×16	16×16×9	24×24×13
Pd	23×23×23	23×23×23	19×19×19	23×23×23	23×23×23	30×30×30	23×23×23	28×28×28
Pt	21×21×21	20×20×20	20×20×20	20×20×20	21×21×21	22×22×22	21×21×21	31×31×31
Re	16×16×9	18×18×10	16×16×9	18×18×10	15×15×8	20×20×11	15×15×8	17×17×9
Rh	19×19×19	22×22×22	25×25×25	19×19×19	19×19×19	18×18×18	22×22×22	23×23×23
Ru	16×16×9	16×16×9	18×18×10	16×16×9	16×16×9	25×25×14	16×16×9	29×29×16
Sc	11×11×6	11×11×6	11×11×6	11×11×6	11×11×6	11×11×6	11×11×6	11×11×6
Ta	13×13×13	13×13×13	13×13×13	13×13×13	13×13×13	23×23×23	13×13×13	16×16×16
Tc	17×17×9	17×17×9	17×17×9	13×13×7	17×17×9	22×22×12	13×13×7	19×19×10
Ti	13×13×7	12×12×7	12×12×7	12×12×7	12×12×7	12×12×7	12×12×7	12×12×7
V	16×16×16	15×15×15	16×16×16	15×15×15	16×16×16	22×22×22	15×15×15	20×20×20
W	15×15×15	15×15×15	15×15×15	15×15×15	15×15×15	21×21×21	15×15×15	21×21×21
Y	14×14×8	14×14×8	14×14×8	14×14×8	14×14×8	14×14×8	14×14×8	14×14×8
Zr	12×12×7	11×11×6	11×11×6	11×11×6	11×11×6	14×14×8	11×11×6	14×14×8

Table 4.1: Converged values of the  $k$ -point mesh for 23 elements, across 8 functionals.

Relaxations of this sort result in DFT energies that no longer represent the original structure. This was handled by discarding any offending configurations. Specifically, by measuring deformations both of the unit cell and of the basis sites from the start to the end of a relaxation, a “mapping score” was generated. All structures in a binary family were mapped onto all other structures, and if a configuration mapped better (scored lower) onto any structure other than its starting structure, the configuration was removed. These relaxations tended to occur only in certain binaries, though even in these situations no more than 10% of a binary’s enumerated configurations were discarded.

Although much of the calculation work remains to be finished, some interesting trends have already begun to emerge. In binaries where no mixed phases were experimentally observed, but the two components were mutually soluble, the PBE functional usually predicted a series of shallow ( $< -10$  meV formation enthalpy) stable ordered phases. However, in cases where not only were there a lack of ordered phases, but the two elements were experimentally immiscible, no stable ordered were predicted ( $> 0$  meV formation enthalpy). In binaries known to form one or more ordered phases, most of the configurations calculated were negative in enthalpy, though usually not as negative as the experimentally observed ordered phases. Additionally, alloys with two elements with different space groups had a higher percentage of structures that relaxed onto different structures, than alloys that shared

Element	PZ	PW91	BLYP	PBE	PBESol	AM05	rTPSS	M06-L
Ag	535	535	535	590	535	535	580	640
Au	470	505	515	495	465	465	515	525
Co	635	635	635	635	635	635	685	700
Cr	575	575	575	575	575	575	655	635
Hf	425	425	425	425	425	425	425	425
Ir	535	535	535	535	535	535	535	535
Mo	525	525	525	525	525	525	525	525
Nb	535	535	535	535	535	535	535	600
Ni	585	585	585	585	595	585	665	710
Os	535	535	535	510	535	535	535	535
Pd	535	570	535	545	535	535	580	605
Pt	515	505	505	560	505	505	560	505
Re	505	505	505	505	475	950	505	585
Rh	525	525	525	525	525	525	580	525
Ru	525	525	525	525	525	525	525	525
Sc	565	565	565	565	565	565	565	565
Ta	435	435	435	435	435	435	435	435
Tc	525	525	525	525	525	525	525	530
Ti	575	575	575	575	575	575	575	575
V	575	575	575	575	575	575	595	605
W	485	485	485	485	485	485	485	485
Y	515	585	515	515	595	515	515	640
Zr	515	515	515	515	515	560	645	625

Table 4.2: Converged values of the energy cutoff, in eV, for 23 elements, across 8 functionals.

a space group (again, in the PBE functional).

**5. Discussion of Design Choices.** Our study places a special emphasis on our third experimental comparison — the formation energy. This parameter plays a critical role in determining whether a given approximation of DFT (i.e., a specific functional) successfully captures the relevant physics present in a material. While present functionals (e.g., PBEsol and AM05) can provide excellent results for lattice constants, in materials with active *d*-electrons (e.g., TMs) the predicted formation energies can be off by a factor of two or more. These results are surprising, but can be understood considering the differences in how formation energy and lattice constant are calculated. DFT, as presently implemented, provides only a *direct* measure of formation energies. Specifically, the only density functional we “know” how to evaluate is one of energy. All other properties, including lattice constant, are second or higher order extrapolations from the energy, usually found via minimizing the energy with respect to some perturbation in said property. Therefore, for correct evaluation of the lattice constant and other secondary properties, given proper convergence settings [42], the functional only needs to give a good representation of the derivative of the energy near equilibrium.

a small number of previous studies have taken inspiration from quantum chemistry and utilized the cohesive energy as an experimental measure of the accuracy of a functional [26, 37]. While attractive due to the ease of calculation as well as the availability of experimental data for TMs, this approach overlooks important differences between how quantum chemistry and solid-state physics handle DFT. Most importantly, solid-state, *periodic* codes are designed to function best in bulk materials with a well-defined unit cell. When calculating surfaces, defects, and isolated atoms, the so-called “image problem” appears if a sufficiently large simulation cell size is not employed, as the isolated feature can begin to see copies of itself across a periodic boundary [19]. Additionally, most functionals developed for periodic materials are based on some solution of the *uniform* electron gas (PBEsol and AM05 being two obvious exceptions), whereas an isolated atom more closely represents a localized system of mostly vacuum. Finally, in systems with higher electron densities, i.e., in a solid, where neighboring atoms can contribute electron density, the fraction of the coulomb energy term made up by self-interaction energy is smaller than in a geometry mostly constructed of vacuum. All of these factors feed into the larger problem that periodic codes are not *designed* for isolated atoms, and while accurate results can be obtained on such systems, such a unique application cannot be considered to be representative of the overall predictive power and accuracy of periodic codes when applied to true periodic systems.

**6. Conclusions.** While DFT has become a powerful, expansive, and insightful tool in the past decade, caution must be taken in the interpretation of results. Specifically, in the case of *d*-electron materials, DFT is known to suffer several difficulties owing to the mixed localized-and-delocalized electronic behavior present. In light of several high-profile failures of DFT to describe mixed TM binaries, we have developed a detailed test-set suited to probe a technique’s accuracy in predicting TM mixture behaviors. Our test set utilizes binary formation enthalpies, an often-neglected characteristic, to provide a more direct measure of the quality of approximations made by a given functional. We have applied our test set to eight of the most commonly used functionals, and compared their performance. No one functional provides an accurate description of all members of our test set, and the differences in their failures highlight the wide variety of unique properties present in TM alloys. The authors urge that forward progress using DFT as a *predictive* rather than a *descriptive* tool must be approached with care and rigor.

- [1] L. A. AGAPITO, S. CURTAROLO, AND M. BUONGIORNO NARDELLI, Reformulation of DFT +  $U$  as a Pseudohybrid Hubbard Density Functional for Accelerated Materials Discovery, *Phys. Rev. X*, 5 (2015), p. 011006.
- [2] V. I. ANISIMOV, F. ARYASETIWAN, AND A. I. LICHTENSTEIN, First-Principles Calculations of the Electronic Structure and Spectra of Strongly Correlated Systems: the LDA +  $U$  method, *Journal of Physics: Condensed Matter*, 9 (1997), p. 767.
- [3] R. ARMIENTO AND A. E. MATTSSON, Functional Designed to Include Surface Effects in Self-Consistent Density Functional Theory, *Phys. Rev. B*, 72 (2005), p. 085108.
- [4] A. D. BECKE, Density-Functional Exchange-Energy Approximation with Correct Asymptotic Behavior, *Phys. Rev. A*, 38 (1988), pp. 3098–3100.
- [5] S. BHATTACHARYA AND G. K. H. MADSEN, High-Throughput Exploration of Alloying as Design Strategy for Thermoelectrics, *Phys. Rev. B*, 92 (2015), p. 085205.
- [6] P. E. BLÖCHL, Projector Augmented-Wave Method, *Phys. Rev. B*, 50 (1994), pp. 17953–17979.
- [7] P. E. BLÖCHL, O. JEPSEN, AND O. K. ANDERSEN, Improved Tetrahedron Method for Brillouin-Zone Integrations, *Phys. Rev. B*, 49 (1994), pp. 16223–16233.
- [8] E. BYLASKA, K. TSEMEKHMAN, N. GOVIND, AND M. VALIEV, *Large-Scale Plane-Wave-Based Density Functional Theory: Formalism, Parallelization, and Applications*, John Wiley & Sons, Inc., 2011, pp. 77–116.
- [9] J. CARRETE, W. LI, N. MINGO, S. WANG, AND S. CURTAROLO, Finding Unprecedentedly Low-Thermal-Conductivity Half-Heusler Semiconductors via High-Throughput Materials Modeling, *Phys. Rev. X*, 4 (2014), p. 011019.
- [10] M. COCCIONI AND S. DE GIRONCOLI, Linear Response Approach to the Calculation of the Effective Interaction Parameters in the LDA +  $U$  Method, *Phys. Rev. B*, 71 (2005), p. 035105.
- [11] C. J. CRAMER AND D. G. TRUHLAR, Density Functional Theory for Transition Metals and Transition Metal Chemistry, *Phys. Chem. Chem. Phys.*, 11 (2009), pp. 10757–10816.
- [12] S. CURTAROLO, G. L. HART, M. B. NARDELLI, N. MINGO, S. SANVITO, AND O. LEVY, The High-Throughput Highway to Computational Materials Design, *Nature Materials*, 12 (2013), pp. 191–201.
- [13] S. CURTAROLO, W. SETYAWAN, S. WANG, J. XUE, K. YANG, R. H. TAYLOR, L. J. NELSON, G. L. HART, S. SANVITO, M. BUONGIORNO-NARDELLI, N. MINGO, AND O. LEVY, AFLOWLIB.ORG: A Distributed Materials Properties Repository from High-Throughput Ab Initio Calculations, *Computational Materials Science*, 58 (2012), pp. 227 – 235.
- [14] J. P. DAY, P. S. HO, AND A. L. RUOFF, A cw Interferometer for Elastic Constant Measurements, *Review of Scientific Instruments*, 44 (1973), pp. 478–481.
- [15] E. DECOLVENAERE, M. J. GORDON, AND A. VAN DER VEN, Testing Predictions from Density Functional Theory at Finite Temperatures:  $\beta_2$ -like Ground States in Co-Pt, *Phys. Rev. B*, 92 (2015), p. 085119.
- [16] A. V. DER VEN, J. THOMAS, Q. XU, AND J. BHATTACHARYA, Linking the Electronic Structure of Solids to their Thermodynamic and Kinetic Properties, *Mathematics and Computers in Simulation*, 80 (2010), pp. 1393 – 1410.
- [17] F. FURCHE AND J. P. PERDEW, The Performance of Semilocal and Hybrid Density Functionals in 3d Transition-Metal Chemistry, *The Journal of Chemical Physics*, 124 (2006).
- [18] E. FURET, K. COSTUAS, P. RABILLER, , AND O. MAURY, On the Sensitivity of  $f$  Electrons to Their Chemical Environment, *Journal of the American Chemical Society*, 130 (2008), pp. 2180–2183. PMID: 18225894.
- [19] M. J. GILLAN, Calculation of the Vacancy Formation Energy in Aluminium, *Journal of Physics: Condensed Matter*, 1 (1989), p. 689.
- [20] P. HAAS, F. TRAN, AND P. BLAHA, Calculation of the Lattice Constant of Solids with Semilocal Functionals, *Phys. Rev. B*, 79 (2009), p. 085104.
- [21] G. L. W. HART, S. CURTAROLO, T. B. MASSALSKI, AND O. LEVY, Comprehensive Search for New Phases and Compounds in Binary Alloy Systems Based on Platinum-Group Metals, Using a Computational First-Principles Approach, *Phys. Rev. X*, 3 (2013), p. 041035.
- [22] J. N. HARVEY, On the Accuracy of Density Functional Theory in Transition Metal Chemistry, *Annu. Rep. Prog. Chem., Sect. C: Phys. Chem.*, 102 (2006), pp. 203–226.
- [23] F. H. HERBSTSTEIN, How Precise are Measurements of Unit-Cell Dimensions from Single Crystals?, *Acta Crystallographica Section B*, 56 (2000), pp. 547–557.
- [24] J. HEYD AND G. E. SCUSERIA, Efficient Hybrid Density Functional Calculations in Solids: Assessment of the HeydScuseriaErnzerhof Screened Coulomb Hybrid Functional, *The Journal of Chemical Physics*, 121 (2004), pp. 1187–1192.
- [25] A. JAIN, S. P. ONG, G. HAUTIER, W. CHEN, W. D. RICHARDS, S. DACEK, S. CHOLIA, D. GUNTER, D. SKINNER, G. CEDER, AND K. A. PERSSON, The Materials Project: A Materials Genome Approach to Accelerating Materials Innovation, *APL Materials*, 1 (2013), p. 011002.
- [26] P. JANTHON, S. A. LUO, S. M. KOZLOV, F. VIES, J. LIMTRAKUL, D. G. TRUHLAR, AND F. ILLAS, Bulk

- Properties of Transition Metals: A Challenge for the Design of Universal Density Functionals, *Journal of Chemical Theory and Computation*, 10 (2014), pp. 3832–3839. PMID: 26588528.
- [27] O. KLEPPA AND L. TOPOR, A New Calorimeter for Temperatures above 1400 K, *Thermochimica Acta*, 139 (1989), pp. 291 – 297.
  - [28] W. KOHN AND L. J. SHAM, Self-Consistent Equations Including Exchange and Correlation Effects, *Phys. Rev.*, 140 (1965), pp. A1133–A1138.
  - [29] G. KRESSE AND J. FURTHMÜLLER, Efficient Iterative Schemes for *ab initio* Total-Energy Calculations Using a Plane-Wave Basis Set, *Phys. Rev. B*, 54 (1996), pp. 11169–11186.
  - [30] G. KRESSE AND J. FURTHMÜLLER, Efficiency of Ab-Initio Total Energy Calculations for Metals and Semiconductors Using a Plane-Wave Basis Set, *Computational Materials Science*, 6 (1996), pp. 15 – 50.
  - [31] G. KRESSE AND J. HAFNER, *Ab initio* Molecular Dynamics for Liquid Metals, *Phys. Rev. B*, 47 (1993), pp. 558–561.
  - [32] G. KRESSE AND J. HAFNER, *Ab initio* Molecular-Dynamics Simulation of the Liquid-Metal~Amorphous-Semiconductor Transition in Germanium, *Phys. Rev. B*, 49 (1994), pp. 14251–14269.
  - [33] G. KRESSE AND D. JOUBERT, From Ultrasoft Pseudopotentials to the Projector Augmented-Wave Method, *Phys. Rev. B*, 59 (1999), pp. 1758–1775.
  - [34] H. J. KULIK, M. COCOCIONI, D. A. SCHERLIS, AND N. MARZARI, Density Functional Theory in Transition-Metal Chemistry: A Self-Consistent Hubbard  $U$  Approach, *Phys. Rev. Lett.*, 97 (2006), p. 103001.
  - [35] C. LEE, W. YANG, AND R. G. PARR, Development of the Colle-Salvetti Correlation-Energy Formula into a Functional of the Electron Density, *Phys. Rev. B*, 37 (1988), pp. 785–789.
  - [36] K. LEJAEGHERE, G. BIHLMAYER, T. BJÖRKMAN, P. BLAHA, S. BLÜGEL, V. BLUM, D. CALISTE, I. E. CASTELLI, S. J. CLARK, A. DAL CORSO, S. DE GIRONCOLI, T. DEUTSCH, J. K. DEWHURST, I. DI MARCO, C. DRAXL, M. DULAK, O. ERIKSSON, J. A. FLORES-LIVAS, K. F. GARRITY, L. GENOVESE, P. GIANNOZZI, M. GIANTOMASSI, S. GOEDECKER, X. GONZE, O. GRÄNÄS, E. K. U. GROSS, A. GULANS, F. GYGI, D. R. HAMANN, P. J. HASNIP, N. A. W. HOLZWARTH, D. IUŞAN, D. B. JOCHYM, F. JOLLET, D. JONES, G. KRESSE, K. KOEPERNIK, E. KÜÇÜKBENLİ, Y. O. KVASHNIN, I. L. M. LOCHT, S. LUBECK, M. MARSMAN, N. MARZARI, U. NITZSCHE, L. NORDSTRÖM, T. OZAKI, L. PAULATTO, C. J. PICKARD, W. POELMANS, M. I. J. PROBERT, K. REFSO, M. RICHTER, G.-M. RIGNANESE, S. SAHA, M. SCHEFFLER, M. SCHLIPF, K. SCHWARZ, S. SHARMA, F. TAVAZZA, P. THUNSTRÖM, A. TKATCHENKO, M. TORRENT, D. VANDERBILT, M. J. VAN SETTEN, V. VAN SPEYBROECK, J. M. WILLS, J. R. YATES, G.-X. ZHANG, AND S. COTTENIER, Reproducibility in Density Functional Theory Calculations of Solids, *Science*, 351 (2016).
  - [37] K. LEJAEGHERE, V. V. SPEYBROECK, G. V. OOST, AND S. COTTENIER, Error Estimates for Solid-State Density-Functional Theory Predictions: An Overview by Means of the Ground-State Elemental Crystals, *Critical Reviews in Solid State and Materials Sciences*, 39 (2014), pp. 1–24.
  - [38] O. LEVY, G. L. W. HART, AND S. CURTAROLO, Structure Maps for HCP Metals from First-Principles Calculations, *Phys. Rev. B*, 81 (2010), p. 174106.
  - [39] S. B. MAISEL, M. HÖFLER, AND S. MÜLLER, Configurationally Exhaustive First-Principles Study of a Quaternary Superalloy with a Vast Configuration Space, *Phys. Rev. B*, 94 (2016), p. 014116.
  - [40] A. E. MATTSSON AND R. ARMIENTO, Implementing and Testing the AM05 Spin Density Functional, *Phys. Rev. B*, 79 (2009), p. 155101.
  - [41] A. E. MATTSSON, R. ARMIENTO, J. PAIER, G. KRESSE, J. M. WILLS, AND T. R. MATTSSON, The AM05 Density Functional Applied to Solids, *The Journal of Chemical Physics*, 128 (2008).
  - [42] A. E. MATTSSON, P. A. SCHULTZ, M. P. DESJARLAIS, T. R. MATTSSON, AND K. LEUNG, Designing Meaningful Density Functional Theory Calculations in Materials Sciencea Primer, *Modelling and Simulation in Materials Science and Engineering*, 13 (2005), p. R1.
  - [43] A. E. MATTSSON AND J. M. WILLS, Density Functional Theory for d- and f-Electron Materials and Compounds, *International Journal of Quantum Chemistry*, 116 (2016), pp. 834–846.
  - [44] M. METHFESSEL AND A. T. PAXTON, High-Precision Sampling for Brillouin-Zone Integration in Metals, *Phys. Rev. B*, 40 (1989), pp. 3616–3621.
  - [45] H. J. MONKHORST AND J. D. PACK, Special Points for Brillouin-Zone Integrations, *Phys. Rev. B*, 13 (1976), pp. 5188–5192.
  - [46] N. NAKAJIMA, T. KOIDE, T. SHIDARA, H. MIYAUCHI, H. FUKUTANI, A. FUJIMORI, K. IIO, T. KATAYAMA, M. NÝVL, AND Y. SUZUKI, Perpendicular Magnetic Anisotropy Caused by Interfacial Hybridization via Enhanced Orbital Moment in Co/Pt Multilayers: Magnetic Circular X-Ray Dichroism Study, *Phys. Rev. Lett.*, 81 (1998), pp. 5229–5232.
  - [47] L. NORDSTRÖM, J. M. WILLS, P. H. ANDERSSON, P. SÖDERLIND, AND O. ERIKSSON, Spin-Orbit Coupling in the Actinide Elements: A Critical Evaluation of Theoretical Equilibrium Volumes, *Phys. Rev. B*, 63 (2000), p. 035103.
  - [48] V. OZOLIŅŠ, C. WOLVERTON, AND A. ZUNGER, Cu-Au, Ag-Au, Cu-Ag, and Ni-Au Intermetallics: First-Principles Study of Temperature-Composition Phase Diagrams and Structures, *Phys. Rev. B*, 57



- (1998), pp. 6427–6443.
- [49] J. P. PERDEW, K. BURKE, AND M. ERNZERHOF, Generalized Gradient Approximation Made Simple, *Phys. Rev. Lett.*, 77 (1996), pp. 3865–3868.
  - [50] J. P. PERDEW, K. BURKE, AND M. ERNZERHOF, Generalized Gradient Approximation Made Simple [Phys. Rev. Lett. 77, 3865 (1996)], *Phys. Rev. Lett.*, 78 (1997), pp. 1396–1396.
  - [51] J. P. PERDEW, J. A. CHEVARY, V. S. H., K. A. JACKSON, M. R. PEDERSON, D. J. SINGH, AND C. FIOHAIS, Atoms, Molecules, Solids, and Surfaces: Applications of the Generalized Gradient Approximation for Exchange and Correlation, *Phys. Rev. B*, 46 (1992), pp. 6671–6687.
  - [52] J. P. PERDEW, J. A. CHEVARY, S. H. VOSKO, K. A. JACKSON, M. R. PEDERSON, D. J. SINGH, AND C. FIOHAIS, Erratum: Atoms, Molecules, Solids, and Surfaces: Applications of the Generalized Gradient Approximation for Exchange and Correlation, *Phys. Rev. B*, 48 (1993), pp. 4978–4978.
  - [53] J. P. PERDEW, A. RUZSINSZKY, G. I. CSONKA, L. A. CONSTANTIN, AND J. SUN, Workhorse Semilocal Density Functional for Condensed Matter Physics and Quantum Chemistry, *Phys. Rev. Lett.*, 103 (2009), p. 026403.
  - [54] J. P. PERDEW, A. RUZSINSZKY, G. I. CSONKA, O. A. VYDROV, G. E. SCUSERIA, L. A. CONSTANTIN, X. ZHOU, AND K. BURKE, Restoring the Density-Gradient Expansion for Exchange in Solids and Surfaces, *Phys. Rev. Lett.*, 100 (2008), p. 136406.
  - [55] J. P. PERDEW AND K. SCHMIDT, Jacobs Ladder of Density Functional Approximations for the Exchange-Correlation Energy, *AIP Conference Proceedings*, 577 (2001), pp. 1–20.
  - [56] J. P. PERDEW AND A. ZUNGER, Self-Interaction Correction to Density-Functional Approximations for Many-Electron Systems, *Phys. Rev. B*, 23 (1981), pp. 5048–5079.
  - [57] B. PUCHALA, J. C. THOMAS, J. GOIRI, M. RADIN, N. S. H. GUNDA, A. R. NATARAJAN, AND L. DECOLVENAERE, *CASMcode: v0.2.0*, Aug. 2016.
  - [58] B. PUCHALA AND A. VAN DER VEN, Thermodynamics of the Zr-O System from First-Principles Calculations, *Phys. Rev. B*, 88 (2013), p. 094108.
  - [59] P. PYYKKO, Relativistic Effects in Structural Chemistry, *Chemical Reviews*, 88 (1988), pp. 563–594.
  - [60] J. E. SAAL, S. KIRKLIN, M. AYKOL, B. MEREDIG, AND C. WOLVERTON, Materials Design and Discovery with High-Throughput Density Functional Theory: The Open Quantum Materials Database (OQMD), *JOM*, 65 (2013), pp. 1501–1509.
  - [61] S. K. SIDOROV AND A. V. DOROSHENKO, On the Magnetic Structure of Some Alloys of Transition Metals, *physica status solidi (b)*, 16 (1966), pp. 737–744.
  - [62] A. SOROURI, W. M. C. FOULKES, AND N. D. M. HINE, Accurate and Efficient Method for the Treatment of Exchange in a Plane-Wave Basis, *The Journal of Chemical Physics*, 124 (2006).
  - [63] J. TAO, J. P. PERDEW, V. N. STAROVEROV, AND G. E. SCUSERIA, Climbing the Density Functional Ladder: Nonempirical Meta-Generalized Gradient Approximation Designed for Molecules and Solids, *Phys. Rev. Lett.*, 91 (2003), p. 146401.
  - [64] J. C. THOMAS AND A. V. D. VEN, Finite-Temperature Properties of Strongly Anharmonic and Mechanically Unstable Crystal Phases from First Principles, *Phys. Rev. B*, 88 (2013), p. 214111.
  - [65] M. C. TROPAREVSKY, J. R. MORRIS, P. R. C. KENT, A. R. LUPINI, AND G. M. STOCKS, Criteria for Predicting the Formation of Single-Phase High-Entropy Alloys, *Phys. Rev. X*, 5 (2015), p. 011041.
  - [66] J. H. VAN VLECK, On the Anisotropy of Cubic Ferromagnetic Crystals, *Phys. Rev.*, 52 (1937), pp. 1178–1198.
  - [67] NEW RELEASE: PAW datasets v.54!
  - [68] P. VILLARS, M. BERNDT, K. BRANDENBURG, K. CENZUAL, J. DAAMS, F. HULLIGER, T. MASSALSKI, H. OKAMOTO, K. OSAKI, A. PRINCE, H. PUTZ, AND S. IWATA, The Pauling File, Binaries Edition, *Journal of Alloys and Compounds*, 367 (2004), pp. 293 – 297.
  - [69] P. VILLARS, H. OKAMOTO, AND K. CENZUAL, ASM Alloy Phase Diagram Database.
  - [70] S. H. VOSKO, L. WILK, AND M. NUSAIR, Accurate Spin-Dependent Electron Liquid Correlation Energies for Local Spin Density Calculations: a Critical Analysis, *Canadian Journal of Physics*, 58 (1980), pp. 1200–1211.
  - [71] Y. ZHAO AND D. G. TRUHLAR, A New Local Density Functional for Main-Group Thermochemistry, Transition Metal Bonding, Thermochemical Kinetics, and Noncovalent Interactions, *The Journal of Chemical Physics*, 125 (2006).
  - [72] Y. ZHAO AND D. G. TRUHLAR, The M06 Suite of Density Functionals for Main Group Thermochemistry, Thermochemical Kinetics, Noncovalent Interactions, Excited States, and Transition Elements: Two New Functionals and Systematic Testing of Four M06-Class Functionals and 12 other Functionals, *Theoretical Chemistry Accounts*, 120 (2008), pp. 215–241.

## AN EXACT SOLUTION AND SENSITIVITY COMPUTATIONS FOR A FORCED BURGER'S EQUATION

KOA L. FISHER<sup>§</sup> AND ALLEN C. ROBINSON<sup>¶</sup>

**Abstract.** Advanced numerical methods and software systems for partial differential equations allow for both solution approximation and efficient computation of integral quantities of interest and sensitivities of these quantities of interest with respect to model parameters. It is useful to test the accuracy and correctness of such solution methods for nonlinear hyperbolic systems that may form shocks. To this end, an exact analytic solution to a forced Burger's equation is developed using the methods of characteristics, quantities of interest are developed, and the values and sensitivities are compared with output from the Drekar code.

**1. Introduction.** Drekar is research code framework for developing and evaluating numerical discretizations of multiphysics partial differential equation operators using implicit and explicit operators and embedded adjoints for computing sensitivities to equation parameters [1, 2]. We are interested in verifying the Drekar methodology for solving nonlinear hyperbolic problems with parameters for both linear and nonlinear quantities of interest over domains. The equation that we use for a test problem is the forced Burger's equation,

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = s. \quad (1.1)$$

for  $u(x, t; s)$ . We also assume the following conservative form,

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x} \left( \frac{u^2}{2} \right) = s. \quad (1.2)$$

We solve this equation for a simple initial condition using the method of characteristics with shock fitting. We then analyze five different quantities of interest representing different levels of challenge for the numerical method. The different quantities of interest are integrals of  $u$  and  $u^2$  over a two-dimensional domain of space-time. One of the domains crosses two different characteristic regions, one crosses a shock and the three remaining domains are contained completely within a single characteristic region. Although the solution is exact and the quantities of interest can be represented exactly, we evaluate the solution and quantities of interest numerically. Finally, we compare the computed values and observed rates of convergence for each of the quantities of interest.

### 2. An Exact Solution of the Forced Burger's Equation.

**2.1. Method of Characteristics.** Integration along characteristic curves allows for development of exact solutions of one dimensional nonlinear hyperbolic conservation laws. A good reference for how to solve nonlinear hyperbolic equations using the method of characteristics is the book by Whitham [3]. Equation 1.1 can be rewritten along characteristics as

$$\frac{du}{dt} = s \text{ on } \frac{dx}{dt} = u \quad (2.1)$$

with solution

$$u = st + f(\xi), \quad (2.2)$$

---

<sup>§</sup>Sandia National Laboratories, kfisher@sandia.gov

<sup>¶</sup>Sandia National Laboratories, acrobin@sandia.gov

along the characteristics

$$x = \frac{1}{2}st^2 + f(\xi)t + \xi \quad (2.3)$$

parametrized by the initial location,  $\xi$ , at time  $t = 0$ . We chose initial conditions that specify  $f(\xi)$  as

$$f(\xi) = \begin{cases} 0 & \text{if } \xi < -1, \\ (1 + \xi) & \text{if } -1 \leq \xi < 0, \\ (1 - \xi) & \text{if } 0 \leq \xi < 1, \\ 0 & \text{if } 1 \leq \xi \end{cases} \quad (2.4)$$

This allows  $x$  to vary in different regions corresponding to initial values,  $\xi$ , as  $t$  changes. It is in these four characteristic regions that we are interested in looking at the quantities of interest. In Figure 2.1 we plot the different curves of  $x$  as  $t$  changes for varying values of  $s$  and  $\xi$ .

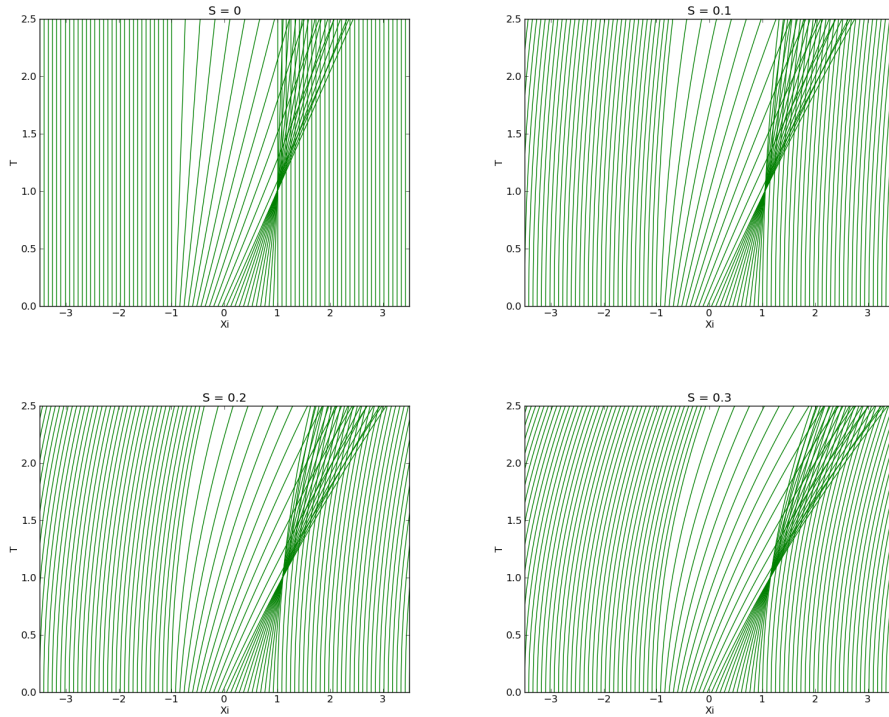


Fig. 2.1: Characteristics curves before shock fitting.

**2.2. The Shock Curve.** Where the characteristics overlap, we need to fit a shock. We want to compute this shock curve analytically so that we will be able to compare the exact solution to verify the solutions computed using Drekar. To find an equation for the shock curve, we use the conservative form Equation 1.2 to develop the shock jump conditions:

$$-\dot{x}_s[u] + \left[\frac{1}{2}u^2\right] = 0, \quad (2.5)$$

which means

$$-\dot{x}_s(u^+ - u^-) + \left(\frac{1}{2}u^{+2} - \frac{1}{2}u^{-2}\right) = 0. \quad (2.6)$$

The "plus" indicates the right hand side of the shock and the "minus" indicates the left hand side. The shock jump equation simplifies to

$$-\dot{x}_s + \frac{1}{2}(u^+ + u^-) = 0. \quad (2.7)$$

All of the characteristics in  $0 < \xi < 1$  break at  $t = 1$ . Thus the shock will be composed of (left) characteristics from the  $-1 < \xi < 0$  region interacting with (right) characteristics from the  $\xi > 1$  region. The shock starts at  $t = 1$ ,  $x = \frac{s}{2} + 1$ . We have the characteristic equations for the shock from the left and right

$$x_s = \frac{st^2}{2} + (1 + \xi_-)t + \xi_-; \quad u_- = st + (1 + \xi_-) \quad (2.8)$$

and

$$x_s = \frac{st^2}{2} + \xi_+; \quad u_+ = st. \quad (2.9)$$

Substituting  $u^+$  and  $u^-$  into the shock jump equation and solving for  $\xi_-$ , we get

$$\dot{x}_s = \frac{1}{2}(2st + (1 + \xi_-)) = \frac{1}{2}\left(2st + \left(1 + \frac{x_s - \frac{st^2}{2} - t}{1 + t}\right)\right). \quad (2.10)$$

Moving the  $x_s$  term to the other side, we end up with a differential equation,

$$\dot{x}_s - \frac{1}{2(1+t)}x_s = st + \frac{1}{2} - \frac{st^2}{4(1+t)} - \frac{t}{2(t+1)} = \frac{3s}{4}(1+t) - \frac{s}{2} + \frac{2-s}{4} \frac{1}{1+t} \quad (2.11)$$

Then we find an integrating factor,  $(1+t)^{-\frac{1}{2}}$ , for this equation and obtain the solution

$$x_s = \frac{1}{2}st^2 - 1 + c(1+t)^{\frac{1}{2}}. \quad (2.12)$$

The initial conditions,  $t = 1$ ,  $x = \frac{s}{2} + 1$ , imply that the appropriate particular solution is

$$x_s = \frac{1}{2}st^2 - 1 + \sqrt{2}(1+t)^{\frac{1}{2}}. \quad (2.13)$$

The characteristic curves with the fitted shock are shown in Figure 2.2 for various values of  $s$ .

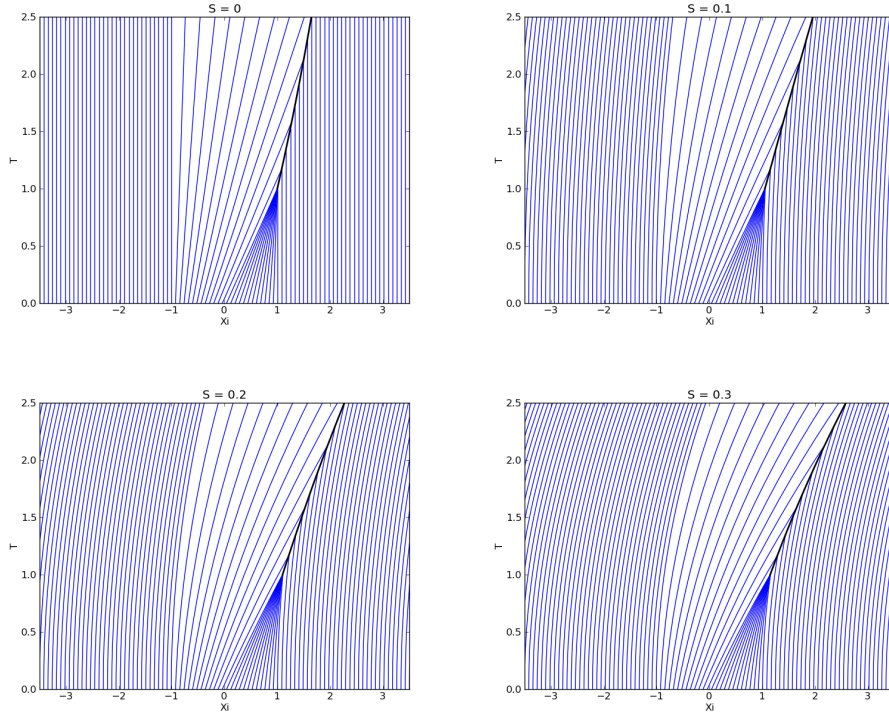


Fig. 2.2: Characteristic curves after shock fitting.

**3. Quantities of Interest.** We are interested in computing scalar quantities of interest,  $Q(s; g(u))$  defined by space-time integrals of  $g(u) = u$  and the nonlinear  $g(u) = u^2$  over specified domains of interest in order to fully test the Drekar methodology. Both smooth single region domains and domains that cross characteristic regions and cross the shock should be included. We have not included a domain that includes the shock breaking point in space-time, although this could be done. We choose specific rectangular domains so that the numerical integration methodology of the exact solution for all values of  $s$  need not be specialized.

**3.1. Integration Methodology.** Numerical computation of the quantities of interest is performed using a two-dimensional nested trapezoidal rule methodology. There are five different domains of interest. The first domain crosses from one region to another where the characteristic curves are different. The second domain crosses the shock. The third, fourth, and fifth domains are completely contained within one characteristic region. We only look at three domains fully contained within a single characteristic region because the first and the fourth characteristic regions have exactly the same properties.

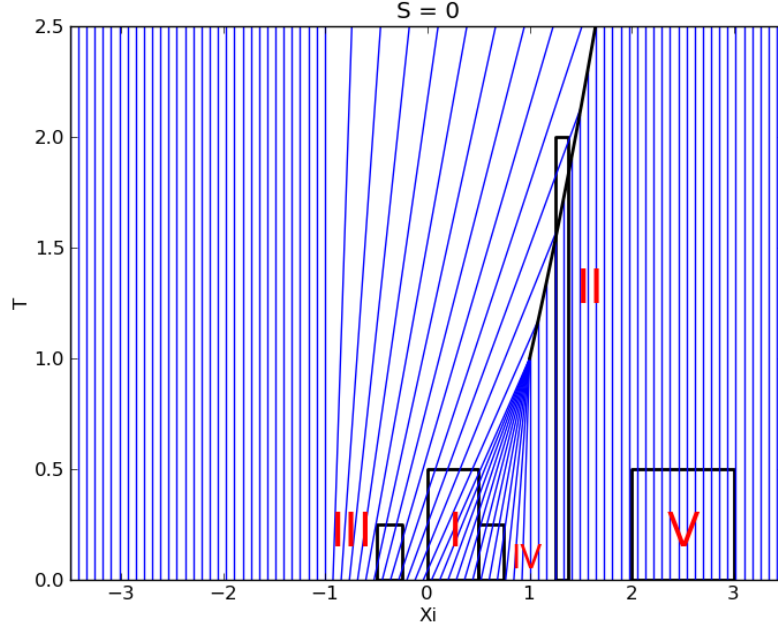


Fig. 3.1: The shock curve and characteristic curves with boxed off regions of interest.

We have constructed the domains so that throughout the range of  $s$  considered, we only need to split the domains into a maximum of two different sub-domains for integration purposes.

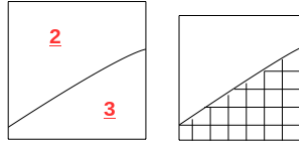


Fig. 3.2: The box on left shows how a domain may be split in two by either a bounding characteristic or a shock,  $t(x, s)$ . The box on the right shows how the two-dimensional integration is done in the  $t$ -direction and then the  $x$ -direction.

We numerically evaluate a line integral in the  $t$  direction using the trapezoidal rule and then sum the results in the  $x$  direction again using the trapezoidal rule. For the first two domains of integration it is necessary to divide the domain into two sub-domains. The first domain is split by a bounding characteristics and the second is split by the shock. In order to find the  $t(x, s)$  integration bound we can use the exact solution for the bounding characteristic and Newton's method for the shock.

**3.2. Exact Solution for  $Q_i(s; u)$ .** We now present the representation of the fundamental quantities of interest over the different domains of interest.

I. The first domain we want to integrate over is  $t \in [0, \frac{1}{2}]$ ,  $x \in [0, \frac{1}{2}]$ ,

$$\begin{aligned}
Q_I(s; u) &= \int_0^{\frac{1}{2}} \int_0^{\frac{1}{2}} u \, dx \, dt = \int_0^{\frac{1}{2}} \int_0^{\frac{1}{2}} st + f(\xi) \, dx \, dt = \\
&\int_0^{\frac{1}{2}} \left( \int_0^{t(x,s)} st + (1 - \xi) \, dt + \int_{t(x,s)}^{\frac{1}{2}} st + (1 + \xi) \, dt \right) dx = \\
&\int_0^{\frac{1}{2}} \left( \int_0^{t(x,s)} st + \left(1 - \frac{x - \frac{1}{2}st^2 - t}{(1 - t)}\right) dt + \int_{t(x,s)}^{\frac{1}{2}} st + \left(1 + \frac{x - \frac{1}{2}st^2 - t}{(1 + t)}\right) dt \right) dx. \quad (3.1)
\end{aligned}$$

This domain is split by two characteristic regions. The bounding characteristic curve can be obtained analytically as

$$t(x, s) = \frac{2x}{1 + \sqrt{1 + 2sx}} \quad (3.2)$$

II. The next domain is on  $t \in [0, 2]$ ,  $x \in [\frac{5}{4}, \frac{11}{8}]$ ,

$$\begin{aligned}
Q_{II}(s; u) &= \int_{\frac{5}{4}}^{\frac{11}{8}} \int_0^2 u \, dt \, dx = \int_{\frac{5}{4}}^{\frac{11}{8}} \int_0^2 st + f(\xi) \, dt \, dx = \\
&\int_{\frac{5}{4}}^{\frac{11}{8}} \left( \int_0^{t(x,s)} st \, dt + \int_{t(x,s)}^2 st + (1 + \xi) \, dt \right) dx = \\
&\int_{\frac{5}{4}}^{\frac{11}{8}} \left( \int_0^{t(x,s)} st \, dt + \int_{t(x,s)}^2 st + \left(1 + \frac{x - \frac{1}{2}st^2 - t}{(1 + t)}\right) dt \right) dx. \quad (3.3)
\end{aligned}$$

This domain was chosen because it crosses the shock curve defined by Equation 2.13. The bounding  $t(x, s)$  curve defined implicitly by Equation 2.13 is evaluated using Newton's method.

III. Then we have  $t \in [0, \frac{1}{4}]$ ,  $x \in [-\frac{1}{2}, -\frac{1}{4}]$ ,

$$\begin{aligned}
Q_{III}(s; u) &= \int_0^{\frac{1}{4}} \int_{-\frac{1}{2}}^{-\frac{1}{4}} u \, dx \, dt = \int_0^{\frac{1}{4}} \int_{-\frac{1}{2}}^{-\frac{1}{4}} st + f(\xi) \, dx \, dt = \\
&\int_0^{\frac{1}{4}} \int_{-\frac{1}{2}}^{-\frac{1}{4}} st + \left(1 + \frac{x - \frac{1}{2}st^2 - t}{(1 + t)}\right) dx \, dt. \quad (3.4)
\end{aligned}$$

IV. Followed by  $t \in [0, \frac{1}{4}]$ ,  $x \in [\frac{1}{2}, \frac{3}{4}]$ ,

$$Q_{IV}(s; u) = \int_{\frac{1}{2}}^{\frac{3}{4}} \int_0^{\frac{1}{4}} u \, dt \, dx = \int_{\frac{1}{2}}^{\frac{3}{4}} \int_0^{\frac{1}{4}} st + f(\xi) \, dt \, dx =$$

$$\int_{\frac{1}{2}}^{\frac{3}{4}} \int_0^{\frac{1}{4}} st + \left(1 - \frac{x - \frac{1}{2}st^2 - t}{(1-t)}\right) dt dx. \quad (3.5)$$

V. Finally  $t \in [0, \frac{1}{2}]$ ,  $x \in [2, 3]$ ,

$$Q_V(s; u) = \int_2^3 \int_0^{\frac{1}{2}} u dt dx = \int_2^3 \int_0^{\frac{1}{2}} st dt dx = \frac{s}{8}. \quad (3.6)$$

The domains III, IV, and V were chosen because they are contained completely within characteristic regions with distinct properties. The solution over the last three domains is smoother and we might expect rates of convergence for the quantities of interest to be higher. Similar integrals can be written for  $Q_i(s; u^2)$  in an obvious way. Using the trapezoidal integration method described above we computed the domain integrals of  $u$  and  $u^2$ . These results are shown in Tables 3.1 and 3.3.

**3.3. Numerical Results for  $Q_i(s; u)$  and  $Q_i(s; u^2)$ .** We implemented this problem using Drekar and computed  $Q_i(s; u)$  and  $Q_i(s; u^2)$  numerically for the five chosen domains. The results are shown along with the computed rate of convergence, in parentheses, with respect to uniform refinement in space and time. Convergence rates are computed from successive errors and are only computed if both errors are larger than  $10^{-6}$ . If this is not the case, then a  $(-)$  is shown. Negative convergence rates imply that the results are not yet in the asymptotic region. We only integrate to the end of the time interval specified by the domain specific quantity of interest because Drekar does not support multiple quantities of interest in a given simulation. The x-domain is  $[-4, 4]$  with 100, 200, 400, then 800 intervals in the x-direction and in time we decrease our time step by half giving us time steps of 1.25e-2, 6.25e-3, 3.125e-3, and 1.5625e-3, corresponding to uniform refinement in space and time. The forward solution was solved using an implicit backward difference time integrator (BDF2) and the adjoint solution described in the next section was solved using a BDF1 time integrator with a different time step. A stabilized finite element methodology with entropy shock capturing was utilized for the forward problem.

Table 3.2 gives values  $Q_i(s; u)$  for four refinement levels along with computed convergence rates relative to the exact solution shown in Table 3.1.

Exact $Q_i(s; u)$						
		I	II	III	IV	V
$s$	0	0.210624	0.033201	0.034866	0.026970	0.000000
	0.1	0.216675	0.080950	0.035593	0.027832	0.012500
	0.2	0.222693	0.118001	0.036319	0.028693	0.025000
	0.3	0.228681	0.150004	0.037045	0.029555	0.037500

Table 3.1: Exact solution for  $Q_i(s; u)$  for different domains and values of  $s$

$Q_i(s; u)$ 100						
		I	II	III	IV	V
$s$	0.0	0.216786	0.034632	0.038707	0.024779	0.000000
	0.1	0.223200	0.079608	0.039520	0.025607	0.012500



	0.2	0.229578	0.114574	0.040333	0.026435	0.025000
	0.3	0.235923	0.145112	0.041146	0.027264	0.037500

$Q_i(s; u)$ 200						
		I	II	III	IV	V
$s$	0.0	0.210357 (4.53)	0.030028 (-1.15)	0.033180 (1.19)	0.026197 (1.50)	0.000000 (-)
	0.1	0.216433 (4.75)	0.076459 (-1.74)	0.033877 (1.20)	0.027025 (1.46)	0.012500 (-)
	0.2	0.222475 (4.98)	0.112313 (-0.73)	0.034574 (1.20)	0.027852 (1.42)	0.025000 (-)
	0.3	0.228484 (5.20)	0.143177 (-0.48)	0.035272 (1.21)	0.028680 (1.39)	0.037500 (-)

$Q_i(s; u)$ 400						
		I	II	III	IV	V
$s$	0.0	0.210491 (1.01)	0.032343 (1.89)	0.034852 (6.89)	0.026952 (5.38)	-0.000000 (-)
	0.1	0.216554 (0.99)	0.078047 (0.63)	0.035578 (6.92)	0.027813 (5.46)	0.012500 (-)
	0.2	0.222582 (0.97)	0.113543 (0.35)	0.036305 (6.95)	0.028675 (5.53)	0.025000 (-)
	0.3	0.228578 (0.93)	0.144226 (0.24)	0.037031 (6.97)	0.029537 (5.60)	0.037500 (-)

$Q_i(s; u)$ 800						
		I	II	III	IV	V
$s$	0.0	0.210561 (1.06)	0.033150 (4.08)	0.034860 (1.18)	0.026960 (0.83)	0.000000 (-)
	0.1	0.216616 (1.04)	0.080919 (6.57)	0.035586 (1.19)	0.027822 (0.83)	0.012500 (-)
	0.2	0.222637 (0.99)	0.117975 (7.44)	0.036313 (1.20)	0.028683 (0.83)	0.025000 (-)
	0.3	0.228628 (0.95)	0.149981 (7.98)	0.037039 (1.21)	0.029545 (0.84)	0.037500 (-)

Table 3.2: Drekar solution values of  $Q_i(s; u)$  (and associated convergence rates) for different domains and values of  $s$  for a sequence of meshes.

Similarly, Table 3.4 gives values of  $Q_i(s; u^2)$  for four refinement levels along with computed convergence rates relative to the exact solution shown in Table 3.3.

Exact $Q_i(s; u^2)$						
		I	II	III	IV	V
$s$	0	0.180556	0.027127	0.019792	0.012153	0.000000
	0.1	0.191128	0.067335	0.020584	0.012950	0.000417
	0.2	0.202014	0.110877	0.021398	0.013780	0.001667
	0.3	0.213212	0.157752	0.022235	0.014643	0.003750

Table 3.3: Exact solution for  $Q_i(s; u^2)$  for different domains and values of  $s$

$Q_i(s; u^2) 100$						
		I	II	III	IV	V
$s$	0.0	0.184168	0.024256	0.021853	0.010665	0.000000
	0.1	0.195293	0.063656	0.022732	0.011399	0.000417
	0.2	0.206762	0.105418	0.023635	0.012164	0.001667
	0.3	0.218570	0.150553	0.024563	0.012960	0.003750

$Q_i(s; u^2) 200$						
		I	II	III	IV	V
$s$	0.0	0.180070 (2.89)	0.022430 (-0.71)	0.018653 (0.86)	0.011895 (2.53)	0.000000 (-)
	0.1	0.190675 (3.20)	0.061685 (-0.62)	0.019407 (0.87)	0.012669 (2.46)	0.000417 (-)
	0.2	0.201596 (3.51)	0.103845 (-0.37)	0.020182 (0.88)	0.013475 (2.40)	0.001667 (-)
	0.3	0.212829 (3.81)	0.149071 (-0.27)	0.020978 (0.89)	0.014312 (2.35)	0.003750 (-)

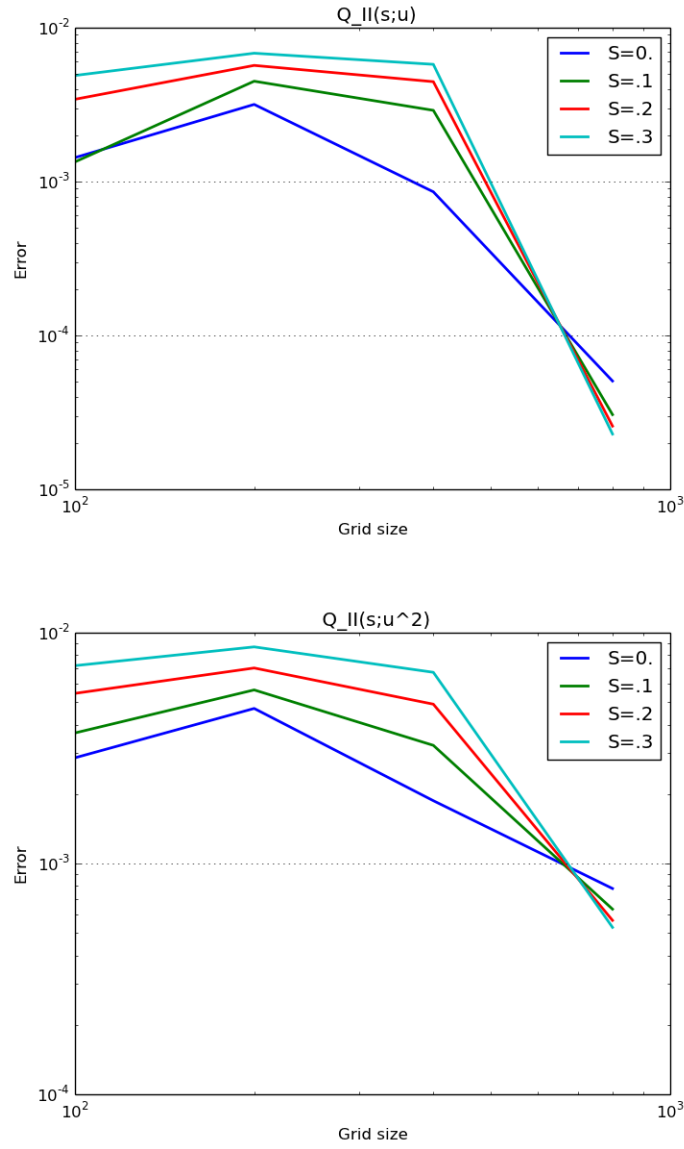
$Q_i(s; u^2) 400$						
		I	II	III	IV	V
$s$	0.0	0.180322 (1.06)	0.025256 (1.33)	0.019776 (6.17)	0.012135 (3.87)	0.000000 (-)
	0.1	0.190909 (1.05)	0.064085 (0.80)	0.020568 (6.19)	0.012932 (3.95)	0.000417 (-)
	0.2	0.201808 (1.02)	0.105975 (0.52)	0.021382 (6.20)	0.013762 (4.03)	0.001667 (-)
	0.3	0.213018 (0.98)	0.151016 (0.37)	0.022218 (6.22)	0.014624 (4.10)	0.003750 (-)

$Q_i(s; u^2) 800$						
		I	II	III	IV	V
$s$	0.0	0.180445 (1.08)	0.026348 (1.26)	0.019785 (1.22)	0.012143 (0.89)	0.000000 (-)
	0.1	0.191022 (1.05)	0.066701 (2.36)	0.020577 (1.23)	0.012940 (0.89)	0.000417 (-)
	0.2	0.201911 (1.00)	0.110310 (3.11)	0.021391 (1.23)	0.013770 (0.90)	0.001667 (-)
	0.3	0.213111 (0.94)	0.157223 (3.67)	0.022228 (1.24)	0.014633 (0.90)	0.003750 (-)

Table 3.4: Drekar solution values of  $Q_i(s; u^2)$  (and associated convergence rates) for different domains and values of  $s$  for a sequence of meshes.

For additional clarity we show a graph of the domain II errors in Figure 3.3. This gives a better visual idea of how the domain II quantities of interest are converging.

**3.4. Exact Solution for  $Q'_i(s; u)$  and  $Q'_i(s; u^2)$ .** We are also interested in the  $s$  derivative of the quantities of interest. For those domains that cross between characteristic regions or across a shock curve we can use Leibniz' integral rule to take the derivative of the integrals with respect to  $s$ . With the bounds for integration being  $a$  and  $b$  in the  $x$  direction,  $c$  and  $d$  in the  $t$  direction we have,

Fig. 3.3: Error plots for  $Q_{II}(s;u)$  and  $Q_{II}(s;u^2)$ 

$$\begin{aligned}
 \frac{d}{ds} \int_a^b \int_c^d u \, dt \, dx &= \int_a^b \int_c^{t(x,s)} \frac{\partial u_i}{\partial s} \, dt \, dx + \int_a^b u_i(x, t(x, s); s) \frac{\partial t}{\partial s}(x, s) \, dx \\
 &+ \int_a^b \int_{t(x,s)}^d \frac{\partial u_j}{\partial s} \, dt \, dx - \int_a^b u_j(x, t(x, s); s) \frac{\partial t}{\partial s}(x, s) \, dx
 \end{aligned} \tag{3.7}$$

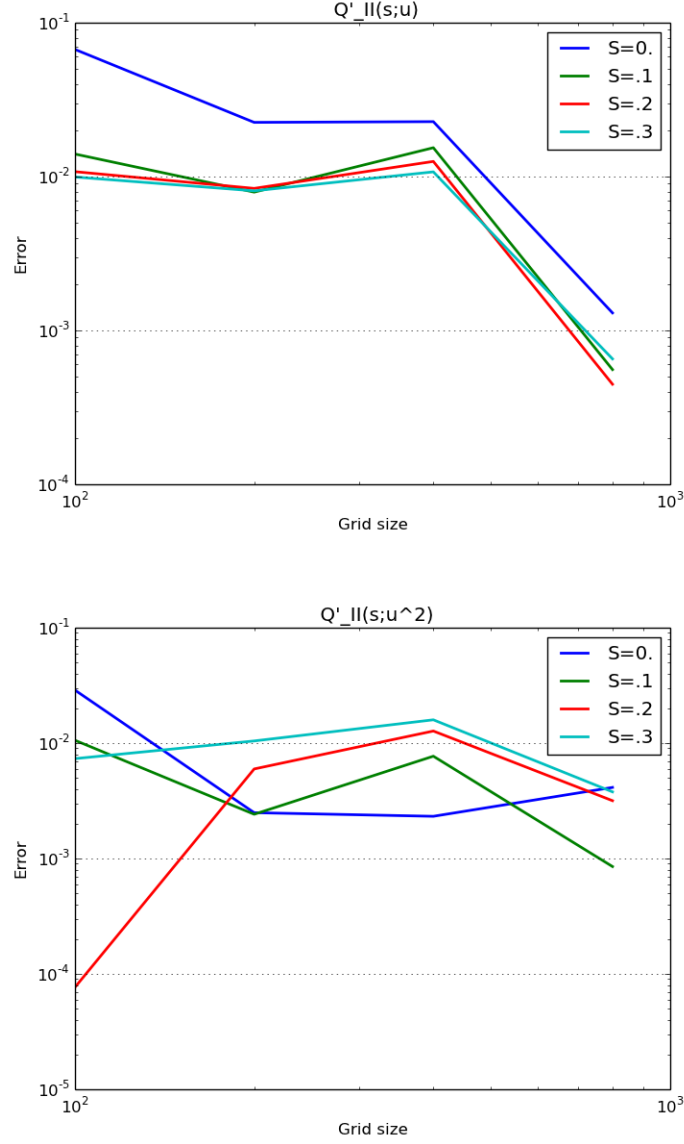


Fig. 3.4: Error plots for  $Q'_{II}(s; u)$  and  $Q'_{II}(s; u^2)$

where  $t(x, s)$  is a representation of the separating curve and the subscripts  $i$  and  $j$  refer to the solution in the lower and upper time regions, respectively, in the  $x-t$  plane. All 4 terms must be calculated for the shock curve of domain II, but the boundary terms need not be calculated in domain I because the solution  $u$  is continuous and these terms cancel.

For those domains that do not cross a separating curve

$$\frac{d}{ds} \int_a^b \int_c^d u \, dt dx = \int_a^b \int_c^d \frac{\partial u}{\partial s} \, dt dx \quad (3.8)$$

The corresponding integrals for  $u^2$  are

$$\begin{aligned} \frac{d}{ds} \int_a^b \int_c^d u^2 dt dx &= \int_a^b \int_c^{t(x,s)} 2u_i \frac{\partial u_i}{\partial s} dt dx + \int_a^b u_i^2(x, t(x, s); s) \frac{\partial t}{\partial s}(x, s) dx \\ &+ \int_a^b \int_{t(x,s)}^d 2u_j \frac{\partial u_j}{\partial s} dt dx - \int_a^b u_j^2(x, t(x, s); s) \frac{\partial t}{\partial s}(x, s) dx \end{aligned} \quad (3.9)$$

for domains I and II, and

$$\frac{d}{ds} \int_a^b \int_c^d u dt dx = \int_a^b \int_c^d 2u \frac{\partial u}{\partial s} dt dx \quad (3.10)$$

for the other domains. Tables 3.5 and 3.7 gives the results of the computation of these quantities of interest from the exact solution.

**3.5. Numerical Results for  $Q'_i(s; u)$  and  $Q'_i(s; u^2)$ .** In addition to the forward solution, we computed  $Q'(s; u)$  and  $Q'_i(s; u^2)$  numerically using the adjoint methodology in Drekar. The computed results for the five quantities of interest are shown along with the convergence rates in parentheses with respect to uniform refinement in space and time.

Table 3.6 gives values  $Q'_i(s; u)$  for four refinement levels along with computed convergence rates relative to the exact solution shown in Table 3.5.

Exact $Q'_i(s; u)$						
		I	II	III	IV	V
$s$	0	0.060696	0.577330	0.007263	0.008617	0.125000
	0.1	0.060338	0.408644	0.007263	0.008617	0.125000
	0.2	0.060023	0.340288	0.007263	0.008617	0.125000
	0.3	0.059743	0.302975	0.007263	0.008617	0.125000

Table 3.5: Exact solution for  $Q'_i(s; u)$  for different domains and values of  $s$

$Q'_i(s; u)$ 100						
		I	II	III	IV	V
$s$	0.0	0.064672	0.510307	0.008092	0.008335	0.125000
	0.1	0.064335	0.394634	0.008091	0.008337	0.125000
	0.2	0.064009	0.329528	0.008091	0.008337	0.125000
	0.3	0.063701	0.293038	0.008091	0.008337	0.125000

$Q'_i(s; u)$ 200						
		I	II	III	IV	V
$s$	0.0	0.062550 (1.10)	0.554827 (1.57)	0.007294 (4.77)	0.008748 (1.10)	0.128145 (-)
	0.1	0.062196 (1.11)	0.400721 (0.82)	0.007295 (4.71)	0.008751 (1.06)	0.128145 (-)
	0.2	0.061889 (1.09)	0.331911 (0.36)	0.007295 (4.71)	0.008751 (1.06)	0.128145 (-)
	0.3	0.061602 (1.09)	0.294899 (0.30)	0.007295 (4.71)	0.008751 (1.06)	0.128145 (-)

$Q'_i(s; u)$ 400						
		I	II	III	IV	V
$s$	0.0	0.061630 (0.99)	0.554581 (-0.02)	0.007429 (-2.45)	0.008861 (-0.90)	0.126567 (1.00)
	0.1	0.061275 (0.99)	0.393244 (-0.96)	0.007431 (-2.40)	0.008864 (-0.88)	0.126567 (1.00)
	0.2	0.060971 (0.98)	0.327746 (-0.58)	0.007431 (-2.40)	0.008864 (-0.88)	0.126567 (1.00)
	0.3	0.060688 (0.98)	0.292244 (-0.41)	0.007431 (-2.40)	0.008864 (-0.88)	0.126567 (1.00)

$Q'_i(s; u)$ 800						
		I	II	III	IV	V
$s$	0.0	0.060823 (2.87)	0.578630 (4.13)	0.007258 (4.93)	0.008627 (4.48)	0.125000 (-)
	0.1	0.060481 (2.71)	0.409201 (4.79)	0.007259 (5.29)	0.008626 (4.69)	0.125000 (-)
	0.2	0.060177 (2.62)	0.340735 (4.81)	0.007259 (5.29)	0.008626 (4.69)	0.125000 (-)
	0.3	0.059897 (2.61)	0.303628 (4.04)	0.007259 (5.29)	0.008626 (4.69)	0.125000 (-)

Table 3.6: Drekar solution values of  $Q'_i(s; u)$  (and associated convergence rates) for different domains and values of  $s$  for a sequence of meshes.

Tables 3.8 gives values  $Q'_i(s; u^2)$  for four refinement levels along with computed convergence rates relative to the exact solution shown in Table 3.7.

Exact $Q'_i(s; u^2)$						
		I	II	III	IV	V
$s$	0	0.104167	0.385417	0.007812	0.007813	0.000000
	0.1	0.107292	0.418750	0.008034	0.008138	0.008333
	0.2	0.110417	0.452083	0.008255	0.008464	0.016667
	0.3	0.113542	0.485417	0.008477	0.008789	0.025000

Table 3.7: Exact solution for  $Q'_i(s; u^2)$  for different domains and values of  $s$

$Q'_i(s; u^2) \cdot 100$						
		I	II	III	IV	V
$s$	0.0	0.110062	0.356649	0.008670	0.007171	0.000000
	0.1	0.113557	0.429323	0.008907	0.007474	0.008176
	0.2	0.116984	0.452160	0.009145	0.007777	0.016352
	0.3	0.120368	0.478082	0.009382	0.008079	0.024527

$$Q'_i(s; u^2) \quad 200$$

		I	II	III	IV	V
$s$	0.0	0.107203 (0.96)	0.382922 (3.53)	0.007770 (4.32)	0.008034 (1.53)	0.000000 (-)
	0.1	0.110489 (0.97)	0.421167 (2.13)	0.007994 (4.46)	0.008370 (1.51)	0.008569 (-0.58)
	0.2	0.113782 (0.96)	0.446114 (-6.28)	0.008218 (4.57)	0.008704 (1.52)	0.017139 (-0.58)
	0.3	0.117037 (0.97)	0.474980 (-0.51)	0.008441 (4.68)	0.009037 (1.52)	0.025708 (-0.58)

$Q'_i(s; u^2) \text{ 400}$						
		I	II	III	IV	V
$s$	0.0	0.105710 (0.98)	0.387738 (0.10)	0.007984 (-2.00)	0.008034 (0.00)	0.000000 (-)
	0.1	0.108917 (0.98)	0.411051 (-1.67)	0.008213 (-2.17)	0.008373 (-0.01)	0.008451 (1.01)
	0.2	0.112139 (0.97)	0.439359 (-1.09)	0.008440 (-2.30)	0.008709 (-0.03)	0.016902 (1.01)
	0.3	0.115331 (0.97)	0.469501 (-0.61)	0.008667 (-2.42)	0.009045 (-0.05)	0.025353 (1.01)

$Q'_i(s; u^2) \text{ 800}$						
		I	II	III	IV	V
$s$	0.0	0.104380 (2.86)	0.389547 (-0.83)	0.007810 (6.19)	0.007812 (-)	0.000000 (-)
	0.1	0.107541 (2.70)	0.417899 (3.18)	0.008032 (6.27)	0.008134 (5.88)	0.008314 (2.59)
	0.2	0.110699 (2.61)	0.448915 (2.01)	0.008252 (5.75)	0.008458 (5.56)	0.016628 (2.59)
	0.3	0.113833 (2.62)	0.481636 (2.07)	0.008472 (5.39)	0.008783 (5.31)	0.024941 (2.59)

Table 3.8: Drekar solution values of  $Q'_i(s; u^2)$  (and associated convergence rates) for different domains and values of  $s$  for a sequence of meshes.

We show a graph of domain II derivative errors in Figure 3.4. The nonlinear quantity of interest,  $Q'_{II}(s; u^2)$  is not yet exhibiting a clear asymptotic convergence response. Additional investigation is needed to study the convergence of the adjoint based derivative computations.

**4. Conclusions.** We have demonstrated an exact solution to a forced Burger's equation and computed numerically integral quantities of interest of the exact solution,  $u$ , and the nonlinear,  $u^2$ , over space-time domains and compared these results with a numerical solution using Drekar. Derivatives of the quantities of interest with respect to the forcing parameter,  $s$ , are also computed from the exact solution and compared with adjoint computations of these derivatives in Drekar. Five different domains were analyzed. Once the discretization is fine enough to approach the asymptotic region we generally observe at least first order convergence for all domains. Domain V converges so rapidly to the exact answer that rates easily become uncomputable. The results obtained show promise that the combined shock-capturing and adjoint derivative capability found in Drekar can effectively compute linear and nonlinear integral domain quantities of interest and sensitivities. However, additional work is still needed to investigate the quality of the Drekar solution and the associated adjoint derivative solution and to understand the precise nature of the numerical convergence rates for this verification problem.

**Acknowledgments.** The authors thank Timothy Wildey for his help with running Drekar and a thoughtful review of this work.

## REFERENCES

- [1] J. N. SHADID, R. P. PAWLOWSKI, E. C. CYR, R. S. TUMINARO, L. CHACON, AND P. D. WEBER, Scalable Implicit Incompressible Resistive MHD with Stabilized FE and Fully-coupled Newton-Krylov-AMG, *Comput. Methods Appl. Mech. Engrg.*, 304 (2016), pp. 1–25.
- [2] J. N. SHADID, T. M. SMITH, E. C. CYR, T. M. WILDEY, AND R. P. PAWLOWSKI, Stabilized FE Simulation of Prototype Thermal-Hydraulics Problems with Integrated Adjoint-based Capabilities, *J. Computational Physics*, 321 (2016), pp. 321–341.
- [3] G. B. WHITHAM, *Linear and Nonlinear Waves*, John Wiley & Sons, 1974.



# A COMPARATIVE STUDY OF SIMPLIFIED MODELS FOR ION CROWDING AND CORRELATION EFFECTS IN ELECTROLYTE SOLUTIONS

BRIAN D. HONG\* AND MAURO PEREGO†

**Abstract.** We study phenomenological models for ion crowding and correlations in the context of ionic fluids. The reduced models are compared to classical density functional theory to examine their ability to model the layer of ions near a charged surface. We find that for a 1:1 electrolyte, the modified Poisson Boltzmann model provides an advantage over the standard Poisson Boltzmann model by describing crowding effects which are not taken into account in the standard model. However, in the same case, it is unclear that the addition of a biharmonic operator to model ion correlations yields any improvement. We also study the effects of increasing ion diameter on the mobility of a colloid in a nanochannel. This study suggests that increases in ion diameter tend to, in general, increase the mobility of the colloid.

**1. Introduction.** The structure of an ionic fluid near a charged surface can be modeled at the nanoscale by a variety of representations. One more accurate method for describing the system is classical density functional theory (cDFT). This method utilizes an ab-initio approach combined with additional assumptions, which make cDFT computationally feasible for relatively small problems. On the opposite end of the spectrum, the Poisson Boltzmann representation of the electrostatic potential assumes that all ions are point charges and therefore greatly reduces the computational cost. However, particularly at high concentrations, assuming the ions are point charges neglects important ion-ion interactions. In order to reintroduce these effects with relatively little additional computational cost, phenomenological models have been proposed. Here we review two such models: the modified Poisson Boltzmann (MPB) model [2] for ion crowding and a Biharmonic version of the MPB model [9] which accounts for ion-ion correlations.

In the first part of the paper we introduce these models and briefly discuss their origin. Subsequently, we discuss their numerical implementation. Next, in order to assess the physical fidelity of the models, we present a comparison of the models with increasing ion size and correlation length. In the last part, we apply the phenomenological model to measure the mobility of a particle in a nanochannel.

**2. Physical Models.** In all models considered in this report the electric potential  $\psi$  is governed by the Poisson equation:

$$-\nabla \cdot (\epsilon_0 \epsilon \nabla \psi) = q, \text{ in } \Omega. \quad (2.1)$$

In this paper the domain  $\Omega$  is typically taken to be a channel in  $\mathbb{R}^2$  or  $\mathbb{R}^3$  that is periodic in the  $z$ -direction. Here  $\epsilon_0$  is the permittivity of free space,  $\epsilon$  is the relative permittivity of the medium, and  $q$  is the free charge density of the fluid. If we consider a fluid medium with  $\alpha = 0, 1, \dots, m$  charged ionic species dissolved within it, the free charge density is the sum of the densities of the individual species.

$$q = \sum_{\alpha} e z_{\alpha} \rho_{\alpha} \quad (2.2)$$

In this expression,  $z_{\alpha}$  is the valence,  $\rho_{\alpha}$  is the number density, and  $e$  is the fundamental charge.

---

\*University of Arizona, Program in Applied Mathematics, bhong@math.arizona.edu

†Sandia National Laboratories, mperego@sandia.gov

**2.1. Poisson Boltzmann.** The fundamental simplified model for the electric potential is the Poisson Boltzmann (PB) model. In this model the individual ions are assumed to be non-interacting point charges, implying that no crowding or correlation effects between the ions are considered. The name of the PB model is derived from the assumption that the ion density of the  $\alpha$  species (in an assumed equilibrium) is given by the Boltzmann distribution.

$$\rho_\alpha = \rho_\alpha^b \exp\left(-\frac{ez_\alpha}{kT}\psi\right) \quad (2.3)$$

where  $\rho_\alpha^b$  is the density of the  $\alpha$  species in the bulk,  $k$  is the Boltzmann constant, and  $T$  is the temperature of the medium. Combining (2.1), (2.2), and (2.3) results in the classical PB equation.

$$-\nabla \cdot (\epsilon_0 \epsilon \nabla \psi) = \sum_\alpha ez_\alpha \rho_\alpha^b \exp\left(-\frac{ez_\alpha}{kT}\psi\right) \quad (2.4)$$

The PB equation is typically paired with a Neumann boundary condition for a fixed surface charge density  $q_s$ .

$$-\epsilon \epsilon_0 \nabla \psi \cdot \mathbf{n} = q_s, \text{ on } \partial\Omega \quad (2.5)$$

where  $\mathbf{n}$  is the surface normal to  $\partial\Omega$ .

**2.2. Modified Poisson Boltzmann.** The modified Poisson Boltzmann (MPB) model uses a phenomenological modification of the Poisson Boltzmann equation that takes into account the finite sizes of the ions. Intuitively this can be thought of as including crowding, or equivalently, steric effects of the ions. This model is derived in some detail in [2]. The basic idea is that each of the ions, assumed to be of equal size, is given a diameter  $a$ . Then, the energy associated with this system is adjusted so that crowding (i.e., high local densities) will increase the energy. The result is that in areas where the density would be unphysically large (for example near a charged surface) the distribution will instead saturate and spread out into the medium. The density of the  $\alpha$  ionic species for arbitrary valences is given in [3].

$$\rho_\alpha = \frac{\rho_\alpha^b \exp(-\frac{ez_\alpha}{kT}\psi)}{1 + a^3 \sum_j \rho_j^b (\exp(-\frac{ez_j}{kT}\psi) - 1)} \quad (2.6)$$

It is useful to note that as the ion size diminishes ( $a \rightarrow 0$ ), the saturation factor in the denominator disappears and the density is reduced to the Boltzmann distribution (2.3). Combining this with (2.1) and (2.2) yields the MPB equation.

$$-\nabla \cdot (\epsilon_0 \epsilon \nabla \psi) = \sum_\alpha ez_\alpha \left[ \frac{\rho_\alpha^b \exp(-\frac{ez_\alpha}{kT}\psi)}{1 + a^3 \sum_j \rho_j^b (\exp(-\frac{ez_j}{kT}\psi) - 1)} \right] \quad (2.7)$$

It is possible to allow the ion sizes to vary between species  $a = a_\alpha$ . However, no analytic form exists for the density (2.6) in this case [3]. In cases where the ions differ substantially in size, (2.6) is unsuitable because it will ignore the ability of the smaller ions to pack between the larger ions [6]. However, the details of such a consideration are outside the scope of this report.

**2.3. Biharmonic Modified Poisson Boltzmann.** A further modification to the PB equation that accounts for ion-ion correlations is presented in [9]. The result is a modified

Laplace equation that includes a biharmonic operator. The resulting model assumes a spatially invariant dielectric constant, and is given by

$$\epsilon_0 \epsilon (l_c^2 \nabla^2 - 1) \nabla^2 \psi = q, \text{ in } \Omega. \quad (2.8)$$

Here  $l_c$  is a phenomenological constant that describes the length scale for the correlation effects. This is usually taken close to the Bjerrum length  $l_B = e^2 / (4\pi kT \epsilon \epsilon_0)$ . For details concerning the physics and derivation we refer the reader to [9]. The model includes the “crowding” effects described by (2.6). For spatially invariant  $\epsilon$  the result is a biharmonic modified Poisson Boltzmann equation (BMPB).

$$\epsilon_0 \epsilon (l_c^2 \nabla^2 - 1) \nabla^2 \psi = \sum_{\alpha} e z_{\alpha} \left[ \frac{\rho_{\alpha}^b \exp(-\frac{e z_{\alpha}}{kT} \psi)}{1 + a^3 \sum_j \rho_j^b (\exp(-\frac{e z_j}{kT} \psi) - 1)} \right] \quad (2.9)$$

This equation is coupled with the original surface charge density boundary condition for the Poisson equation (2.5) and the natural boundary condition

$$-\nabla(\nabla^2 \psi) \cdot \mathbf{n} = 0 \text{ on } \partial\Omega. \quad (2.10)$$

**2.4. Classical Density Functional Theory.** To provide a basis for evaluating the phenomenological models described above, we will use classical density functional theory (cDFT). In particular, we will use fundamental measure theory (FMT) for hard spheres. This technique utilizes finite ion sizes and provides fundamental descriptions for the correlation effects of the ions resulting from the Kohn-Sham equations. For details on this approach, we refer the reader to [10] and [7].

**2.5. Stokes equations.** We require a fluid model for the numerical experiments that compute colloid velocity in a microchannel. We choose the low Reynolds number Stokes limit to the Navier-Stokes equations. The Stokes equations with an added forcing term from the electric field are

$$-\eta \nabla^2 \mathbf{u} + \nabla P = q \mathbf{E}, \quad (2.11)$$

$$\nabla \cdot \mathbf{u} = 0 \text{ in } \Omega. \quad (2.12)$$

Where  $\mathbf{u}$  is the fluid velocity,  $P$  is the pressure,  $q$  is the charge density given by the previous models,  $\eta$  is the dynamic viscosity, and  $\mathbf{E}$  is the total electric field. The electric field comprises two components: a static driving field  $\mathbf{E}_{app}$  and contributions from the local charges  $-\nabla\psi$ .

$$\mathbf{E} = -\nabla\psi + \mathbf{E}_{app} \quad (2.13)$$

In this report we focus on a one way coupling between the fluid and charge models in which the fluid density does not influence the equation for  $\psi$ . Two way coupling will be considered in future work.

**3. Numerical Implementation and Verification.** All of the models are implemented using the finite element method (FEM). The PB-based models are implemented in the front-end “Drekar” which is part of Sandia’s numerical library Trilinos [4]. We use a previous implementation of cDFT for hard spheres, Tramonto [1], which also utilizes the Trilinos library. The Navier-Stokes equations were previously implemented in Drekar.

The implementation of the PB (2.4) and MPB (2.7) systems are straightforward because they only require standard  $C^0$  finite element spaces. To implement the BMPB system (2.9)

we introduce an auxiliary variable  $w$  which transforms this fourth order problem into an equivalent system of two second order equations.

$$\begin{cases} -l_c^2 \nabla^2 w + w &= q \\ -\epsilon \epsilon_0 \nabla^2 \psi &= w \end{cases} \quad (3.1)$$

with the corresponding boundary conditions (2.10) and (2.5),

$$\begin{cases} \partial_{\mathbf{n}} w &= 0 \\ -\epsilon \epsilon_0 \partial_{\mathbf{n}} \psi &= q_s \end{cases} \quad \text{on } \partial\Omega \quad (3.2)$$

where  $\partial_{\mathbf{n}}$  is the normal derivative (i.e.  $\partial_{\mathbf{n}} w = \nabla w \cdot \mathbf{n}$ ). This system can be discretized using  $C^0$  elements.

**3.1. FEM Convergence Verification.** To establish correct implementation of the models, we fabricate analytical solutions and forcing functions for the Laplace operator (2.1) and the biharmonic-Laplace operator (2.8). For example, for the biharmonic-Laplace operator we use a spherically symmetric solution

$$\psi = \exp\left[-\frac{2}{3}(x^2 + y^2 + z^2)\right], \quad w = -\frac{4}{9}\epsilon(4x^2 + 4y^2 + 4z^2 - 9)\psi \quad (3.3)$$

and a standard trigonometric solution

$$\psi = \sin(\pi x) \sin(\pi y) \cos(\pi z), \quad w = 3\pi^2 \epsilon \psi \quad (3.4)$$

with appropriate domains and boundary conditions. Fig. 3.1 shows  $L^2$  convergence data for (3.1). Similar results were obtained for the other models.

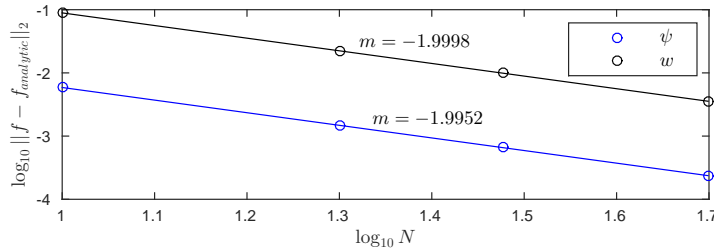


Fig. 3.1:  $L^2$  norm convergence of the FEM solution of (3.1) to the analytical trig solution (3.4).  $N$  is the number of points in each coordinate direction.

**4. Electric potential comparison.** In this section we compare the phenomenological models for electron correlations discussed in sections 2.2 and 2.3 with cDFT (sec. 2.4), which we treat here as the relative “golden standard.” We use the typical geometry for such a comparison: a single infinite charged plate at  $x = x_0$ . This implies that the equations for the potential can be reduced, by symmetry, to equations in  $x$  only. The implementations are fully 3D, but we use the reduced equations here to demonstrate the differences between the models more directly.

Bulk Density ( $\rho_\alpha^b$ )	$6.02 \times 10^{26}$ ions/m <sup>3</sup>
Relative permittivity ( $\epsilon$ )	78.5
Ionic charges ( $z_\alpha$ )	+1, -1
Temperature ( $T$ )	298 K
Boundary surface charge ( $q_s$ )	-0.3 C/m <sup>2</sup>

Fig. 4.1: Parameters used for simulations in sec. 4.

**4.1. Effects of adjusting ion length scales.** In the baseline comparison, we expect all three models to reduce to the PB solution. We compute solutions for a 1:1 electrolyte (fig. 4.1) with densities  $\rho_+$  and  $\rho_-$ . The electrostatic potential is scaled by  $\beta = e/(kT)$ . Choosing  $a = 0$  in the steric model (2.7) or  $a, l_c = 0$  in the biharmonic model (2.9) reduces the respective equations to the PB model (2.4). In light of this we only include a single representation for all three equations in fig. 4.2. It can also be shown that in the limit as the diameter of the hard spheres  $\rightarrow 0$ , the cDFT model also reduces to the PB model. In Tramonto the cDFT model is implemented using non-dimensionalized units, so we adjusted the length scales to match our dimensional PB equations (fig. 4.2).

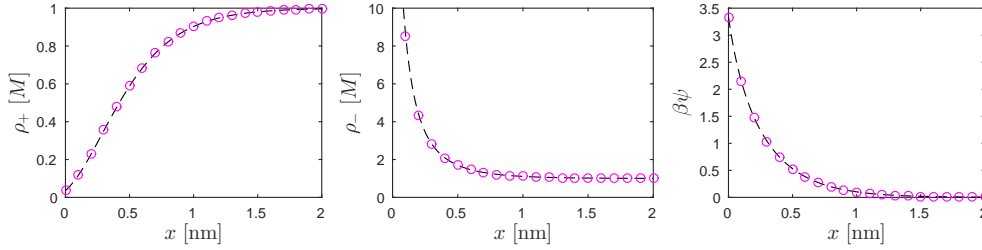
Fig. 4.2: PB (--) compared to the small ion limit in cDFT (○) for an infinite charged sheet at  $x = 0$ .

Figure 4.3 demonstrates the effects of varying the parameters that determine length scale of the ion interactions within cDFT and the phenomenological models. The first row shows the characteristic behavior of the charged layer in cDFT as the ion size is increased: the density of ions near the charged surface is reduced, and for a sufficiently large ion size alternating layers are formed. The increased width of the charged layer also increases the potential ( $\beta\psi$ ) since the potential is not immediately balanced by the large surface charge density seen in the PB case.

The results for the MPB case (second row) are similar to those in the cDFT case. Increasing the effective ion diameter ( $a$ ) results in a saturated layer of negative ions. The phenomenological model does not produce layering, since the description is too simple to generate this effect. However, the overall spread of the density can reproduce the length of the layer if the correct effective ion diameter is chosen. It is interesting to note that increasing the effective ion diameter  $a$  in the MPB model regularly has a more dramatic effect than increasing the relative ion diameter  $\sigma_{ff}$  in the cDFT model. This is probably due to the cubic dependence on  $a$  in (2.7). This means that the scaling for  $a$  should probably not be chosen in the same way as the scaling for the actual ion diameter. It is noted in [3] that the value of  $a$  chosen to fit experimental data in one article was close to the diameter of

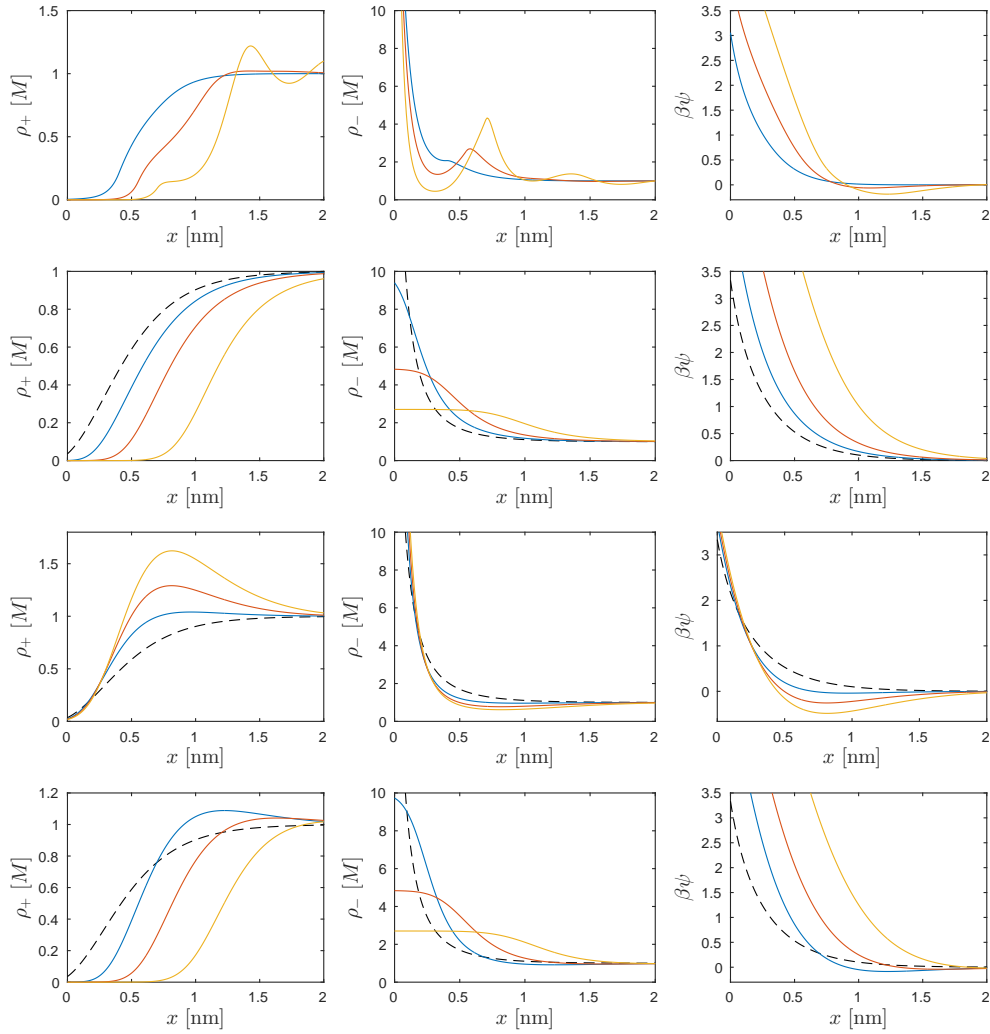


Fig. 4.3: First row: cDFT with increasing relative ion diameter ( $\sigma_{ff}$ ).  $\sigma_{ff} = 0.4$ ,  $\sigma_{ff} = 0.55$ ,  $\sigma_{ff} = 0.7$ . Second row: MPB with increasing effective ion diameter ( $a$ ).  $a = 0.55$  nm,  $a = 0.70$  nm,  $a = 0.85$  nm. Third row: BMPB with increasing correlation length ( $l_c$ ) but no effective diameter ( $a = 0$ ).  $l_c = 0.2$  nm,  $l_c = 0.3$  nm,  $l_c = 0.4$  nm. Fourth row: the full BMPB model. This is the same as row two, but with  $l_c = 0.3$  nm. In all cases the reference line (--) is the pure PB case.  $\beta$  is the inverse of the natural scale for  $\psi$ ,  $\beta = e/(kT)$ . Note that there are changes in the vertical scales.

the first solvation shell, rather than the diameter of the ion. In general, it appears that the value of  $a$  would probably be chosen phenomenologically rather than as the known value of the ion diameter.

The effect of increasing the correlation length  $l_c$  in the BMPB model is shown in the third row. The decrease in density of the negative ions is qualitatively similar to that seen in the cDFT case, but the large increase in the positive ion density along with negative

potential of that region are not seen in any of the cDFT cases. It is difficult to see for this example how increasing  $l_c$  could provide any improvement to the steric model.

The combined BMPB model shown in the final row does not appear to have any advantage over the MPB model (second row). The increase in the positive density is not seen in the cDFT case at the smaller ion sizes, and when it is seen it is in much narrower layers. In conclusion, for this 1:1 electrolyte, it appears that the MPB sufficiently represents the crowding (but not alternate layering) effects, while the addition of the biharmonic operator does not appear to improve the representation (in a purely qualitative sense).

**5. Colloid mobility in a microchannel.** In this section we compute the result of adding steric effects to the problem of computing the mobility of a colloid in a microchannel. We use a cylindrical microchannel which is assumed to be of infinite length by making use of periodic boundary conditions at the termini of the domain. At the center of the microchannel we place a spherical colloid by subtracting it from the cylinder.

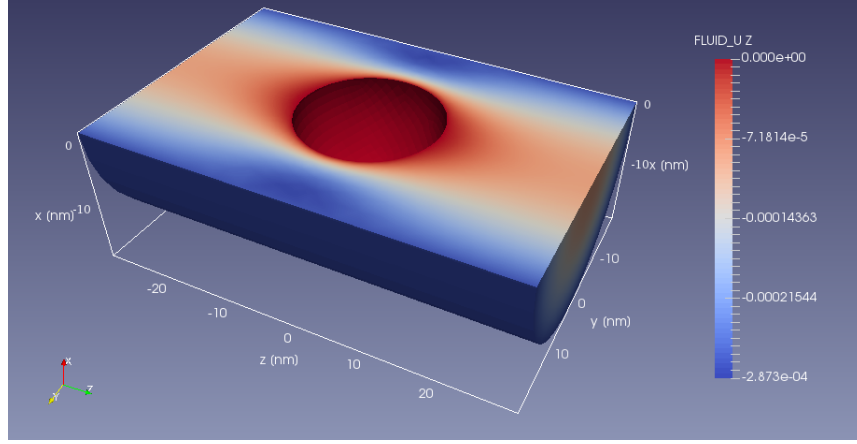


Fig. 5.1: Fluid velocity ( $z$  component) in the microchannel. The appropriate slip boundary condition has been computed resulting in near zero flow velocity in the vicinity of the colloid. This can be interpreted physically as the colloid flowing along with the fluid in the center of the channel.

**5.1. Colloid velocity computation.** In this simulation we use a static mesh and compute the solution to a time independent problem. As a result it is not possible to compute the velocity of the colloid directly. In order to circumvent this issue, we apply a slip boundary condition  $u_z = c$  at the cylinder wall. The velocity  $c$  can be computed so that the total force acting on the colloid in the  $z$  direction is zero. In this case the velocity  $-c$  is equivalent to the velocity of the colloid if it were driven by the same forces in a non-static mesh. This is equivalent (under certain assumptions) to fixing the frame of reference to the colloid. We compute  $c$  by requiring that

$$\int_{\partial S} T_z \, da = 0 \quad (5.1)$$

where  $T_z$  is the  $z$ -component of the normal (total) traction vector and  $\partial S$  is the spherical colloid surface.

$$\mathbf{T} = \sigma_H \cdot \mathbf{n} + \sigma_E \cdot \mathbf{n}$$

Here  $\sigma_H$  is the contribution from hydrodynamic effects and  $\sigma_E$  is the Maxwell stress tensor that determines the electromotive force. These stresses are defined by (see, e.g., [5])

$$\sigma_H = -pI + \eta(\nabla \mathbf{u} + (\nabla \mathbf{u})^\top), \quad \sigma_E = \epsilon\epsilon_0 \left( \mathbf{E}\mathbf{E}^\top - \frac{1}{2}(\mathbf{E}^\top \mathbf{E})I \right). \quad (5.2)$$

An example of the fluid velocity field resulting from this computation is shown in fig. 5.1.

**5.2. Finite element mesh.** The computational mesh is generated using Cubit [8], a mesh generation toolkit created at Sandia National Labs. Using Cubit, we specify the refinement of the mesh on the colloid and the cylindrical shell (fig. 5.2). Since most of the fine details and the surface integral are computed at the colloid, it is more efficient to refine the mesh near the colloid more so than at the cylindrical surface. To illustrate this, we computed the colloid velocity at several mesh refinements (fig. 5.3). It is clear from this graph that the colloid velocities appear to be converging to a value near  $5.4 \times 10^{-5}$  m/s. However, by using a mesh that is more refined near the colloid (e.g., the 5-2 mesh) a more converged value is obtained with far fewer elements (pink diamonds) than with the regularly refined meshes.

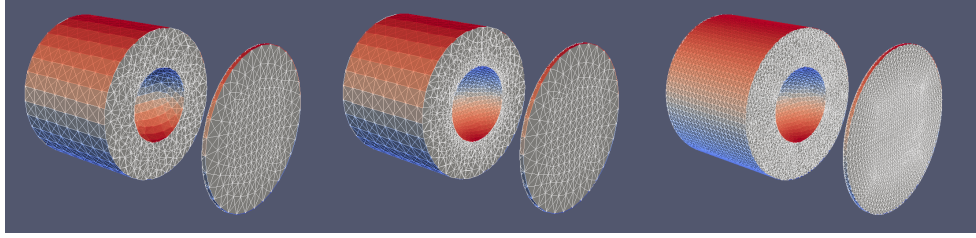


Fig. 5.2: Cutaway of mesh refinements. Left: a 5-4 mesh, where 5 refers to the mesh refinement of the cylinder and 4 refers to the level of refinement on the spherical colloid. The numbers are the mesh refinement values defined in Cubit [8]. Center: a 5-2 mesh. Right: a 2-2 mesh.

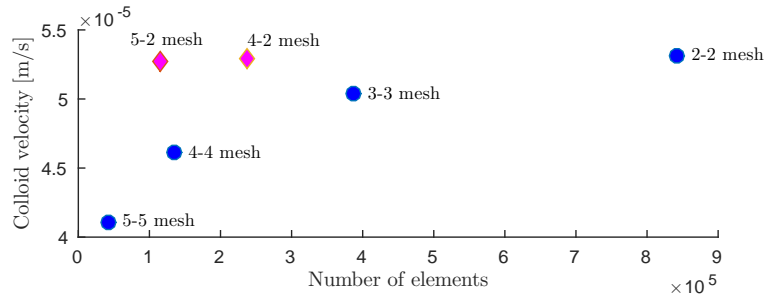


Fig. 5.3: Calculated colloid velocity at various mesh refinements. The blue data points represent regular mesh refinements.

**5.3. Mobilities at various effective ion sizes.** To compute the mobility we run several simulations with various applied fields ( $E_{app}$ ). The fixed parameters are listed in fig.



5.4. The external field is applied along the axis of the cylinder, i.e.  $\mathbf{E}_{app} = [0, 0, E_{app,z}]$ . In the first case we compute the mobility with the pure PB model (sec. 2.1). The velocity should be linearly related to the applied field and for the most part this is observed in fig. 5.5, barring some error probably resulting from insufficient convergence of the model. The mobility is computed as the average value:  $1.38 \times 10^{-10}$  for the PB case.

External field $E_{app,z}$ (N/C)	$0.25 \times 10^6$	$1 \times 10^6$	$4 \times 10^6$
Colloid velocity $v$ (m/s)	$0.336 \times 10^{-4}$	$1.390 \times 10^{-4}$	$5.607 \times 10^{-4}$
Colloid Mobility $v/E$	$1.344 \times 10^{-10}$	$1.390 \times 10^{-10}$	$1.402 \times 10^{-10}$

Fig. 5.5: Mobilities computed in the pure PB case.

Performing the same computation with varying ion sizes ( $a$ ) in the steric model (2.7) demonstrates the effect of an increasingly saturated double layer on the mobility of the colloid. As the effective ion sizes increase, the mobility is increased (fig. 5.6). We are still working on understanding the reason for this effect and whether or not it is a physically relevant. In order to assess whether or not this is meaningful we still need to compare this to mobilities calculated using the cDFT model.

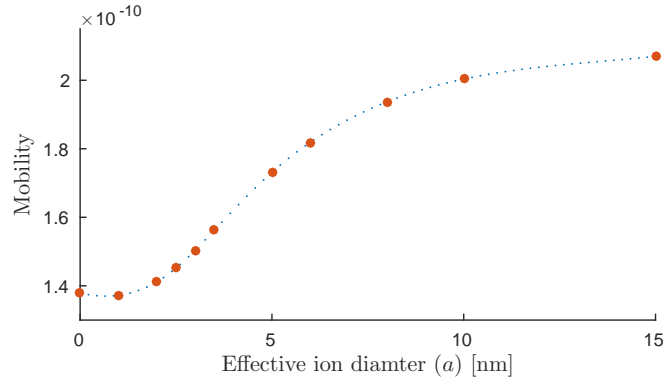


Fig. 5.6: Calculated mobilities using the MPB equation (2.7) at various effective ion sizes ( $a$ ).

Bulk Density ( $\rho_\alpha^b$ )	$2.5 \times 10^{23}$ ions/m <sup>3</sup>	Colloid Radius	10 nm
Relative permittivity ( $\epsilon$ )	80	Channel length	60 nm
Ionic charges ( $z_\alpha$ )	+1, -1	Channel Radius	20 nm
Temperature ( $T$ )	300 K	Viscosity	0.856 Pa · s
Colloid charge	200e	Fluid density	10 <sup>3</sup> kg/m <sup>3</sup>

Fig. 5.4: Parameters used for simulations in sec. 5.

**6. Conclusions.** We have implemented models that account for the crowding and correlation effects of ions in charged layers without the high computational cost of an ab-initio model. These models are able to produce some important features, such as saturation of the ion layer. However, it is unclear if all of the modifications to the PB model are useful in the case of the 1:1 electrolyte that we simulated. We plan to continue our work by investigating the effects of ion-ion correlations on colloid mobility in a nanochannel, which we described at the end of this article. In order to make a fair comparison, we still require a coupled implementation of the cDFT and Stokes equations. Once this is complete, we will be able to assess at what level of detail the ion interactions should be modeled for this application. For example, we are interested in determining if the alternating layers seen in cDFT impact the colloid mobility substantially or if the MPB or BMPB models are sufficient.

## REFERENCES

- [1] See <http://software.sandia.gov/tramonto>.
- [2] I. BORUKHOV, D. ANDELMAN, AND H. ORLAND, Adsorption of Large Ions from an Electrolyte Solution: a Modified Poisson–Boltzmann Equation, *Electrochimica Acta*, 46 (2000), pp. 221–229.
- [3] P. GROCHOWSKI AND J. TRYLSKA, Continuum Molecular Electrostatics, Salt Effects, and Counterion Binding – a Review of the Poisson–Boltzmann Theory and its Modifications, *Biopolymers*, 89 (2008), pp. 93–113.
- [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] J.-P. HSU, L.-H. YEH, AND M.-H. KU, Evaluation of the Electric Force in Electrophoresis, *Journal of colloid and interface science*, 305 (2007), pp. 324–329.
- [6] A. A. KORNYSHEV, Double-Layer in Ionic Liquids: Paradigm Change?, *The Journal of Physical Chemistry B*, 111 (2007), pp. 5545–5557.
- [7] R. ROTH, R. EVANS, A. LANG, AND G. KAHL, Fundamental Measure Theory for Hard-Sphere Mixtures Revisited: the White Bear Version, *Journal of Physics: Condensed Matter*, 14 (2002), p. 12063.
- [8] J. SHEPHERD ET AL., CUBIT Mesh Generation Toolkit, SAND2000-2647, Sandia National Laboratories, Albuquerque, New Mexico, (2000).
- [9] B. D. STOREY AND M. Z. BAZANT, Effects of Electrostatic Correlations on Electrokinetic Phenomena, *Physical Review E*, 86 (2012), p. 056303.
- [10] J. WU, Density Functional Theory for Chemical Engineering: from Capillarity to Soft Materials, *AIChE Journal*, 52 (2006), pp. 1169–1193.

## ANALYSIS OF PHASE PRECESSION FROM DUAL COMPONENTS IN HIPPOCAMPAL PLACE CELLS

KELSEY R. KALMBACH\* AND FRANCES S. CHANCE†

**Abstract.** Phase precession in hippocampal place cells is a well studied phenomenon in which cells spike at progressively earlier phase relative to the theta oscillation as the animal moves through the place field of that cell. Little is known about the underlying mechanism causing this firing. There exist two well-known, anatomically-distinct inputs to the CA1 region of the hippocampus, and the combination of these two inputs could lead to phase precession in the place cells. In this study, we examine neurobiological data to look for evidence that CA1 place cell firing is driven by two distinct components. We compare place cell data of both old and young rats to determine if age, which has an impact on hippocampus connectivity [5], has an impact on place cell firing. We use clustering analysis, phase range analysis, and curve fitting techniques to quantify the underlying structure of the place field data.

**1. Introduction.** The hippocampus is believed to play a crucial role in spatial navigation and the formation of episodic memories [14, 18, 19]. Such tasks are often difficult for computers to reproduce. Neurally-inspired algorithms can take advantage of what is known about how the brain processes information to compute similar tasks. However, much is still unknown about the exact mechanisms by which the brain processes information. Further insights into how the brain performs these computations could lead to improved neurally-inspired algorithms.

In the hippocampus, it has been demonstrated that many cells only fire when the animal is in a specific location. These cells are referred to as place cells. The place field of a cell refers to that specific location where the cell shows elevated firing rates [16, 17, 18]. Because these cells only fire in specific locations, it is believed that place cells are crucial for spatial navigation [18]. Additionally, it has been well demonstrated that the firing of these cells is correlated with the 7-12 Hz theta oscillation in the local field potential (LFP). At the entry of each place field, spikes occur at a late phase of the theta oscillation. As the animal moves through the place field, the spikes occur at progressively earlier phases relative to the theta oscillation. This phenomenon of spikes transitioning from firing at high phases relative to the theta oscillation to low phases relative to the theta oscillation is known as phase precession. The phase of spikes at the end of the place field is typically nearly  $2\pi$  radians earlier in the theta phase than the spikes at the beginning of the place field, that is, spikes occur at nearly all values of theta phase [19].

This project focuses on place cells from CA1 pyramidal cells, one of the subfields of the hippocampus. There are two well-known anatomically-distinct input pathways to CA1: CA3 and layer 3 of the entorhinal cortex, EC3 [1]. It has been shown that CA3 cells also form place fields which can show phase precession; however, CA3 cells form place fields less frequently than CA1 cells [11]. Additionally, CA3 place fields are typically shorter and show less phase precession than CA1 place fields [11]. Certain studies have shown that by eliminating either CA3 or EC3 input, CA1 cells will still form place fields and show phase precession, but the firing is more diffuse and the place fields are larger when either pathway is eliminated [3, 4, 15]. Therefore, it is possible that both of these pathways can independently drive CA1 place cell firing, but the combination of these two inputs creates a more well-defined place field [1]. Input from EC3 arrives in CA1 more than one-quarter theta-cycle before the input from CA3 [12, 13]. If this theta-phase offset is combined with a spatial offset between CA3 and EC3 inputs, these two inputs to CA1 could give rise to phase precession [6].

---

\*Colorado School of Mines, kkalmbac@mines.edu

†Sandia National Laboratories, fschanc@sandia.gov

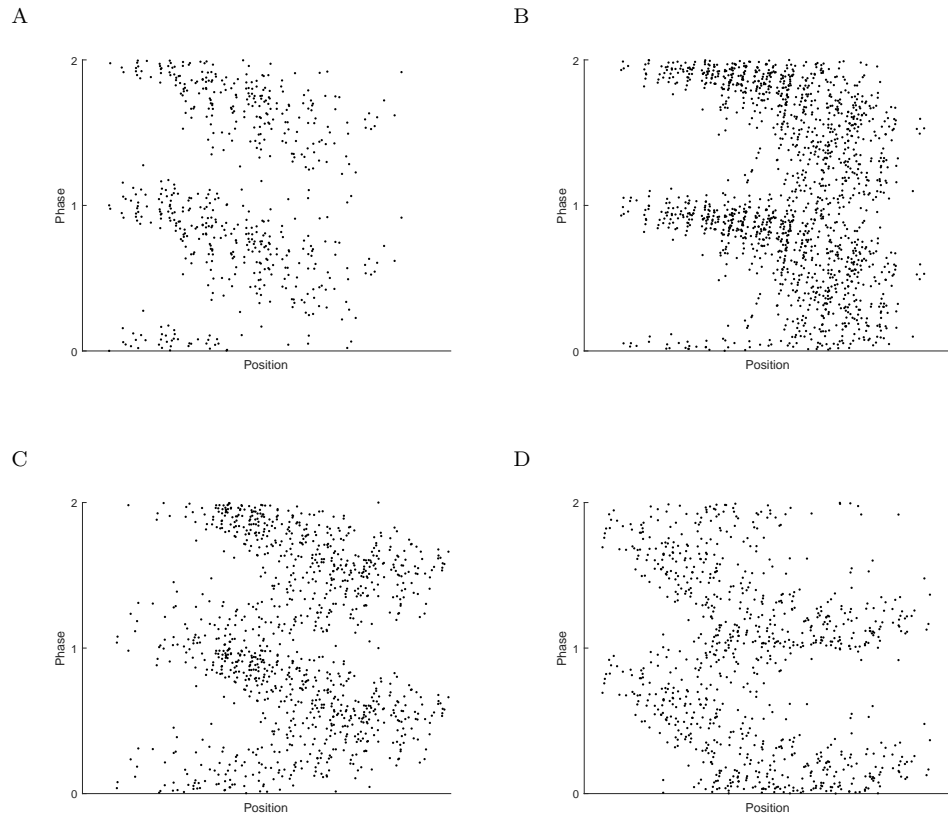


Fig. 1.1: Examples of typical relationships between the theta phase and animal position. Each dot is one recorded spike. The animal ran from left to right in all figures. Here, theta phase is reported mod 1 and the phase precession is plotted twice in all cases for clarity. Typical relationships between theta phase and position are: (a) linear phase precession, (b) phase precession which accelerates as position increases, (c) two distinct clusters of spikes giving rise to a boomerang-shaped phase precession, and (d) other nonlinear phase precession. Data is replotted from Schimanski et al. [22].

Phase precession in rat hippocampal place cells is traditionally thought to be linear with respect to position [19], but Skaggs et al. demonstrated that phase precession is often not well described by a linear shift [25]. Some place cells do show linear phase precession (Figure 1.1A), but many show more complicated phase precession. The phase precession can accelerate as the rats moves across the place field, giving rise to a downward curve in the plot of phase versus position (Figure 1.1B). The phase precession can also show two distinct clusters of spikes giving rise to a boomerang-shaped plot of phase versus position (Figure 1.1C) or other nonlinear phase precessions (Figure 1.1D) [22, 25]. If the relative strength of CA3 or EC3 input changed, or CA3 input came from a cell which showed phase precession, a model based on these two inputs to CA1 could account for many of these complex patterns of phase precession [6]. This project aims to examine neurobiological data to look for evidence that CA1 firing is driven by two distinct components.

It has been shown that there are some connectivity changes to the hippocampus as a rat ages, which could lead to different firing patterns of place cells [5]. However, the number of CA1 and CA3 pyramidal cells does not decrease with age. Additionally, in older rats, it has been demonstrated that the number of synaptic contacts in CA1 from both CA3 and EC3 does not decrease [5]. However, in older rats, there is a decrease in the number of functional synaptic contacts from CA3 [5, 21]. Little is known about how aging affects the connections between EC3 and CA1 [5]. Despite these changes in synaptic connections, the number of place fields and the range of phase precession is similar in both young and old rats [24]. It is possible that there is a change in firing patterns of CA1 place cells in older rats due to changes in synaptic contacts.

In this study, we analyze the firing patterns of both old and young rats to explore the possibility that CA1 place cell firing is driven by two components, the input from CA3 and from EC3. We look at clustering the data to get a sense of the underlying structure of the place field data, and we perform an arctangent fit to the place field data to quantify the transition from firing at a high theta phase to firing at a low theta phase. We find that, in general, there are no differences between the structure of place fields for the young and old rat populations. However, there is some evidence for a transition between a component firing at a high theta phase to a component firing at a low theta phase, which could be caused by dual inputs.

**2. Methods and Materials.** To investigate the underlying structure of the place field data, we analyze CA1 place field data from a study published in 2013, provided to us by members of the Barnes lab at U. Arizona. First we preprocess the data with techniques including scaling and unwrapping the data. Different preprocessing techniques were needed for different analyses. We then analyze the data in three different ways: data clustering, curve fitting, and a phase range analysis.

**2.1. Electrophysiology Data.** A data set from a study published in 2013, [22], has been provided to us by members of the Barnes lab at U. Arizona. The experimental studies have been described in detail elsewhere [22]. Briefly, in these experiments, 12 adult Fischer-344 rats were trained to run back and forth on a track for a food reward. Of the 12 rats, 6 were young (9-12 months) and 6 were old (25-28 months). After a period of initial training, the rats were implanted with 12 tetrodes in the dorsal layer of the hippocampus to record spikes extracellularly from CA1 pyramidal cells. The rats were allowed 14 days to recover from surgery and to have the tetrodes adjusted to obtain clear CA1 action potentials. After recovery from the surgery, rats ran a novel circular track with a barrier (85 cm diameter), with CA1 activity recorded. Rats were placed on one side of the barrier and ran to the other side of the barrier for a food reward, then turned around and ran back to the original starting position for another food reward. Rats ran 74 laps in a session with a resting period in their home cages between training sessions. Cell activity was recorded for two training sessions a day over a 31 day period. One old rat became ill and did not complete all 31 days of the experiment [22].

Only cells with an average firing rate of greater than 4 Hz during the running sessions and greater than 0.01 Hz during rest periods were considered in this analysis [22]. In addition, spikes which occurred when the running speed was less than 6 cm/s were not considered in this analysis since theta oscillations do not occur when the rat is stationary [27]. Place fields were then defined as locations where the peak firing rate was greater than  $> 2$  Hz. The place field boundary was set as the location when the firing rate fell below 10% of the local peak firing rate on either side of the peak. Cells with no place-specific firing patterns were not included in this analysis. Additionally, firing that occurred near the reward sites were not included in this analysis since the acts of turning and eating are not associated

with theta wave activity [22, 27]. Finally, only place fields whose distribution of spikes is approximately unimodal along the position axis are included in this analysis. Runs in the clockwise and counterclockwise directions were analyzed individually since it has been well demonstrated that the firing pattern and formation of place cells is highly dependent on direction in narrow arm mazes [10, 22].

**2.2. Data Scaling.** For many computations done on the place field data, the relative scaling of the data can have a large effect on the results obtained. As such, we have scaled the data in two different ways for different applications of data analysis. In both, we have scaled the data so that it does not depend on the absolute length of the place field, which can vary greatly within and between rats. For all place fields, we define the position and phase of each spike as  $(x_i, \phi_i)$  for spike  $i$ , with  $i = 1, \dots, N$ , the total number of spikes in a given place field. The position variable  $x$  is a one dimensional linearization of the circular track.

For the first data scaling, we scaled the theta phase such that it is defined on  $[0, 2\pi)$ . This scaling was chosen so that future calculations on the periodic data are based on the assumptions of  $2\pi$  periodic data. We then scaled the position data such that the standard deviation in the position is the same as the standard deviation in the theta phase. Because the data is periodic with respect to the theta phase variable, a modified standard deviation definition was used. The angular standard deviation is defined as

$$s_\phi = \sqrt{2(1 - R)} \quad (2.1)$$

where  $R$  is the length of the mean resultant vector, defined as

$$R = \frac{1}{N} \sqrt{\left(\sum_{i=1}^N \sin(\phi_i)\right)^2 + \left(\sum_{i=1}^N \cos(\phi_i)\right)^2} \quad (2.2)$$

[2]. Then the position data is scaled by  $\frac{s_\phi}{s_x}$ , where  $s_x$  is the standard deviation of position, so that the two variables so that the two variables have equal standard deviations. The standard deviation of position is dependent on the length of the place field and the spread of the spikes, so scaling the position variable based on the standard deviation will take into account the absolute length of the place field. This scaling ensures that distance measures are not biased towards one variable over the other. This scaling is used when a distance measure between data points is needed.

For the second scaling, the theta phase is once again scaled such that values for theta can range from  $[0, 2\pi)$ . Then the position variable is rescaled so that it defines the percent of the place field crossed. That is, we rescale  $x$  such that

$$x_i = \frac{x_i - \min_{i=1, \dots, N} x_i}{\max_{i=1, \dots, N} x_i - \min_{i=1, \dots, N} x_i}. \quad (2.3)$$

We use this scaling whenever a comparison across place fields is needed. This scaling ensures that the both the phase and location are defined on the same range of values,  $[0, 2\pi)$  and  $[0, 1]$  respectively.

**2.3. Unwrapping Data.** For certain computations, it is required that the data not be defined on the unit cylinder, but rather on a plane. Often, to unwrap the circular data, the data is cut at a phase such that the linear regression line of the unwrapped data gives the largest explained variance [11, 19, 23]. However, with this method of unwrapping the data,

there will nearly always be at least a few points which are classified as having a low phase when all neighboring points have a high phase or the other way around. These points will provide a large amount of bias when trying to perform calculations using this unwrapped data [9]. To address this issue, we developed a new method for unwrapping data using a nearest neighbors approach. Starting in the center of the place field and moving out to points further away from the center, we look at all data points within a certain radius of the given data point. If more than half of these points are at a much higher phase than the given point (more than  $\pi$  greater),  $2\pi$  is added to the phase. Similarly, if more than half of the points in the radius are at a much lower phase,  $2\pi$  is subtracted from the phase. This process is iterated over the entire data set until no points change in phase. Because the density and general shape of the phase precession varies greatly between place fields, there is no single radius which will give a proper unwrapping of the data for every place field. Instead, this algorithm is run over a range of radii: 0.5 to 2 in steps of 0.05. The proper unwrapping is chosen as the resultant place field with the most negative correlation coefficient. Since this process is dependent on the distance between data points, we use the first method of scaling described above, such that the standard deviation is the same in both variables.

**2.4. Data Clustering.** Clustering analysis was performed on the position and phase data for each individual place field. To cluster the data, we used a variation on the  $K$ -means clustering algorithm. Since the data is defined on a cylinder, that is,  $\phi$  is periodic while  $x$  is not, the distance measure needs to be modified accordingly. The standard Euclidean distance is used in the  $x$  dimension, but in the  $\phi$  direction, the distance between two points  $\phi_i$  and  $\phi_j$  is

$$\min(|\phi_i - \phi_j|, 2\pi - |\phi_i - \phi_j|). \quad (2.4)$$

This ensures that the distance between 0 and  $2\pi$  is zero, and the distance between 0 and  $\pi$  is maximal. Additionally, the average of cylindrical data needs to be modified accordingly. The standard average is used in the  $x$  direction, but the average of the phases  $\phi_i$  is defined as

$$\bar{\phi} = \arctan^* \frac{\sum_i (\sin(\phi_i))}{\sum_i (\cos(\phi_i))} \quad (2.5)$$

where  $\arctan^*$  is the quadrant specific arctangent function [2]. Here we will use the first scaling method defined above, with the standard deviation the same for both the phase and position variables, since the clustering depends on the distance between two points.

Briefly, the  $K$ -means algorithm begins by randomly setting  $k$  points to be the cluster centers. Then, each additional data point is assigned to the nearest cluster center, using the appropriate distance measure. Once all points are assigned a cluster, the cluster center is redefined as the average of all points in that cluster. Every point is then once again assigned to the nearest cluster center and the cluster centers are redefined based on that set of points. This process is continued until the assignment of points does not change after the cluster center is redefined [8].

The  $K$ -means algorithm requires the number of clusters  $k$  to be known beforehand [8], but we would like to determine the optimal number of clusters to describe the data. To do this, we use the Bayesian Information Criterion. The BIC attempts to determine the optimal number of clusters by rewarding how well the data is described by the clustering while penalizing the added complexity of adding more clusters to the model [20]. The BIC value was calculated for clustering done with  $k = 1, \dots, 6$ . We only optimized the number of clusters over the possibility of  $1, \dots, 6$  clusters because it was expected that the data would

be well described by either 1 or 2 clusters. Allowing a maximum of 6 clusters allowed us to test well beyond the expected limit without adding a significant amount of computation time to the algorithm. The model with the highest value of BIC was considered the ‘best’ model, and the corresponding value of  $k$  was considered the optimal number of clusters [20].

The  $K$ -means algorithm only converges on a locally optimal solution, or can fail to converge if the clusters do not remain constant after the algorithm reaches some maximum number of iterations of the algorithm [7]. Because of these two factors, the clustering might not be consistent. If the clustering is not consistent, the BIC values might be different for different runs of the algorithm, which can in turn change the optimal number of clusters  $k$  determined by the BIC measure. Therefore, the algorithm to determine the best model was run several times and the optimal number of clusters was determined to be the mode of the optimal number of clusters for each run. Typically, the algorithm converged on the same number of clusters for all runs.

**2.5. Fitting Data to the Arctangent Function.** The unwrapped data of each place field was fit to an arctangent function with four parameters

$$\Phi(x) = -a \cdot \arctan((x - b) \cdot c) + d. \quad (2.6)$$

where  $x$  are the position of the spikes and  $\Phi(x)$  is the predicted phase of the spike. This functional form was chosen because it approaches two different horizontal asymptotes as  $x \rightarrow \pm\infty$ . If there exist two distinct clusters of spikes which fire at different average phases, we expect that the theta phase will tend to two different values at the entrance and exit of the place field respectively. This behavior might be well captured by the arctangent function. We would like to be able to compare certain aspects of these fits, such as the location of the inflection point across place fields, so we scale the unwrapped data using the second method described above, with location defined as percentage of place field crossed. Here, the curve is fit such that the difference between the actual phase of the data and the phase predicted by the curve is minimized. This does not depend on the distance in both the position and phase variables, so the first scaling method is not needed.

**2.6. Phase Range Analysis.** The phase range is defined as the slope of the line calculated using a circular-linear fit, described below, multiplied by the spatial range of the trial. If the place field is being considered as a whole, the spatial range of the trial is one. If we are considering each pass through the place field separately or only one cluster of the data, the spatial range is the fraction of the place field crossed in those spikes [23]. To obtain the slope of the line passing through the place field, we use a circular-linear fit as described in [9] to fit the data points  $(\phi_i, x_i)$  to the line

$$\Phi(x) = 2\pi \cdot a \cdot x + \phi_0 \quad (2.7)$$

where  $a$  is the slope and  $\phi_0$  is the phase offset. The estimate of the slope  $a$  is the value of  $a$  that maximizes the function

$$R(a) = \sqrt{\left[ \frac{1}{n} \sum_{j=1}^n \cos(\phi_j - 2\pi \cdot a \cdot x_j) \right]^2 + \left[ \frac{1}{n} \sum_{j=1}^n \sin(\phi_j - 2\pi \cdot a \cdot x_j) \right]^2} \quad (2.8)$$

The estimate for the phase offset is given by

$$\phi_0 = \arctan^* \frac{\sum_j \sin(\phi_j - 2\pi \cdot a \cdot x_j)}{\sum_j \cos(\phi_j - 2\pi \cdot a \cdot x_j)} \quad (2.9)$$



where  $\arctan^*$  is the quadrant specific arctangent function [9]. The values of the slope were limited to the range  $[-2, 0]$  since for a circular-linear fit, there always exists some arbitrarily high slope which will pass through every data point, but this does not give a good indication of the average slope of the place field [23]. For the phase range analysis, we use the second method for data scaling, such that position is defined as percentage of place field crossed. The only distance measure used here is the difference between the phase of the data and the estimated phase, so there is no need for the data to be scaled such that there is no bias in distance in one variable over the other. Having the position variable be defined as the percent of the place field crossed is consistent with the notion of the spatial range of the trial, which is used in this calculation.

**3. Results.** All results discussed are for day seven of the experiment, unless otherwise noted. This day was chosen because there was data from each of the twelve rats and it was not a novel environment for the rats. Similar results were obtained for all other days of the experiment.

**3.1. Arctangent Fit.** Many place fields show a distinct transition between firing at a high theta phase to firing at a low theta phase. To quantify this transition, we fit the place field data to an arctangent function. An arctangent function transitions from higher values to lower values, with parameters that dictate the slope and place of that transition. Information about where and how quickly that transition occurs would be useful in quantifying how the firing in the place fields evolves for different rats.

For the fit to the arctangent function, we require that there is phase precession in the place field. We say that the place field exhibits phase precession if the correlation coefficient of the unwrapped data is less than  $-0.4$  [23]. This eliminates both place fields that show no phase precession over the entire range of theta (Figure 3.1A) and place fields that show no phase precession because there are only spikes over a small range of theta phases (3.1B). Both of these cases where there is no phase precession are common in certain rats on different days. In cases where there is no phase precession, an arctangent fit will not give any meaningful results, so we do not include these place fields in our analysis.

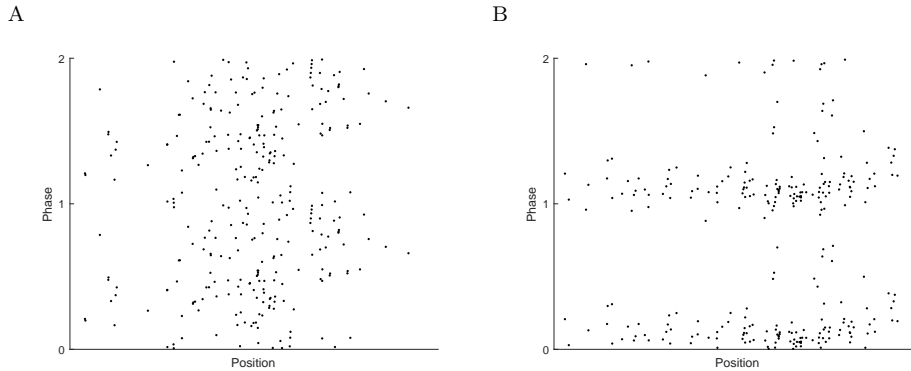


Fig. 3.1: Examples of typical phase precession plots where there is no phase precession evident. **A**, No phase precession seen over the entire range of theta phase. **B**, No phase precession is seen because spikes only occur over a short range of theta phases. Here, theta phase is plotted mod 1 and the phase precession is plotted twice for clarity. Data is replotted from Schimanski et al [22].

When there is phase precession in the place field, the arctangent fit will generally capture the overall behavior of the phase precession; however, this does not seem to be the best function to fit the data. The arctangent function will tend to a slope of zero as  $x$  reaches the entrance or exit of the place field, which does not usually fit the data well. A typical fit to the arctangent function when there is a clear transition from firing at a high theta phase to firing at a low theta phase is shown in Figure 3.2A. Here, the point and shape of the transition is well captured by the arctangent function, but the end behavior tends to be oversimplified. Both the upper and lower components of this phase precession show some amount of slope. This behavior is not well captured by the arctangent fit, which assumes that the behavior on either side on the transition range will tend to a constant value. However, in other cases, the arctangent fit does not well capture the transition from the firing at a high phase to firing at a low phase, such as in Figure 3.2B. Here, the spikes at higher phases show a general downward trend while the spikes at a lower phase show a general upward trend. Because an arctangent function cannot capture this end behavior, the curve does not fit the data well. In particular, the point of inflection does not seem to well capture where the actual transition between the groups occurs.

There are cases where the phase precession appears linear and there is no clear evidence of two components. In these cases the arctangent fit does not give meaningful results. The arctangent function approaches horizontal asymptotes as  $x \rightarrow \pm\infty$ , but in cases where the phase precession appear linear, this long term behavior of the arctangent fit is not observed in the range of meaningful position values and the asymptotes are often well outside the range of  $[0, 2\pi)$ . The arctangent function was primarily chosen to see how the data transitioned from high to low theta phases as indicated by the long term asymptotic behavior of the function. For cases where the phase precession appears linear, this long term behavior has no relevant meaning in understanding the phase precession. So, for these cases, the arctangent function gives no insight to the overall behavior of the phase precession.

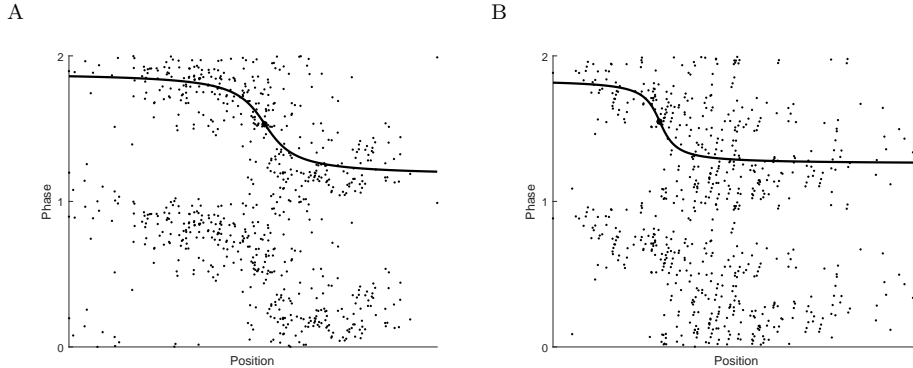


Fig. 3.2: Examples of typical arctangent fits to place field data which show clear evidence of two components, one at a high theta phase and one at a low theta phase. In both plots, the larger black dot indicates the point of inflection of the arctangent function. **A**, The transition is well captured by the point of inflection. **B**, The transition is not well captured by the point of inflection. Here, theta phase is plotted mod 1 and the phase precession is plotted twice for clarity. Data is replotted from Schimanski et al [22].

Because the arctangent fit cannot reliably give a good assessment of the point of in-

flection where the graph transitions from the higher phase to the lower phase, we do not explore the properties of this fit further.

**3.2. Clustering.** In addition to being able to quantifying the transition between a group of spikes occurring at a high phase to those spikes occurring at a low phase, we would like to be able to classify the individual spikes as coming from a component driven by a high phase or a low phase. To group these spikes, we cluster the data using a  $K$ -means algorithm. With the data clustered, we are able to quantify the number of place fields that appear to have two distinct components from those that do not. We are also able to quantify the relative size and other properties of the components from the place fields with two distinct clusters.

When the clustering algorithm is run on all place fields, the spikes are typically clustered into either one or two clusters, while occasionally the algorithm determines that three or more clusters are optimal. The numbers of clusters for all place fields on day seven of the experiment are summarized in Table 3.1. From visual inspection, the place fields that are classified as having three clusters often appear to be well classified as two groups, with one group much larger than the other. The  $K$ -means algorithm tends to group data into roughly equal size groups. When the data is well described by two components, one being much larger than the other, the algorithm will often determine that the optimal number of clusters is three, with the larger component being split into two clusters. Also if there are clear regions of firing at high and low theta phases, but a rather large region where these two seem to overlap, the algorithm will often choose three clusters and specify one cluster for that transition area. If there exists a significant amount of noise in the data, the clustering algorithm will typically not give meaningful results, which explains many of the place fields that are grouped into more than three clusters.

In general, the data is grouped into one cluster when there is no phase precession, such as those examples in Figure 3.1, or when the phase precession is generally linear (Figure 3.3A). The data is often clustered into two clusters when there appears to be a distinct transition between spikes at a high phase to spikes at a low phase (Figure 3.3B). The clustering distribution is similar for the data on other days.

Number of Clusters	1	2	3	> 3	Number of Place Fields
All place fields	248	190	52	21	511
Old rats	94	90	13	9	206
Young rats	154	100	39	12	305

Table 3.1: The number of place fields clustered into 1,2,3, and more than 3 clusters respectively on day seven of the experiment. The breakdown of distribution of clusters for all place fields, all place fields of old rats, and all place fields of young rats respectively

**3.3. Properties of Clustering Analysis.** We see that, in general, the number of spikes is approximately evenly split between the two clusters in the cases where the data is grouped into two clusters. For all rats on day seven of the experiment, the first cluster has approximately half of the total number of spikes ( $53.18\% \pm 9.89\%$ ). There is little difference between the old and young rats in the relative size the two clusters. For old rats, the first cluster has  $54.18\% \pm 9.84\%$  of the total spikes, and for the young rats, the first cluster has  $50.24\% \pm 9.60\%$  of the total spikes. The fact that the number of spikes in each cluster is nearly equal is in part explained by the fact that the  $K$ -means algorithm tends to find clusters of equal sizes. However, spikes of very different phases and positions are

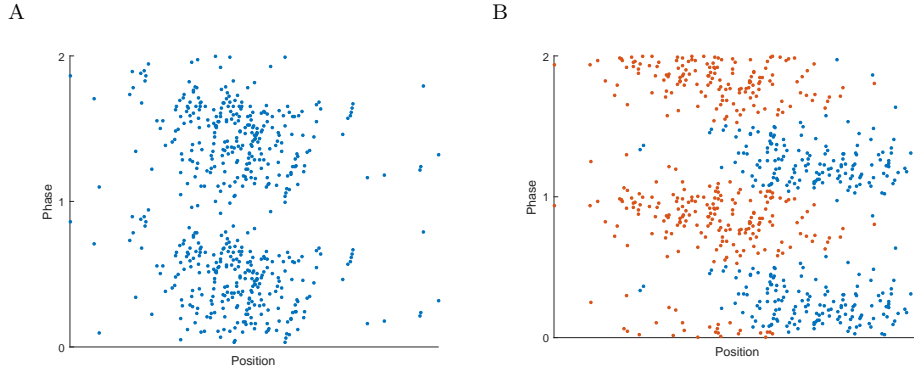


Fig. 3.3: Examples of typical clustering: **A**, One cluster. **B**, Two clusters. Here, theta phase is plotted mod 1 and the phase precession is plotted twice for clarity. Data is replotted from Schimanski et al [22]

rarely clustered together to account for the equal sized clusters. More often, spikes in this transition space between the high and low theta phases are clustered in such a way that the two clusters are roughly equal size, which still gives reasonable looking clusters.

We also find that the two clusters tend to be nearly anti-phase to each other. The center of mass of the first cluster is, on average, at 4.73 radians and the center of mass of the second cluster is, on average, 1.69 radians. The difference between the center of mass of these two clusters is  $2.98 \pm 0.42$  radians for all place fields with two clusters on day seven of the experiment. This is similar to the phase difference between the two clusters seen in the clustering analysis of place fields by Yamaguchi et al [28]. There was no difference in the phase difference between the two clusters for the young and the old rat populations ( $2.94 \pm 0.44$  radians for old rats and  $3.01 \pm 0.39$  radians for young rats). These properties were similar on all other days of the experiment.

**3.4. Phase Range Analysis by Cluster.** Looking at the phase range of the individual place fields gives a good indication of the overall phase precession of the place field. It has been shown that, typically, spikes precess over an entire theta cycle [19], but the exact degree of phase range can vary greatly between specific place fields. A lower phase range would imply that there are significantly fewer spikes at either a high or low phase. Thus, a lower average phase range in a group of place fields could indicate that that group of cells is typically receiving less input that would normally cause the cell to fire at either the high or low phase.

The phase range was calculated for each of the place fields on day seven of the experiment. The average phase range was  $-5.32 \pm 2.93$  radians. That is, on average, the place fields precessed nearly one entire theta cycle, which is consistent with previously reported data [19, 23]. We find that the phase range distribution is different among the place fields with one cluster and those with two clusters (Figure 3.4). The place fields with one cluster had, on average a phase range of  $-4.53 \pm 3.32$  radians while the the place fields with two clusters had, on average a phase range of  $-5.57 \pm 1.76$  radians. The difference between these two means was 1.05 radians, and was statistically significant ( $p < 0.001$ ). We see as well that the spread of phase range is much higher for the place fields with only one cluster. This is likely because the place fields categorized as having one cluster tended to either show

linear phase precession, which would give a high phase range, or did not show much phase precession, which would give a low phase range. Additionally, there are many place fields that show some, approximately linear, phase precession, though not over the entire range of theta phases. These place fields contribute to those intermediate values of phase range. This gives rise to both the lower average and wider spread of phase ranges. There was no significant difference in the phase ranges of the old and young rat populations.

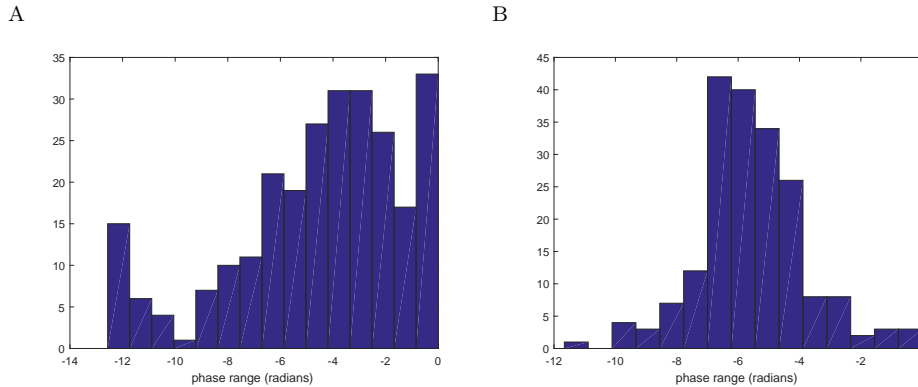


Fig. 3.4: Distribution of phase ranges for place fields on day seven of the experiment **A**, one cluster place fields. **B**, two cluster place fields.

For each place field that was grouped into two clusters, the spikes within each cluster do not show a large phase range. This is likely, in part, due to the clustering algorithm itself. The *K*-means clustering algorithm tends to cluster data into spherical components, so many of the points that do not clearly fall into one of the two clusters are clustered so that the resulting clusters are more spherical. Therefore, it is unlikely that the average slope of these clusters will give an accurate description of the true slope of the points belonging to that component.

**3.5. Results for Novel Environment.** On day one of the experiment, the track and environment were novel to the rat. On this day, the map of the environment, which is retrieved on most subsequent runs along the track, is formed [22, 26]. Therefore, the properties of the place fields may be slightly different for the novel environment compared to the subsequent trials. If the memories of the rats are formed in a different way than they are stored and later retrieved, there may be different properties of the place field structures on the first day of the experiment as opposed to subsequent days of the experiment. The same analysis as described in the previous sections was run on the day one data to compare the effects of a novel environment on the place field properties.

The clustering analysis gave very similar results as on day seven. Once again, most place fields were classified as either one or two clusters, with a few place fields being classified as having more clusters. The number of clusters is described in Table 3.2. Once again, for all place fields classified as having two clusters, the two clusters tend to be of roughly equal sizes.

Of those place fields with two clusters, the two clusters still tended to be nearly anti-phase from one another on day one of the experiment. The difference between the center of mass of these two clusters is  $2.89 \pm 0.42$  radians for all place fields described by two clusters. Once again, there was no difference in the young and old rat populations.

Number of Clusters	1	2	3	> 3	Number of Place Fields
All place fields	248	182	49	24	504
Old rats	106	98	28	18	250
Young rats	142	84	21	7	254

Table 3.2: The number of place fields clustered into 1,2,3, and more than 3 clusters respectively on day one of the experiment. The breakdown of distribution of clusters for all place fields, all place fields of old rats, and all place fields of young rats respectively

The phase range was similar between day one and day seven of the experiment. Both the phase range over all the place fields and the phase range over the one and two cluster place fields gave similar results with no statistically-significant difference between the days. Once again, there was no difference between the young and old rat populations on day one.

Overall, based on these measures, there was no difference between the place field properties on the first day of the experiment as compared to the subsequent days of the experiment. This could in part be because there were two sessions each day, with many runs along the track. Depending on how many runs it took to form the place field maps of the environment, it could be that most of these spikes came from runs in which the map was being retrieved rather than newly formed. Therefore, any properties that might have been slightly different about the formation of the map over the first several runs of the track were lost when averaging out over all runs of the track for that day.

**4. Discussion.** There is some evidence in the data that might suggest that phase precession is a process driven by two distinct components. One particularly interesting feature of this data is those place fields which show no phase precession but only have spikes over a very limited range of theta phases, such as in Figure 3.1B. This behavior is fairly common in certain rats, although there does not appear to be any correlation between the age of rats and the prevalence of these types of place fields. Additionally, the range of theta phases for these place fields can either be centered around a high phase or a low phase. If the firing of CA1 cells is driven by both input from CA3 and EC3, place fields that show firing over only a limited range of theta phase could be CA1 cells driven by only one of these two inputs. The limited range of theta phase, either high or low, could be a result of which of these two inputs is driving the firing. Additionally, the clustering revealed that many of the place fields were well described by a distinct transition from a firing at a high theta phase to firing at a low theta phase. This distinct transition could be a result of a change in the inputs causing the CA1 cells to fire.

Based on the measures considered here, there appears to be no significant difference in CA1 place field spiking between old and young rats. This could imply that while the number of functional synapses from CA3 decreases in older rats, this does not have a large impact on the excitatory input CA1 place cells receive from CA3 [5, 21]. Also, since little is known about the connections between EC3 and CA1 in aged rats [5], it is possible that these connections also weaken with age and thus the balance of inputs to CA3 does not change in aged rats, giving rise to similar properties in both young and old rat place field data. Conversely, if CA1 place cell firing were driven by some mechanism other than dual input from CA3 and EC3, changes in CA3 connectivity would not impact the place cell firing of CA1 cells. This could also explain the lack of differences between CA1 place cell firing in the young and old rat populations.

In general, the  $K$ -means clustering algorithm is able to distinguish between place fields that show a distinct transition between a firing at a high theta phase to a firing at a low

theta phase and those place fields that do not. However, there are a few features of this algorithm that introduce some bias in the clustering. First, the  $K$ -means algorithm is highly dependent on the relative scaling of the two variables [7]. We have scaled the data in such a way that the distance measure should not be biased towards either phase or position. However, a different scaling of the data, with a different justification, would lead to very different results in the clustering. Additionally, the  $K$ -means algorithm tends to find clusters that are of approximately equal size [7]. Looking at the resultant clusters of the place fields, roughly equal size clusters tend to be reasonable. However, there are cases where one cluster is obviously smaller than the other, so the algorithm will tend to group the larger cluster into two clusters. This makes it slightly harder to classify all of the place fields based just on the number of clusters. Also, the algorithm is not robust to noise [7]. If there is a significant amount of noise in the data, which is prevalent in these data sets, the algorithm will often find three or more clusters to try to account for the noise in the data. This makes it very difficult to classify the behavior of place fields with more than two clusters. They could either be two distinct groups of greatly different sizes or they could be of either one or two distinct groups, but with noise that is being captured in the other clusters.

A different clustering algorithm might be able to classify the properties of the place field data. The difficulty, however, is that the data is defined on a cylinder. This leads to different distance measures in the two directions of the data, which can create difficulties in the clustering algorithm. Clustering the data based on some underlying probability distribution would mitigate many of the above difficulties; however, there are no common probability distributions which are defined on the unit cylinder. Other common period distribution such as the von Mises distribution assume all data is periodic. If the data is unwrapped to be defined on a plane, a more common probability distribution could be used, but any unwrapping of the data still creates artificial boundaries which could cause a bias in the clustering. The data could also be redefined where the place field is replicated over many theta periods. This would eliminate the issues of the data being periodic in the theta direction for all but the first and last repetitions of the place field. Then the clustering of the data in one of the middle replications of the place field could be taken to be the true clustering of the data [28]. However, this method often does not work well, particularly in the presence of noise. Noise at the entrance or exit of the place field is often grouped into one cluster across all place fields, which makes it difficult to determine the number of clusters for just one replication of the data. Also, with many place fields that do not show two distinct clusters, this method of clustering will often not give consistent clustering across all replications of the place field, making it impossible to determine the appropriate number of clusters for that place field.

While the arctangent fit was not able to accurately capture the behavior of the transition between the two components of the place field, a way to describe this transition could provide useful insights to underlying structure of the phase precession. Where the point of transition occurs and how quickly the data transitions between these two components could provide useful information about which of the two components is dominant and what the interaction between these two components is. A better fit would be some function which tended to a linear function, not simply a line with zero slope, on either side of this transition. This would prevent the common issue we saw with the inflection point not accurately capturing the transition due to the spikes on either side not tending to a single phase. This kind of a fit would also be able to provide more information about the behavior of the spikes on either side of the transition point than a simple arctangent fit. The slope of the two components may contain useful information for understanding the underlying mechanism driving CA1 firing. As seen in the phase range analysis, simply fitting a line to both of the clusters in all two-cluster place fields did not give meaningful results of the slope of those components

due to the clustering algorithm. A more sophisticated measure would be needed to classify all spikes that have approximately the same trend, which could then be used to determine the change in slope between the two components.

**5. Conclusions.** The  $K$ -means clustering was able to distinguish between place fields that showed a clear transition from spiking at a high theta phase to spiking at a low theta phase and those that do not. However, of those place fields which did not show this clear transition between a firing at a high and low theta phase, some showed a linear phase precession, while others showed very little phase precession. The difference between these firing patterns is not captured by the clustering analysis. However, some of the differences between these firing patterns is captured in the phase range analysis. For place fields with one cluster, there is a much larger spread of phase ranges as compared to those place fields with two clusters. This spread of phase ranges helps to quantify the different behaviors of phase precession for one cluster place fields, those which show linear phase precession over the entire range of theta, and those which do not spike over a wide range of theta phases.

While a way to quantify the transition between the two components would provide useful insight to the difference in place field firing patterns, a simple arctangent fit was not able to fully capture the behavior of the place field and thus did not usually well capture that transition. Further exploration on quantifying the way in which the spikes transition from high theta phases to low theta phases could help further explain the underlying mechanism of what causes the place cell phase precession.

Overall, there was some evidence to suggest that there is a distinct change in firing patterns in CA1 place cells, which could be caused by a shift in the underlying mechanism causing the CA1 cells to fire. However, we did not find that the age of the rat had a significant impact on the firing pattern of the place cells.

## REFERENCES

- [1] O. J. AHMED AND M. R. MEHTA, The Hippocampal Rate Code: Anatomy, Physiology and Theory, *Trends in Neurosciences*, 32 (2009), pp. 329 – 338.
- [2] P. BERENS, CircStat: A MATLAB Toolbox for Circular Statistics, *Journal of Statistical Software*, 31 (2009).
- [3] V. H. BRUN, S. LEUTGEB, H. Q. WU, R. SCHWARCZ, M. P. WITTER, E. I. MOSER, AND M. B. MOSER, Impaired Spatial Representation in CA1 after Lesion of Direct Input from Entorhinal Cortex, *Neuron*, 57 (2008), pp. 290 – 302.
- [4] V. H. BRUN, M. K. OTNAESS, S. MOLDEN, H.-A. STEFFENACH, M. P. WITTER, M.-B. MOSER, AND E. I. MOSER, Place Cells and Place Recognition Maintained by Direct Entorhinal-Hippocampal Circuitry, *Science*, 269 (2002), pp. 2243–2246.
- [5] S. N. BURKE AND C. A. BARNES, Senescent Synapses and Hippocampal Circuit Dynamics, *Trends in Neurosciences*, 33 (2010), pp. 153 – 161.
- [6] F. S. CHANCE, Hippocampal Phase Precession from Dual Input Components, *The Journal of Neuroscience*, 32 (2012), pp. 16693–16703.
- [7] A. K. JAIN, Data Clustering: 50 Years beyond K-means, *Pattern Recognition Letters*, 31 (2010), pp. 651–666.
- [8] A. K. JAIN AND R. C. DUBES, *Algorithms for Clustering Data*, Prentice-Hall, 1988.
- [9] R. KEMPTER, C. LEIBOLD, G. BUZSAKI, K. DIBA, AND R. SCHMIDT, Quantifying Circular-Linear Associations: Hippocampal Phase Precession, *Journal of Neuroscience Methods*, 207 (2012), pp. 113–124.
- [10] B. L. MCNAUGHTON, C. A. BARNES, AND J. O’KEEFE, The Contributions of Position, Direction, and Velocity to Single Unit activity in the Hippocampus of Freely-Moving Rats, *Experimental Brain Research*, 52 (1983), pp. 42 – 49.
- [11] K. MIZUSEKI, S. ROYER, K. DIBA, AND G. BUZSAKI, Activity Dynamics and Behavioral Correlates of CA3 and CA1 Hippocampal Pyramidal Neurons, *Hippocampus*, 22 (2012), pp. 1659–1680.
- [12] K. MIZUSEKI, A. SIROTA, E. PASTALKOVA, AND G. BUZSAKI, Theta Oscillations Provide Temporal Windows for Local Circuit Computation in the Entorhinal-Hippocampal Loop, *Neuron*, 64 (2009), pp. 267 – 280.



- [13] S. M. MONTGOMERY, M. I. BETANCUR, AND G. BUZSAKI, Behavior-Dependent Coordination of Multiple Theta Dipoles in the Hippocampus, *The Journal of Neuroscience*, 29 (2009), pp. 1381 – 1394.
- [14] M. MOSCOVITCH, L. NADEL, G. WINOCUR, A. GILBOA, AND R. S. ROSENBAUM, The Cognitive Neuroscience of Remote Episodic, Semantic and Spatial Memory, *Curr. Opin. Neurobiol.*, 16 (2006), pp. 179–190.
- [15] T. NAKASHIBA, J. Z. YOUNG, T. J. MCHUGH, D. L. BUHL, AND S. TONEGAWA, Transgenic Inhibition of Synaptic Transmission Reveals Role of CA3 Output in Hippocampal Learning, *Science*, 319 (2008), pp. 1260–1264.
- [16] J. O'KEEFE, Place Units in the Hippocampus of the Freely Moving Rat, *Experimental Neurology*, 51 (1976), pp. 78–109.
- [17] J. O'KEEFE AND J. DOSTROVSKY, The Hippocampus as a Spatial Map. Preliminary Evidence from Unit Activity in the Freely-Moving Rat, *Brain Res*, 34 (1971), pp. 171–175.
- [18] J. O'KEEFE AND L. NADEL, *The Hippocampus as a Cognitive Map*, Oxford: Oxford Up, 1978.
- [19] J. O'KEEFE AND M. L. RECCE, Phase Relationship between Hippocampal Place Units and the EEG Theta Rhythm, *Hippocampus*, 3 (1993), pp. 317–330.
- [20] D. PELLEG AND A. W. MOORE, X-means: Extending K-means with Efficient Estimation of the Number of Clusters, *ICML '00 Proceedings of the Seventeenth International Conference on Machine Learning*, (2000), pp. 727–734.
- [21] E. S. ROSENZWEIG AND C. A. BARNES, Impact of Aging of Hippocampal Function: Plasticity, Network Dynamics, and Cognition, *Progress in Neurobiology*, 69 (2003), pp. 143–179.
- [22] L. A. SCHIMANSKI, P. LIPA, AND C. A. BARNES, Tracking the Course of Hippocampal Representations during Learning: When Is the Map Required?, *The Journal of Neuroscience*, 33 (2013), pp. 3094 – 3106.
- [23] R. SCHMIDT, K. DIBA, C. LEIBOLD, D. SCHMITZ, G. BUZSAKI, AND R. KEMPTER, Single Trial Phase Precession in the Hippocampus, *The Journal of Neuroscience*, 29 (2009), pp. 13232–13241.
- [24] J. SHEN, C. A. BARNES, B. L. MCNAUGHTON, W. E. SKAGGS, AND K. L. WEAVER, The Effect of Aging on Experience-Dependent Plasticity of Hippocampal Place Cells, *The Journal of Neuroscience*, 17 (1997), pp. 6769 – 6782.
- [25] W. E. SKAGGS, B. L. MCNAUGHTON, M. A. WILSON, AND C. A. BARNES, Theta Phase Precession in Hippocampal Neuronal Populations and the Compression of Temporal Sequences, *Hippocampus*, 6 (1996), pp. 149–172.
- [26] L. T. THOMPSON AND P. J. BEST, Long-Term Stability of the Place-Field Activity of Single Units Recorded from the Dorsal Hippocampus of Freely Behaving Rats, *Brain Research*, 509 (1990), pp. 299–308.
- [27] C. H. VANDERWOLF, Hippocampal Electrical Activity and Voluntary Movement in the Rat, *Electroencephalography and Clinical Neurophysiology*, 26 (1969), pp. 407 – 418.
- [28] Y. YAMAGUCHI, Y. AOTA, B. L. MCNAUGHTON, AND P. LIPA, Bimodality of Theta Phase Precession in Hippocampus Place Cells in Freely Running Rats, *Journal of Neurophysiology*, 87 (2002), pp. 2629–2642.

## CONSTRAINING THERMAL OSCILLATIONS IN MOLECULAR DYNAMIC SIMULATIONS

RACHAEL KELLER\*, STEWART SILLING†, AND QIANG DU‡

**Abstract.** Peridynamics models continuum mechanics where integral equations are considered instead of traditional differential equations. Peridynamic theory has been shown to significantly overlap with molecular dynamics, motivating the casting of molecular dynamics (MD) simulations in the peridynamic framework. Doing so would allow coarse-graining, reducing the scale of the simulations to more computationally reasonable sizes. Peridynamics has been used thus far to model MD with static forces, and this report explores a method for incorporating time-dependent thermal oscillations on the fine scale.

**1. Introduction.** Molecular Dynamic (MD) simulations describe the behavior of many atoms interacting together— a computationally expensive task. Coarse-graining scales the number of degrees of freedom from many to few. With many common properties to MD, peridynamics provides a framework for modeling MD as a continuum. Such work has been done in the static case [6]. This project explores incorporating more general physics at the fine level. Thermal forcing is applied at the atomic level, and an additional force of constraint is added in to capture the net effect on a bin of atoms. This paper describes the framework of the problem, presents the constrained form and a method to solve it, and reports on numerical experimentation.

**1.1. Peridynamics.** Bond-based peridynamic theory is a powerful application of the traditional spring-mass problem in a broader setting, first presented in [8]. Assuming that the interactions among an n-body system have some locally compact behavior within the whole space, peridynamics reformulates the equations of motion using a linear approximation to the potential and integrating spatially over the interaction space, as determined by the peridynamic *horizon*. This subsection details the peridynamic framework and its consistency with MD.

In peridynamics, the trajectory for an atom  $\mathbf{x} \in \{\mathbf{x}_i\}_{i=0}^n$  is given by

$$\rho \ddot{\mathbf{u}} = \int_{H_{\mathbf{x}}} \mathbf{f}(\mathbf{u}(\mathbf{x}', t) - \mathbf{u}(\mathbf{x}, t), \mathbf{x}' - \mathbf{x}) dV_{\mathbf{x}'} + \mathbf{b}(\mathbf{x}, t), \quad (1.1)$$

where  $H_{\mathbf{x}}$  is the horizon of  $\mathbf{x}$ ,  $\mathbf{u}$  is the displacement field,  $\rho$  is the mass density of the body at  $\mathbf{x}$ , and  $\mathbf{f}$  is the pairwise force density function [11]. Consider the following change of variables.

$$\boldsymbol{\xi} = \mathbf{x}' - \mathbf{x}; \quad \boldsymbol{\eta} = \mathbf{u}(\mathbf{x}', t) - \mathbf{u}(\mathbf{x}, t),$$

so that  $\boldsymbol{\xi}$  describes the relative position of atoms and  $\boldsymbol{\eta}$  describes their relative displacements. The horizon of  $\mathbf{x}$  is defined as the positive  $\delta$  such that

$$|\boldsymbol{\xi}| > \delta \implies \mathbf{f}(\boldsymbol{\eta}, \boldsymbol{\xi}) = \mathbf{0} \quad \forall \boldsymbol{\eta}.$$

It is assumed that such a  $\delta$  exists for the given problem. In MD the time evolution of a given atom is determined by the interatomic forces between atoms within a cut-off radius, so MD is consistent with peridynamic theory in this respect.

Further, the properties assumed for microelastic peridynamic materials is consistent with MD. A microelastic peridynamic material has a pairwise force satisfying the following.

---

\*Applied Physics and Applied Mathematics, Columbia University, rachael.keller@columbia.edu

†Sandia National Laboratories, sasilli@sandia.gov

‡Applied Physics and Applied Mathematics, Columbia University, qd2125@columbia.edu ,

First, the net work on a fixed particle due to any interaction with another along a fixed, closed curve is zero. More precisely,

$$\oint_{\Gamma} \mathbf{f}(\boldsymbol{\eta}, \boldsymbol{\xi}) \cdot d\boldsymbol{\eta} = 0, \quad \text{for any closed contour } \Gamma, \quad \forall \boldsymbol{\xi} \neq 0.$$

Next, the forcing function  $\mathbf{f}$  is required to be such that

$$\mathbf{f}(-\boldsymbol{\eta}, -\boldsymbol{\xi}) = -\mathbf{f}(\boldsymbol{\eta}, \boldsymbol{\xi}) \quad \text{and} \quad (\boldsymbol{\xi} + \boldsymbol{\eta}) \times \mathbf{f}(\boldsymbol{\eta}, \boldsymbol{\xi}) = 0 \quad \forall \boldsymbol{\eta}, \boldsymbol{\xi},$$

where the former assures conservation of linear momentum and the latter ensures conservation of angular momentum. Further, we assume  $f$  is of the form

$$\mathbf{f}(\boldsymbol{\eta}, \boldsymbol{\xi}) = \frac{\partial w}{\partial \boldsymbol{\eta}}(\boldsymbol{\eta}, \boldsymbol{\xi}) \quad \forall \boldsymbol{\xi}, \boldsymbol{\eta}.$$

That is,  $\mathbf{f}$  is derivable from a scalar, differentiable function  $w$ , the microelastic pair-wise potential. The Lennard-Jones potential for MD simulations satisfies these properties. Using a first-order, linear approximation,  $\mathbf{f}$  can be approximated as

$$\mathbf{f}(\boldsymbol{\eta}, \boldsymbol{\xi}) = \mathbf{C}(\boldsymbol{\xi})\boldsymbol{\eta} \quad \forall \boldsymbol{\eta}, \boldsymbol{\xi},$$

where  $\mathbf{C}(\boldsymbol{\eta}) = \frac{\partial \mathbf{f}}{\partial \boldsymbol{\eta}}(\mathbf{0}, \boldsymbol{\xi}) \quad \forall \boldsymbol{\xi}$ . Equation (1.1) then becomes

$$\rho \ddot{\mathbf{u}} = \int_{H_x} \mathbf{C}(\mathbf{x}, \mathbf{x}')(\mathbf{u}(\mathbf{x}') - \mathbf{u}(\mathbf{x}))dV_{\mathbf{x}'} + \mathbf{b}(\mathbf{x}), \quad (1.2)$$

so that the equilibrium equation is

$$\int_{H_x} \mathbf{C}(\mathbf{x}, \mathbf{x}')(\mathbf{u}(\mathbf{x}') - \mathbf{u}(\mathbf{x}))dV_{\mathbf{x}'} + \mathbf{b}(\mathbf{x}) = 0 \quad \forall \mathbf{x} \in \mathcal{B}.$$

The spatial integration yields the salient feature of peridynamics— discontinuities in the form of bond breakage or crack growth evolve naturally in the system.

**1.1.1. Clustering atoms.** Suppose the region  $\mathcal{B}$  were partitioned into  $K$  subregions, so that  $\bigcup_{k=1}^K \mathcal{B}^k = \mathcal{B}$ . Define a characteristic function  $\phi$  by

$$\phi^k(\mathbf{x}) = \begin{cases} 1/\text{vol}(\mathcal{B}^k) & \mathbf{x} \in \mathcal{B}^k \\ 0 & \text{otherwise,} \end{cases} \quad (1.3)$$

for each  $k \in \{1, 2, \dots, K\}$ , so that the normalization

$$\int_{\mathcal{B}^k} \phi^k(\mathbf{x})d\mathbf{x} = 1 \quad \forall k \in \{1, 2, \dots, K\}$$

holds. Then for any displacement field  $\mathbf{u}$ , the  $K$  average displacement vectors are defined by

$$\int_{\mathcal{B}} \phi^k(\mathbf{x})\mathbf{u}(\mathbf{x})d\mathbf{x}, \quad k = 1, 2, \dots, K.$$

**1.2. Constrained Minimization.** Consider constraining the atomic displacements of a bin of atoms  $\mathcal{B}^k$  to obtain a coarse-grained approximation  $v^k$ . In 1D

$$v^k - \int_{\mathcal{B}} \phi^k(x) u(x) dx = 0, \text{ for each } k = 1, 2, \dots, K,$$

where  $v^k$  is prescribed average atomic displacement for the set of atoms in bin  $\mathcal{B}^k$ . The constrained potential energy functional  $\mathcal{J}$  is defined

$$\mathcal{J} = \Phi - \sum_{k=1}^K \lambda^k \left( \int_{\mathcal{B}} \phi^k(x) u(x) dx - v^k \right), \quad (1.4)$$

where  $\Phi$  is the potential energy of the system,

$$\Phi = \int_{\mathcal{B}} \left( \left[ \frac{1}{4} \int_{H_x} C(x, x') (u(x') - u(x))^2 dx' \right] - b(x) u(x) \right) dx.$$

In this paper, it is assumed  $b(x) = 0$ . Requiring  $\mathcal{J}$  be stationary and taking the first variations with respect to  $\lambda^k$  and  $u$ , the following is obtained [10].

$$\int_{\mathcal{B}} \phi^k(x) u(x) dx - v^k = 0. \quad (1.5)$$

$$\int_{H_x} C(x, x') (u(x') - u(x)) dV_{x'} + \sum_{k=1}^K \lambda^k \phi^k = 0. \quad (1.6)$$

Let  $C^0$  denote the finest-level (level 0) approximation of  $f$ , with  $\{u_i^0\}$  representing the displacement of the atoms,  $N^0$  and  $K^0$  the total number of atoms and bins, respectively, and  $\mathcal{B}_0^k$  the bins. Using midpoint quadrature, Equation (1.6) is

$$\sum_{j \in H_j} C_{ij}^0 (u_j^0 - u_i^0) + \sum_{k=1}^{K^0} \lambda_k^0 \phi_i^{k,0} = 0, \quad i = 1, 2, \dots, N. \quad (1.7)$$

Then we have an  $(N^0 + K^0) \times (N^0 + K^0)$  system of equations to solve for the unknowns

$$\{u_1^0, u_2^0, \dots, u_{N^0}^0, \lambda_1^0, \lambda_2^0, \dots, \lambda_{K^0}^0\}.$$

These equations form the Karush-Kuhn-Tucker conditions of the system.

$$\begin{bmatrix} C^0 & \phi \\ \phi^T & 0 \end{bmatrix} \begin{bmatrix} u^0 \\ \lambda^0 \end{bmatrix} = \begin{bmatrix} 0 \\ v \end{bmatrix} \quad (1.8)$$

Assuming the  $(N^0 + K^0) \times (N^0 + K^0)$  matrix is nonsingular, Equation (1.8) implies

$$\begin{bmatrix} A^* & R \\ R^T & -C^1 \end{bmatrix} \begin{bmatrix} 0 \\ v \end{bmatrix} = \begin{bmatrix} u \\ \lambda_0 \end{bmatrix}, \quad (1.9)$$

so that the  $\lambda^0$  are the corresponding forces for the next level,  $C^1$ .

**1.3. Problem Statement.** The goal is to determine the forces  $\lambda$  that characterize the additional forcing from the added thermal oscillations for each bin of atoms. The forces acting on a given node are assumed to be the pair-wise forces given by the Lennard-Jones potential, the Lagrange multiplier forces  $\lambda\phi$ , and a thermal source term  $Q(t)$ . The proposed solution method uses a root-solving technique along with Verlet and Simpson approximation to obtain the Lagrange multipliers.

## 2. Time-dependent Dynamics.

**2.1. Dynamic Problem Formulation.** Consider a more general MD model incorporating time-dependent thermal oscillations [10] with a thermal forcing function  $Q(t)$ . Suppose a system has  $N^0$  atoms, each represented by  $u_i^0 = u^0(x_i, t)$ , where  $x_i$  represents the position of atom, or node,  $i$  in the initial configuration and  $u^0(x_i, t)$  the displacement of node  $i$ . Further suppose the system is divided into  $K^0$  bins,  $\mathcal{B}_0^k$  so that each of the  $N^0$  atoms belongs to one and only one bin. It assumed that the atoms are clustered according to nearest-neighbors. Given a set of prescribed mean atomic displacements for each bin,  $v^k$ ,  $k \in \{1, 2, \dots, K\}$ , the constraint imposed is

$$\frac{1}{\tau} \int_0^\tau \int_{B^k} (\phi(x)u(x, t)) dx dt - v^k = 0.$$

Spatial discretization using midpoint rule on  $\phi(x)u(x, t)$  yields the approximation

$$\frac{1}{\tau} \int_0^\tau \sum_{i \in \mathcal{B}^k} \phi_i^k u_i(t) dt - v^k = 0,$$

where now  $\sum_{i \in \mathcal{B}^k} \phi_i^k = 1$ . For node  $i$ ,

$$m\ddot{u}_i(x, t) = \int_{H_i} C_{ij}(u_j - u_i) + Q(t),$$

where  $C_{ij}$  form the  $n \times n$  matrix of linear approximations to the potential  $C^0$  and  $H_i$  is the discrete-form horizon of node  $i$ . Since  $Q(t)$  is applied at a specific time step and is purely temporal, the properties of  $C^0$  are preserved. From Equation (1.6),

$$m\ddot{u}_i(x_i, t) = \int_{j \in H_i} C_{ij}^0(u_j - u_i) du_j + \sum_{j \in H_i, B^k} \lambda_k^0 \phi_j^{k,0} + Q(t),$$

and thus, following (1.7),

$$m\ddot{u}_i(x_i, t) = \sum_{j \in H_j} C_{ij}^0(u_j^0 - u_i^0) + \sum_{j \in H_i, B_k^0} \lambda_k^0 \phi_j^{k,0} + Q(t). \quad (2.1)$$

**2.2. Proposed Solution.** Let  $k \in \{1, 2, \dots, K\}$ . The goal of the constrained problem is to solve for the Lagrange multipliers  $\lambda_k^0$  and the displacements  $u_i^0(t)$  so that each bin has a mean atomic displacement  $v^k$ . That is,

$$\frac{1}{\tau} \int_0^\tau \sum_{i \in \mathcal{B}_k^0} \phi_i^k u_i(t) dt - v^k = 0.$$

Letting  $\mathbf{v} = (v^1, v^2, \dots, v^K)$ , the method is to solve for the root of the equation

$$G(\boldsymbol{\lambda}) = \mathbf{v} - g(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}, \phi),$$

where  $g(\boldsymbol{\lambda})$  is a vector-valued function depending on Verlet integration  $VV()$  so that

$$g^k(\lambda) = \frac{1}{\tau} \int_0^\tau \phi^k \cdot VV(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}, \phi),$$

with acceleration for the Verlet method as shown in Equation (2.1).

**2.3. Molecular Dynamics.** In MD models the forcing function is only calculated within a specified cut-off radius, analogous to the peridynamic model's horizon. This commonality motivates further study and integration of molecular dynamic simulations in the peridynamic framework. The first formulations of such problems were presented in [6], and the converse approach, leading to a casting of peridynamics as an upscaling of MD, was demonstrated by Seleson, Parks, Gunzburger, and Lehoucq in [7]. In both papers, the Lennard Jones (LJ) inter-molecular potential function is considered, as is done in this work. The LJ potential is defined

$$f(r) = 4\epsilon \left[ \left( \frac{\sigma}{r} \right)^{12} - \left( \frac{\sigma}{r} \right)^6 \right], \quad (2.2)$$

where  $r$  is the distance  $x - x'$ , for  $x, x' \in \{x_i\}_{i=0}^n$ ,  $\sigma$  a length-scale parameter, and  $\epsilon$  an energy-scale parameter [3]. The parameters  $\sigma$  and  $\epsilon$  are inherent to the atoms under consideration, and the potential minimum occurs at  $r_{min} = 2^{\frac{1}{6}}\sigma$ , at which point  $V_{min} = -\epsilon$ . The value  $r_{min}$  is the critical point of the system.

For the peridynamic framework, the linear approximation to the Lennard-Jones potential is considered. That is,  $C(r) = \frac{\partial f}{\partial r}(r)$ , where

$$\frac{\partial f}{\partial r}(r) = -24 \frac{\epsilon}{\sigma^2} \left[ 7 \left( \frac{\sigma}{r} \right)^8 + 26 \left( \frac{\sigma}{r} \right)^{14} \right].$$

The choice of  $r$  is taken from the equilibrium configuration of the atoms,  $r = r_{min}$ .

### 3. Numerical Experiments.

**3.1. Truncation Error.** To understand how the algorithm performs, the order of error associated with applying Simpson's rule to the already-approximated,  $\mathcal{O}(h^2)$  displacements is considered. By considering the local analysis for Simpson's rule, the temporal integration is shown to be consistent with the Verlet integration. Simpson's scheme uses a three-point Newton-Coates formula

$$\int_a^b f(x(t), t) dt \approx P_3(x, t) = \sum_{j=1}^3 w_j f(x(t_j), t_j),$$

where  $w_j$  are the weights associated with the Lagrangian basis. Thus, the error  $E$  is

$$E = \int_a^b f(x(t), t) - P_3(x(t), t) dt.$$

Let  $(a, b)$  be any interval  $(t_{n-1}, t_{n+1})$ . For a given time step  $t_n$ ,

$$\begin{aligned} E_n &= \int_{t_{n-1}}^{t_{n+1}} [f(x(t), t) - P_3(x(t), t)] dt. \\ &= \int_{t_{n-1}}^{t_{n+1}} f(x(t), t) - \frac{h}{6} (P_3(x(t_{n-1}), t_{n-1}) + 4P_3(x(t_n), t_n) + P_3(x(t_{n+1}), t_{n+1})) dt. \end{aligned}$$

Now,  $f(x, t) = u(x, t)$ , and

$$\int_a^b u(x(t), t) dt = \sum_{j=1}^3 w_j u^h(x(t_j), t_j),$$

where  $u^h(x(t_j), t_j)$  are approximated values of  $u$  obtained by Verlet integration. Thus

$$P_3(x(t), t) = \sum_{j=1}^3 [w_j u^h(x(t_j), t_j) + \mathcal{O}(h^2)].$$

Assuming proper continuity, the Taylor approximation for general  $f(x, t)$  at  $t_{n+1}$  is

$$\begin{aligned} f(x(t_{n+1}), t_{n+1}) &= f(x(t_n), t_n) + h(f_t(x(t_n), t_n)\dot{x}(t_n)) \\ &\quad + \frac{h^2}{2} (f_{tt}(x(t_n), t_n)\dot{x}(t_n)^2 + f_t(x(t_n), t_n)\ddot{x}(t_n)) \\ &\quad + \frac{h^3}{6} (f_{ttt}(x(t_n), t_n)\dot{x}(t_n)^3 + 2f_{tt}(x(t_n), t_n)\dot{x}(t_n)\ddot{x}(t_n)) \\ &\quad + \frac{h^3}{6} \left( f_{tt}(x(t_n), t_n)\dot{x}(t_n)\ddot{x}(t_n) + f_t(x(t_n), t_n)\frac{dx^3}{dt^3}(t_n) \right), \end{aligned}$$

and a similar result holds for  $f(x(t_{n-1}), t_{n-1})$  by replacing  $h$  with  $(-h)$ . Thus

$$\begin{aligned} E_n &= \int_{t_{n-1}}^{t_{n+1}} f(x, t) - \frac{h}{6} (6f(x(t_n), t_n) + (h-h)f_t(x(t_n), t_n)\dot{x}(t_n) \\ &\quad + h^2 [f_{tt}(x(t_n), t_n)\dot{x}(t_n)^2 + f_t(x(t_n), t_n)\ddot{x}(t_n)]) + \mathcal{O}(h^3)) \\ &= \int_{t_{n-1}}^{t_{n+1}} \frac{h}{6} (h^2 [f_{tt}(x(t_n), t_n)\dot{x}(t_n)^2 + f_t(x(t_n), t_n)\ddot{x}(t_n)] + \mathcal{O}(h^3)) dt \end{aligned}$$

Therefore, if  $f \in C^2 \times C^2$  and  $x \in C^2$ , then Simpson's approximation has  $\mathcal{O}(h^3)$  local error, so that globally the method is  $\mathcal{O}(h^2)$ , consistent with Verlet integration. One further order of truncation can be obtained by adding in an interpolation point by leveraging midpoint rule with the Simpson formula and assuming further continuity. However, this analysis requires less of  $f$  and  $x$  and is enough to show that Simpson integration does not pollute the calculations further.

**3.2. Stability Analysis.** A critical component of numerical schemes is that they are stable. To ensure stability of the Verlet integration, consider the primary forcing on the adiabatic problem [9]. The force on node  $i$  is given by Equation (1.2),

$$m\ddot{u}_i = \int_{H_{x_i}} C(x_i, x_j)(u(x_j) - u(x_i))dV_{x_j}, \quad (3.1)$$

where  $C$  is the micromodulus function for node  $i$  and  $m$  its mass. Letting  $H_i$  be the indices of  $x_j \in H_{x_i}$  and discretizing,

$$m_i\ddot{u}_i = \sum_{j \in H_i, j \neq i} C_{i,j}u_j^n V_{x_j} - \left( \sum_{j \in H_i} C_{ij} \right) u_i^n V_{x_j}, \quad (3.2)$$

Assuming uniform mesh  $h = \Delta t$  and letting  $C_{ij} = 0$  if  $x_j \notin H_{x_i}$ ,

$$\frac{m_i(u_i^{n+1} - 2u_i^n + u_i^{n-1}))}{h^2} = \sum_{j \in H_i, j \neq i} C_{i,j}u_j^n V_{x_j} - \left( \sum_{j \in H_i} C_{ij} \right) u_i^n V_{x_j}. \quad (3.3)$$

Thus, the system has a mass matrix  $A$  corresponding to the second-order centered-difference scheme applied to  $u$  and a stiffness matrix  $C$  corresponding to the linear approximation of the forcing term. Due to the  $\delta$ -horizon, the stiffness matrix will be banded and diagonally-dominant. With the LJ potential, when the bonded materials are homogeneous this matrix is also symmetric and positive semi-definite.

To understand the stability of the approximation scheme, let  $U_j^n = B_j e^{i\omega x_j} \xi^n$ , where  $B_j$  is a coefficient determined by the molecular properties of the  $j^{th}$  node. Von Neumann stability analysis as in [1] on 3.3 yields the following.

$$\begin{aligned} B_j e^{i\omega x_j} \xi^{n+1} &= e^{i\omega x_j} (2B_j \xi^n - B_j \xi^{n-1}) + \frac{\xi^n h^2}{m_j} \left[ \sum_{k \in H_j, k \neq j} C_{jk} (-B_j e^{i\omega x_j} + B_k e^{i\omega x_k}) \right] \\ \implies \xi^1 &= 2\xi^0 - \xi^{-1} + \frac{h^2}{m_j} \left[ \sum_{k \in H_j, k \neq j} -C_{jk} + \frac{B_k}{B_j} C_{jk} e^{i\omega(x_j - x_k)} \right] \\ \implies \xi^2 - \left( 2 + \frac{h^2}{m_j} \left[ \sum_{k \in H_j, k \neq j} -C_{jk} + C_{jk} \frac{B_k}{B_j} e^{i\omega(x_j - x_k)} \right] \right) \xi + 1 &= 0. \end{aligned}$$

This equation has roots

$$\begin{aligned} \xi &= \frac{2 + \frac{h^2}{m_j} \left( \sum_{k \in H_j, k \neq j} \left( \frac{B_k}{B_j} e^{i\omega(x_j - x_k)} - 1 \right) C_{jk} \right)}{2} \pm \frac{\sqrt{2 + \frac{h^2}{m_j} \left( \sum_{k \in H_j, k \neq j} \left( \frac{B_k}{B_j} e^{i\omega(x_j - x_k)} - 1 \right) C_{jk} \right)} - 4}{2} \\ &= \frac{2 + \frac{h^2}{m_j} \left( \sum_{k \in H_j, k \neq j} \left( \frac{B_k}{B_j} e^{i\omega(x_j - x_k)} - 1 \right) C_{jk} \right)}{2} \pm \frac{\sqrt{-2 + \frac{h^2}{m_j} \left( \sum_{k \in H_j, k \neq j} \left( \frac{B_k}{B_j} e^{i\omega(x_j - x_k)} - 1 \right) C_{jk} \right)}}{2} \end{aligned}$$

The following simplifying assumptions are now made.

1. The material is homogeneous; that is, the atoms are the same, and so the proportional amplitude  $\frac{B_k}{B_j} = 1$ .



2.  $C$  is a symmetric matrix of nonnegative off-diagonal entries.

Now, consider the case when the atoms are at their highest vibrational frequency, when they are exactly out-of-phase. The forcing term is then

$$\frac{h^2}{m} \left( \sum_{k \in H_j, k \neq j} (e^{i\omega(x_j - x_k)} - 1) C_{jk} \right) = \frac{-2h^2}{m} \left( \sum_{k \in H_j, k \neq j} C_{jk} \right).$$

Then,

$$\xi = \left( 1 - \frac{h^2}{m_j} \sum_{k \in H_j, k \neq j} C_{jk} \right) \pm \frac{\sqrt{2 - \frac{2h^2}{m} \sum_{k \in H_j, k \neq j} C_{jk}}}{2} i.$$

To obtain the critical value  $h_c$  so that  $|\xi| = 1$ , set  $\text{Re}(\xi) = 1$ .

$$\begin{aligned} \frac{h_c^2}{m_j} \sum_{k \in H_j, k \neq j} C_{jk} &= 1 \\ \Rightarrow h_c &= \sqrt{\frac{m_j}{\sum_{k \in H_j, k \neq j} C_{jk}}}. \end{aligned}$$

Therefore for all  $h < h_c$ , the growth factor  $|\xi| < 1$ , as desired.

### 3.3. Experiments.

**3.3.1. Summary of Findings.** The system of equations and solution method is found to converge to the correct solution for displacements when the magnitude of the thermal forcing is at or below that of the constraint, and further analysis also suggests that the solvability of the system is dependent on the relation of the order of magnitude of the constraints with the number of atoms in each bin.

The proposed method is a discrete approximation that uses Python's `fsolve`, which is a modified Powell, dogleg optimization technique [2]. The dogleg method is a trust-region method that combines a steepest-descent step with a step toward the Cauchy Point, constrained to a region  $\Delta$ . The model function  $m(p)$  is the quadratic approximation to the function being solved, where the gradient is taken with respect to  $\lambda$  [5]. Now,

$$u_i^n = \frac{u_i^{n+1} + u_i^{n-1}}{2} + \ddot{u}_i^n \frac{h^2}{2} + \mathcal{O}(h^4).$$

Replacing this approximation into the constraint equation,

$$\begin{aligned} f(x, u, v, \lambda, \phi)^k &= v^k - \frac{1}{\tau} \int_0^\tau \phi^k \frac{u_i^{n+1} + u_i^{n-1}}{2} + \ddot{u}_i^n \frac{h^2}{2} + \mathcal{O}(h^4) \\ &= v^k - \frac{1}{\tau} \int_0^\tau \phi^k \frac{u_i^{n+1} + u_i^{n-1}}{2} + \int_{j \in H_i} C_{ij}^0 (u_j - u_i) \\ &\quad + Q(t) + \lambda^k \sum_{j \in H_i, B_{k'}^0} \phi_j^{k'} + \mathcal{O}(h^4). \end{aligned}$$

Now,  $\nabla_{\lambda^{k''}} f^k = \sum_{u_i \in B_0^k, j \in H_i, B_0^{k'}}$   $\phi_j^{k'}$ , and therefore  $\nabla_{\lambda}^2 f^k = 0$ . Thus

$$\begin{aligned} m(p)^k &= v^k - \frac{1}{\tau} \int_0^\tau \phi^k \frac{u_i^{n+1} + u_i^{n-1}}{2} + \int_{j \in H_i} C_{ij}^0 (u_j - u_i) \\ &\quad + Q(t) + \lambda^k \sum_{j \in H_i, B_0^{k'}} \phi_j^{k'} + \mathcal{O}(h^4) + \sum_{j \in H_i, B_0^{k'}} \phi_j^{k'} p, \end{aligned}$$

where  $p = \tau p^C$ , with  $p^C$  the Cauchy Point  $p^C = \Delta \frac{\nabla_{\lambda} f}{\|\nabla_{\lambda} f\|}$ . Since  $\nabla_{\lambda}^2 f^k = 0$ , in this algorithm  $\tau = 1$ . Let  $Q(t) = \gamma \cos(t)$ . Fix  $k \in \{1, 2, \dots, K\}$ . The solver minimizes

$$\begin{aligned} m(p)^k &= v^k - \frac{1}{\tau} \int_0^\tau \frac{u_i^{n+1} - u_i^{n-1}}{2} + \sum_{j \in H_i} C_{ij}^0 (u_j - u_i) \\ &\quad + \gamma \cos(t) dt + \lambda^k \left( 1 + \Delta \frac{1}{\|\nabla_{\lambda} f\|} \right) \sum_{j \in H_i, B_0^{k'}} \phi_j^{k'} dt \end{aligned}$$

Setting  $m(p)^k = 0$ ,

$$\begin{aligned} \frac{\gamma}{\tau} \int_0^\tau \cos(t) dt - v^k &= \int_0^\tau \frac{u_i^{n+1} - u_i^{n-1}}{2} + \sum_{j \in H_i} C_{ij}^0 (u_j - u_i)^n \\ &\quad + \lambda^k \left( 1 + \Delta \frac{1}{\|\nabla_{\lambda} f\|} \right) \sum_{j \in H_i, B_0^{k'}} \phi_j^{k'}. \end{aligned}$$

Bounding cos and applying triangle inequality,

$$\begin{aligned} |\gamma - v^k| &\leq \frac{\gamma}{\tau} \int_0^\tau \cos(t) dt - v^k. \\ \implies |\gamma - v^k| &\leq \frac{1}{\tau} \int_0^\tau \frac{u_i^{n+1} - u_i^{n-1}}{2} + \sum_{j \in H_i} C_{ij}^0 (u_j - u_i)^n \\ &\quad + \lambda^k \left( 1 + \Delta \frac{1}{\|\nabla_{\lambda} f\|} \right) \sum_{j \in H_i, B_0^{k'}} \phi_j^{k'}. \end{aligned}$$

Simpson's method is employed to approximate the temporal integral. Letting

$$\tilde{f}(u, C^0, t) = \frac{u_i^{n+1} - u_i^{n-1}}{2} + \sum_{j \in H_i} C_{ij}^0 (u_j - u_i)^n,$$

$$\frac{1}{\tau} \int_0^\tau \tilde{f}(u, C^0, t) dt = \frac{h}{3\tau} \sum_{j=1}^{N/2} \left[ \tilde{f}(u, C^0, t_{2j-2}) + 4\tilde{f}(u, C^0, t_{2j-1}) + \tilde{f}(u, C^0, t_{2j}) \right] + \mathcal{O}(h^2).$$

The  $u_i$  are  $\mathcal{O}(10^{-9})$ ; the  $C_{ij}^0$  are  $\mathcal{O}(10^{-14})$ ; and  $\tau = Nh$ . Thus, for the level set

$$|\gamma - v^k| = \lambda^k \left( 1 + \Delta \frac{1}{\|\nabla_{\lambda} f\|} \right) \sum_{j \in H_i, B_0^{k'}} \phi_j^{k'} + \text{l.o.t. } (\mathcal{O}(10^{-7}))$$

Consider the parenthetical.  $\nabla_\lambda f \leq \|\nabla_\lambda f\|$ . Now,

$$\|\nabla_\lambda f\| = h^2 \max_k \sum_{i \in \mathcal{B}_0^k, j \in H_i} \phi_j^k.$$

The employed algorithm uses MINPACK, which approximates the trust region size  $\Delta_0 = \text{factor} \|D_0 \tilde{\lambda}_0\|$ , where  $D_0$  is a diagonal matrix, by default the norm of the columns of the Jacobi, and  $\tilde{\lambda}_0$  is the supplied initial point [4]. It can be shown that the Jacobi of this system is the element-wise sum of the components of the nodal basis acting on a particular node; thus each component  $k \in \{1, 2, \dots, K\}$ , of  $\tilde{\lambda}_0$  is scaled by a factor of  $\sum_{j=1}^{10^n} \tilde{\delta}_j 10^{-n}$ , where  $\tilde{\delta}_j < 2\delta$  is the number of nodes acting on atom  $j$ .

If  $\tilde{\lambda}_0 = 0$ , then  $\Delta_0 = \text{factor}$ . In this report default parameters were used, so  $\Delta_0 = 100 \|\tilde{\lambda}_0 D_0\|$ . Recalling the linear problem, the initial value  $\lambda_0$  scales proportionally with  $\bar{v}$  and inversely with  $\phi$ . Fix  $n > 0$ . If  $\bar{v} = 10^{-\alpha}$  and a bin has  $10^n$  atoms, experimentally,  $\lambda_0$  was found to scale as  $10^{-\alpha} 10^n$ . Let  $\|\lambda_0\| = C 10^{-\alpha+n}$ , where  $C \in (0, 10)$ . Suppose each bin has  $10^n$  atoms.

$$\begin{aligned} 100 \frac{\Delta_0}{\|\nabla_\lambda f\|} &= 100 \frac{\|C 10^{-\alpha+n} D_0\|}{\|\nabla_\lambda f\|} \\ &\leq 100 \frac{\|\tilde{\lambda}_0\| \|D_0\|}{\|\nabla_\lambda f\|} \\ &\leq 100 \|\tilde{\lambda}_0\| \\ &= C 10^{2-\alpha+n}. \end{aligned}$$

Thus

$$|\gamma - v^k| \approx (1 + C 10^{2-\alpha+n}) \sum_{j \in H_i, B_0^{k'}} \phi_j^{k'} \lambda_k + \text{l.o.t.},$$

so that

$$\frac{|\gamma - v^k|}{(1 + C 10^{2-\alpha+n}) \sum_{j \in H_i, B_0^{k'}} \phi_j^{k'}} \approx \lambda_k.$$

Thus the ratio

$$\rho_k = \frac{|\gamma - v^k|}{(1 + C 10^{2-\alpha+n}) \sum_{j \in H_i, B_0^{k'}} \phi_j^{k'}}$$

determines the solvability of the system. If  $\rho_k > 1$ , then the problem is ill-conditioned; thus this imposes conditions on the thermal forcing, the constraints, the number of atoms per bin, and the  $\delta$ -horizon.

Consider some examples. Suppose  $n = 1$  and take the first deviation such that  $\gamma > v^k$ .

$$1 + C 10^{2-\alpha+n} = 1 + C 10^{1-\alpha} \approx 1,$$

since  $\alpha > 1$ . If all elements in the nodal horizon  $\delta = 6$  were included,

$$\begin{aligned} \rho_k &= \frac{|10^{-\alpha+1} - 10^{-\alpha}|}{\sum_{i=1}^6 \mathcal{O}(10^{-1})} \\ &\approx \frac{100}{60} = \frac{10}{6}. \end{aligned}$$

Thus  $\rho_k$  indicates this system is poorly conditioned, explaining the behavior observed in the 10-bin systems (Table 3.4) with  $\gamma$  one order of magnitude above the constraint. On the other hand, when  $\gamma$  is of the same order of magnitude as the constraint or one degree less, stable behavior is expected and observed. With the same fixed  $\alpha > 1$ , the associated  $\rho_k$  are

$$\begin{aligned}\rho_k &= \frac{|10^{-\alpha} - 10^{-\alpha}|}{\sum_{i=1}^6 \mathcal{O}(10^{-1})} \\ &= 0,\end{aligned}$$

and

$$\begin{aligned}\rho_k &= \frac{|10^{-\alpha-1} - 10^{-\alpha}|}{\sum_{i=1}^6 \mathcal{O}(10^{-1})} \\ &\approx \frac{10^{-\alpha+1}}{6} \leq 1, \quad (\alpha > 1).\end{aligned}$$

In addition, the dependence on the scale of atoms per cluster can further be elucidated by this  $\rho_k$ . Comparing the results from the (10,4) and the (100,10) systems (Table 3.3), the magnitude of the percent error increases with more atoms per bin, as the ratio would predict. To further verify this relationship, experiments were run with eleven bins of ten atoms; findings are consistent with the result that the dependence is on the number of atoms per bin and not the total number of atoms. The results for particular  $\gamma$  are shown in Table 3.4. Now suppose  $n = 2$ , and let  $\gamma = 10^{-\beta}$ .

$$\begin{aligned}\rho_k &= \frac{|10^\beta - 10^{-\alpha}|}{(1 + C10^{4-\alpha}) \sum_{j \in H_i, B_0^{k'}} \phi_j^{k'}} \\ &\approx \frac{|10^\beta - 10^\alpha|}{10^{4-\alpha} 2\delta 10^{-2}} = \frac{10^{2-\alpha} |10^\beta - 10^{-\alpha}|}{2\delta}\end{aligned}$$

Thus for  $\alpha = 1$ ,  $\rho_k = \frac{10(10^\beta - 10^{-1})}{2\delta}$ , so that  $\beta \leq 1 \implies \rho_k < 1$ . If  $\alpha = 2$ , then  $\beta \leq 0 \implies \rho_k < 1$ . These conditions are consistent with the results, shown in Table 3.1 below.

Table 3.1: Percent Error. 2 Bins of 100 atoms.

Bin $\mathcal{B}^i$ ; $\bar{v} = 1\text{e-}1\text{m}$ .				Bin $\mathcal{B}^i$ ; $\bar{v} = 1\text{e-}2\text{m}$ .			
$i$	$\gamma = 10^{-2}$	$\gamma = 10^{-1}$	$\gamma = 1$	$i$	$\gamma = 10^{-1}$	$\gamma = 1$	$\gamma = 10$
1	-0.0022%	-0.1582%	9.834%	1	-0.1582%	-0.085%	-1299%
2	-0.0022%	-0.1582%	9.834%	2	-0.1582%	-0.085%	7489%

**3.3.2. Set-up and Results.** Arguably the most used element for experimental simulations in molecular dynamics is Argon. Following convention, experiments were performed in 1D on systems of these atoms, setting  $\sigma = 3.405 \text{ \AA}$  and  $\epsilon = 1.654e - 21 K$  [12]. From the stability analysis,  $C = \max \text{eig}(C_{ij})$ ; and  $h = 0.25h_c$ . Simpson's integration is employed for  $N = 100$  time steps, and the initial parameter for the  $\lambda$  is that obtained by solving the static problem, equation (1.8). The same displacement is prescribed across bins. That is,  $v_k = \bar{v}$  for each  $k \in 1, 2$ .

With a sinusoidal thermal forcing,  $g(\gamma, t) := \gamma \cos(t)$ , Python's `fsolve` [2], a modified-Powell root-solver using MINPACK's `hybrd` and `hybrj` algorithms, does not converge to a solution that has the prescribed displacements  $\bar{v}$  for consistent  $\gamma$  across different scales. The following table shows a summary of the experimental results.

Table 3.2: Numerical Bounds on Amplitude, 2 binned systems.

Prescribed displacement	Maximum Amplitude
1	1
$10^{-2}$	$10^{-2}$
$10^{-4}$	$10^{-4}$
$10^{-6}$	$10^{-6}$
$10^{-8}$	$10^{-8}$

Table 3.3: Percent Error  $\bar{v} = 1\text{e-}2$  m

$\gamma$	2x4; $\mathcal{B}^1$	2x4; $\mathcal{B}^2$	10x4; $\mathcal{B}^1$	10x4; $\mathcal{B}^2$	100x10; $\mathcal{B}^1$	100x10; $\mathcal{B}^2$
$10^{-5}$	-0.00%	-0.000%	0.000 %	0.000%	0.001%	0.001%
$10^{-4}$	0.001%	0.001%	-0.008%	-0.008%	-0.010%	-0.008%
$10^{-3}$	0.073%	0.073%	-0.026%	-0.026%	0.041%	0.073%
$10^{-2}$	0.132%	0.134%	0.111%	0.111%	0.064%	0.393%
$10^{-1}$	-1.84%	-1.84%	5.28%	5.25%	27.4924%	18.7670%
10	-15.3%	-15.5%	-39.9%	-39.2%	-147 %	-571 %

Table 3.4: Percent Error of Bin  $\mathcal{B}^i$  for 11 bins of 10.  $\mathcal{O}(\phi) = 10^{-1}$ .

$\bar{v} = 1\text{e-}2\text{m.}$				$\bar{v} = 1\text{e-}4$ m.			
$i$	$\gamma = 10^{-2}$	$\gamma = 10^{-1}$	$\gamma = 1$	$i$	$\gamma = 10^{-5}$	$\gamma = 10^{-4}$	$\gamma = 10^{-3}$
1	-0.456%	1.590%	457.8%	1	0.0579%	0.5436%	-6.4734%
2	-0.393%	-0.056%	-742.7%	2	0.0334%	0.6717%	-4.0408%
3	-0.425%	-0.922%	-324.8%	3	0.0174%	0.5630%	-1.9191%
4	-0.369%	-0.961%	598.6%	4	0.0082%	0.3927%	-1.6866%
5	-0.348%	-1.654%	244.4%	5	0.0011%	0.4406%	-1.6866%
6	-0.448%	-0.394%	-136.6%	6	0.0037%	0.5566%	-1.6866%
7	-0.403%	2.751%	-1397.8%	7	0.0253%	0.4405%	-1.6866%
8	-0.320%	5.024%	-56.11%	8	0.0243%	0.3928%	-1.6224%
9	-0.551%	3.464%	-972.6%	9	-0.0118%	0.5630%	-0.3267%
10	-0.829%	-0.712%	-480.7%	10	-0.0371%	0.6717%	1.8266%
11	-0.956%	-4.188%	-664.3%	11	-0.0378%	0.5436%	4.1905%

**4. Future Work.** Forward directions for this project include an alternative formulation of the problem and further integration of the current scheme to multiple levels, a multiscale approach.

At the base, the problem could be cast as an augmented Lagrangian, and instead of iterating once to obtain the Lagrange multipliers, one could iterate over both the resulting displacements and the extracted Lagrange multipliers until the displacements are as desired. This method would be an alternating directions method; at each step, both the displacements and the Lagrange multipliers would be obtained.

Secondly, the scope of the problem could be generalized to multiple levels as described in Section 1.2. The next-level forcing terms  $C^1$  can be obtained from Equation (1.9); they are the sum of the found Lagrange multipliers acting on a particular bin, and the interatomic

forces at the fine level and the general thermal forcing are incorporated into those values. The proposed method may thus be repeated so that the bins  $\mathcal{B}_0^k$  are aggregated in the same way as the  $u^0$ ; they are the coarse-grained approximation. That is,  $u_k^1 = \mathcal{B}_0^k$  with all the properties associated to it, so there are  $N^1 = K^0$  total “atoms.” in the next level. Thus, this method has strong potential for successive reduction in scale of MD simulations.

**Acknowledgments.** The authors would like to thank Sandia National Laboratories for hosting the internship, the Air Force for funding this research, and the National Science Foundation (NSF) and NSF Graduate Research Fellowship Program for academic and research support under grant DGE-11-44155.

## REFERENCES

- [1] S. DELAHAIES, Numerical Solutions for Partial Differential Equations. MAT 3015, University of Surrey. [Online; accessed 2016-07-28].
- [2] E. JONES, T. OLIPHANT, P. PETERSON, AND ET AL, SciPy: Optimization and Root-finding., 2001–2016. [Online; accessed 2016-07-28].
- [3] J. Li, Basic Molecular Dynamics, in Handbook of Materials Modeling, Springer, 2005, pp. 565–588.
- [4] J. MORÉ, G. B., AND K. HILLSTROM, User Guide for MINPACK., Technical Report 74, Argonne National Laboratory, 1980.
- [5] J. NOCEDAL AND S. J. WRIGHT, Numerical Optimization, Springer, 2 ed., 2006. Columbia Libraries Online. SpringerLink ebooks - Mathematics and Statistics (2006).
- [6] M. L. PARKS, R. B. LEHOUCQ, S. J. PLIMPTON, AND S. A. SILLING, Implementing Peridynamics Within a Molecular Dynamics Code, Computer Physics Communications, 179 (2008), pp. 777–783.
- [7] P. SELESON, M. L. PARKS, M. GUNZBURGER, AND R. B. LEHOUCQ, Peridynamics as an Upscaling of Molecular Dynamics, Multiscale Modeling & Simulation, 8 (2009), pp. 204–227.
- [8] S. A. SILLING, Reformulation of Elasticity Theory for Discontinuities and Long-range Forces, Journal of the Mechanics and Physics of Solids, 48 (2000), pp. 175 – 209.
- [9] ———, Numerical Stability in Velocity-Verlet, 2016. Notes. June 24, 2016.
- [10] ———, Scalable Coarse-Graining Method for Linear Peridynamics, 2016. Notes. May 20, 2016.
- [11] S. A. SILLING AND E. ASKARI, A Meshfree Method Based on the Peridynamic Model of Solid Mechanics, Computers & Structures, 83 (2005), pp. 1526–1535.
- [12] O. TALU AND A. L. MYERS, Reference Potentials for Absorption of Helium, Argon, Methane, and Krypton in High-silica Zeolites, Colloids and Surfaces A: Physicochemical and Engineering Aspects, 187 (2001), pp. 83–93.

## INTERFACE BASED CONDUCTIVITY: AN FEM APPROXIMATION OF ELECTRICAL CONDUCTIVITY IN MULTI-MATERIAL CELLS

LOGAN T. MEREDITH\*, CHRISTOPHER M. SIEFERT†, AND RICHARD M. J. KRAMER‡

**Abstract.** Finite element codes numerically compute solutions to partial differential equations by partitioning the domain into a discrete mesh. The calculation of current density in elements of the mesh via Ohm’s Law requires a reasonably accurate value for the conductivity of the element. However, in elements that contain multiple materials, especially those with wildly varying material conductivities, a naïvely calculated element conductivity can quickly lead to large current densities through insulators. We describe a model for element conductivities in multi-material cells that combines the anisotropy of tensor conductivities with intuition from simple resisting circuits and avoids the problem of spurious current densities. We then illustrate the efficacy of this model by implementing it in the magnetohydrodynamics simulation code ALEGRA with a verification problem.

**1. Introduction.** ALEGRA is a finite element physics simulation code used heavily at Sandia [10]. A significant portion of the code is dedicated to simulating magnetohydrodynamics (MHD), and this is the portion with which we concern ourselves here. Current density is of special importance in some situations, since it is the primary vehicle through which electrical heating occurs. Because elements are the fundamental carriers of material volume in ALEGRA, current density is stored as an elemental variable at the centroid of each element in the mesh.

The ALEGRA code was written to calculate current density  $\mathbf{J}$  from the magnetic field  $\mathbf{H}$  according to Ampère’s Law, essentially given by

$$\mathbf{J} = \nabla \times \mathbf{H}. \quad (1.1)$$

This calculation, however, is problematic. The magnetic flux density  $\mathbf{B}$  is the fundamental variable used by ALEGRA, not the magnetic field. A more direct way to calculate the current density is via Ohm’s Law, given by

$$\mathbf{J} = \sigma \mathbf{E}, \quad (1.2)$$

where  $\sigma$  is the conductivity and  $\mathbf{E}$  is the electric field.

New complications arise with our adoption of Ohm’s Law. Although the electric field is known at element centers, the value of the conductivity is questionable in elements which contain more than one material, since a single value must ideally encapsulate qualities of each material. Although mesh refinement is indeed a solution to this problem, it is not typically feasible when materials are inserted directly into a mesh. Previously, the ALEGRA code computed conductivity of multi-material cells by weighting each material’s conductivity  $\sigma_i$  by the fraction of the volume in the cell that the material occupies  $\phi_i$ , as shown by

$$\sigma_{eq} = \sum_{i=1}^M \phi_i \sigma_i, \quad (1.3)$$

where  $M$  is the number of materials in the element. Multi-material cells are typically rare enough that this is very accurate for sufficiently fine grids; in fact, our revised version of the conductivity calculation can reduce to this volume-averaged model when there is an electric field tangent to the material interfaces, those interfaces are all parallel, and the interface

---

\*University of Rochester, logan.meredith@rochester.edu

†Sandia National Laboratories, csiefer@sandia.gov

‡Sandia National Laboratories, rmkrame@sandia.gov

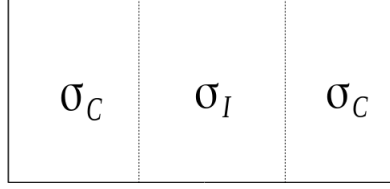


Fig. 1.1: A multi-material cell containing an insulator between two conductors. The volume averaged conductivity would almost ignore the conductivity of the insulator.

normals are eigenvectors of the material conductivities; that is, an electric field tangent to an interface would see the materials as resistors in parallel, since parallel conductivities add arithmetically. Equation (1.3), however, is insufficient in an important and common class of problems.

Suppose there exists a single element with three materials: an insulator between two conductors, as shown in Figure 1.1. For simplicity, we assume that the conductors have the same conductivity. Since the conductivity of the insulator  $\sigma_I$  is small compared to that of the conductors  $\sigma_C$ , its contribution to the conductivity of the element is negligible. The formula approximately reduces to the sum of the conductivities of the conductors scaled by their volume fractions. Now consider an electric field passing normal to the material interfaces. Intuitively, we expect there to be very little current density, since the insulator acts as a sort of barrier. But our previous computation of the element conductivity seems to suggest that the insulator has almost no bearing on the current density in the element.

We present a solution to this problem that takes the orientation of the material interfaces into account. In short, an electric field passing normal to the interface will experience an equivalent conductivity as though the materials were in series, and a tangent field, in parallel. This correctly adjusts the conductivity to give us the result we expect in the above and similar examples. We anticipate the conductivity revision to perform more accurately, and we perform verification analysis to illustrate this.

**2. Further Justification for Ohm's Law.** Before we present our formulation for element conductivity, we should quickly expand upon our reasons for changing the computation of the current density.

**2.1. An *a priori* perspective.** As stated above, the former computation by (1.1) relies on a variable that is derived from a more fundamental variable. This was perhaps an understatement; in fact, the magnetic field is many degrees of separation from the magnetic flux density, and ALEGRA utilized the following chain of calculations to derive the current density, illustrated here:

$$\text{BFLUX} \rightarrow \text{BE} \rightarrow \text{BF} \rightarrow \text{H} \rightarrow \text{HCIRC} \rightarrow \text{JFLUX} \rightarrow \text{JE}$$

Here, BFLUX refers to the magnetic flux through faces, BE represents the magnetic flux density interpolated at element centers, BF is the magnetic flux density tangent to faces on boundaries, H is the magnetic field computed at nodes, HCIRC is the component of the magnetic field parallel to edges, JFLUX is the current flux through faces, and JE represents the current density interpolated at element centers. The arrows correspond to calculations of varying degrees of complexity, and many such calculations incur a significant interpolation error. The current density derivation through Ohm's Law, meanwhile, requires only one calculation. Suppose that the electric field and the magnetic flux density are equally well



defined across the mesh. The current density is likely to be substantially less accurate with Ampère’s Law due to loss of numerical precision alone.

**2.2. Initial qualitative exploration.** We begin with a qualitative comparison of the Ohm’s Law and Ampère’s Law methods of computing current density. Figure 2.1 shows the results of an ALEGRA regression test rendered in ParaView [7] for a problem called “potaslot,” which consists of a coaxial cable whose core and shield are connected at the back.

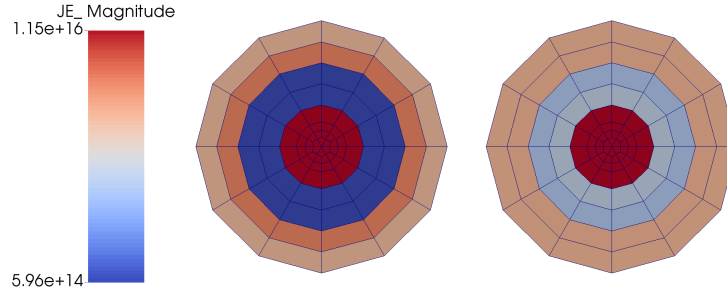


Fig. 2.1: Results from “potaslot” regression test. Left: Ohm’s Law. Right: Ampère’s Law. A brighter red corresponds to high current density, and deeper blue to low.

The orange-red regions in the centers and around the edges in Figure 2.1 correspond to the conductor, and the blue regions between them correspond to the insulator. The two sides are on the same scale to illustrate the differences. Ideally, this insulator carries a few orders of magnitude less current, and so we would expect it to be a very deep blue. However, in the Ampère’s Law simulation, there is a not insignificant current density appearing in the insulator where it should not exist. This effect was commonly encountered in problems with large conductivity drops across boundaries. On the left, however, we can see that Ohm’s Law does not seem to exhibit this phenomenon; the current density in the insulator was calculated around  $8 \times 10^8$  A/m<sup>2</sup> with Ohm’s Law and  $1.2 \times 10^{15}$  A/m<sup>2</sup> with Ampère’s Law. We therefore continue on to a more thorough verification.

**2.3. Verification.** Both versions of the code were run on a suite of verification problems in order to confirm that (1.2) would indeed perform better than (1.1). For brevity, we describe and display the results of only one of these problems, along with a summary of the suite as a whole.

**2.3.1. Problem description and solution.** The problem we will look at is referred to as “transient\_perm\_mag.” The problem domain consists of the 3-cell  $[0, 1]^3$  with a finite electrical conductivity  $\sigma$  upon which we impose the boundary conditions

$$\mathbf{E}(x, y, z, t) = \begin{bmatrix} e^{2t} (e^x + e^{-x}) (e^y - e^{-y}) \\ -e^{2t} (e^x - e^{-x}) (e^y + e^{-y}) \\ 0 \end{bmatrix}$$

on the boundaries along with an initial permanent magnetization across the mesh of

$$\mathbf{M}_0(x, y, z) = \frac{e^x + e^y}{\mu_0} \hat{\mathbf{k}}$$

and an initial magnetic flux density of

$$\mathbf{B}_0(x, y, z) = [e^x + e^y + (e^{-x} + e^x)(e^{-y} + e^y)] \hat{\mathbf{k}}$$

for  $(x, y, z, t) \in [0, 1]^3 \times [0, 10^{-4}]$  and  $\mu_0 = 4\pi \times 10^{-7}$ . In addition, we define the conductivity of the mesh as  $\sigma = \frac{1}{4\pi 10^{-7}}$ . The analytic solution for the resulting current density throughout the block is

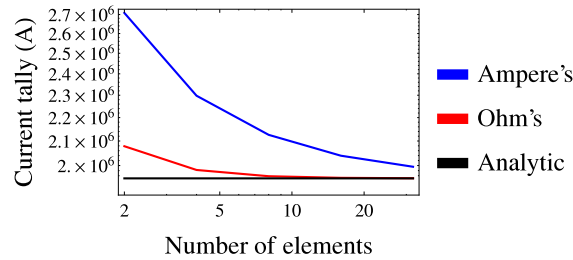
$$\mathbf{J}(x, y, z, t) = \begin{bmatrix} \sigma e^{2t} (e^x + e^{-x})(e^y - e^{-y}) \\ -\sigma e^{2t} (e^x - e^{-x})(e^y + e^{-y}) \\ 0 \end{bmatrix}.$$

The metric that we will use to gauge the improvements afforded by (1.2) is the “current tally.” This is a measure of the total current flowing through a sideset, and is computed internally by ALEGRA. The sideset through which we will tally the current is a plane at  $x = \frac{1}{2}$ . The analytic solution is relatively straightforward to derive, and is given by

$$\int_0^1 \int_0^1 J_x|_{x=\frac{1}{2}} dy dz = 2\sigma(e^2 - 1)e^{2t-1} \cosh \frac{1}{2}.$$

In order to input this problem in ALEGRA, we construct a mesh with a variable number of elements in each direction. Starting with two elements, we refine the mesh by doubling the number of elements with each iteration. This should ideally result in an order 2 convergence rate for the current density onto our analytic solution. We will run this simulation using both (1.2) and (1.1) and compare the computed current tally at time  $t = 10^{-4}$  s to our analytic solution to see which formulation converges more quickly and more accurately.

**2.3.2. Results.** Figure 2.2 shows a log-log plot of the current tally results from ALEGRA against the number of elements in each direction in order to track how each formulation converges toward the analytic solution as the mesh is refined. The black line represents the analytic solution. These results were all computed at  $t = 10^{-4}$  s. We also see a table that displays the relative error of each calculation with its convergence rate in parentheses.



Number of Elements	Ampère	Ohm
2	3.8925e-01 (—)	6.5942e-02 (—)
4	1.7852e-01 (1.12)	1.6817e-02 (1.97)
8	9.0327e-02 (0.98)	4.2764e-03 (1.98)
16	4.6071e-02 (0.97)	1.0784e-03 (1.99)
32	2.3346e-02 (0.98)	2.7061e-04 (1.99)

Fig. 2.2: Results of the ALEGRA current tally in the transient\_perm\_mag problem on a log-log scale and table with relative errors and rates of convergence for both current calculations.

It is clear from Figure 2.2 that Ohm’s Law both converges more quickly and presents a more accurate solution. In fact, Ampère’s Law does not converge at the expected 2<sup>nd</sup> order rate for this problem. Based on the results from this test, we are justified in continuing our implementation of Ohm’s Law to a more comprehensive set of tests, which we describe in the following section.

**2.3.3. Summary.** Here we continue our verification analysis by running simulations with both Ohm’s Law and Ampère’s Law on a small suite of verification tests. Each of these tests is outputting a current tally on its finest mesh in the same fashion as before. In Figure 2.3, we plot the ratio of the Ohm’s Law error to the error in Ampère’s Law on a logarithmic scale for each test. A bar below 1 indicates that Ohm’s Law resulted in a smaller error; above 1, and Ohm’s Law caused a larger error.

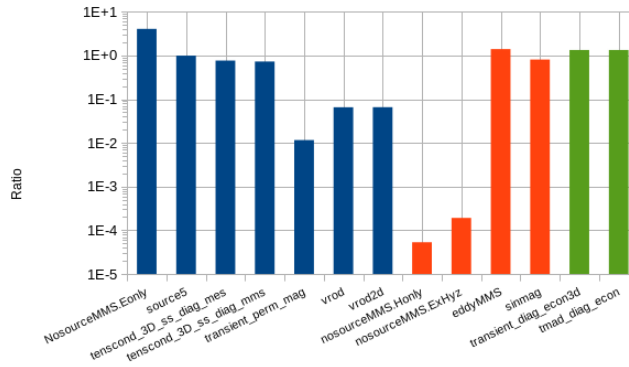


Fig. 2.3: Ohm’s Law error divided by Ampère’s Law error. Values less than one indicate superiority of Ohm’s Law for that problem, and greater than one indicate superiority of Ampère’s Law. Problems driven by electric field boundary conditions are in blue, by magnetic field conditions in green, and by both in red.

We can quickly see that some problems fare better using the older Ampère’s Law formulation. Barring the nosourceMMS problem family, it was observed that these problems tended to be driven by boundary conditions on the magnetic field (in green), whereas most problems in which Ohm’s Law worked better were driven by electric field boundary conditions (blue) or both electric and magnetic fields (red). If the magnetic field is well defined, it is possible to immediately use Ampère’s Law. However, when the electric field is defined, or when electric field boundary conditions are mixed with magnetic field boundaries, Ohm’s Law provides a more direct computation of current density. Indeed, in some cases, Ohm’s Law was several orders of magnitude more accurate relative to Ampère’s Law.

**3. Interface Based Conductivity Formulation.** Consider two materials in an element. Ideally, the component of the electric field tangent to the material interface would see the two materials as though they were two parallel resistors in a circuit, and the normal component would see the conductivity of the two materials as though they were in series. In the example with the insulator between two conductors described in Section 1, this would stunt the current density passing through the insulator, but allow current to flow along the interfaces, which is physically correct. Such anisotropy can be realized with tensor conductivities. In this section, we will provide some necessary background for implementing the algorithm in ALEGRA and a relation describing the model itself.

**3.1. Background.** Upon encountering a multi-material cell, ALEGRA constructs a material priority list, ordering the materials such that the interfaces will be cleanly constructed. It then proceeds to reconstruct the interfaces. Interface reconstruction algorithms are an expansive subject and various examples can be seen in [1, 3, 5, 8]. We will not discuss them here, but it is vital to understand the structure of the completed priority lists.

Consider an element with three materials, as shown in Figure 3.1. An interface is constructed between the material of highest priority and the remainder of the cell. We are most interested in the unit normal of this interface, which is computed at the interface centroid. After this, the fraction of the element containing that material is ignored, and an interface is constructed between the material of second highest priority and the remainder. In the end, if there are  $M$  materials, there are  $M - 1$  interfaces. Note that the interface between the materials of lowest and second lowest priority is the only interface that is constructed between two whole materials.

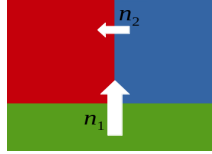


Fig. 3.1: An example of ALEGRA's interface reconstruction scheme. The green material has the highest priority (1), followed by blue (2), then red (3). Note that there is no  $n_3$ ; this is because there are no remaining materials after red in the priority list.

Our work is loosely inspired by the work of White and Solberg in [12], itself inspired by work on uniaxial perfectly matched layers, on which there exists a wealth of literature, as in [2, 4, 9, 11]. Their solution to the mixed element issue was to rotate the interface normals onto the x-axis, then use the same rotation transformation to rotate the conductivity tensors. At that point, it is relatively straightforward to harmonically sum the entries of the first row of the tensors and arithmetically add the remainder. Attention must be paid to avoid destroying tensor symmetry at this point, and this work did not progress to consider nondiagonal tensors. We present a solution that uses projections rather than rotations. It can be shown that, if implemented properly, the rotation method is equivalent to the projection method, which we present in the following section.

**3.2. The model.** For an ideal equivalent tensor conductivity in an element containing two materials that is aware of the material interface and incorporates the changes we have discussed in Sections 1 and 3, we submit the following formula. Let  $n$  be the unit normal to the material interface, let  $\phi_1, \phi_2$  be the volume fractions of the two materials, and let  $\sigma_1, \sigma_2$  be their respective electrical conductivities. Note that our model is valid for individual material conductivities defined both as tensors and as scalars; the result, however, is always a tensor. In addition, we define an arithmetic conductivity sum

$$\sigma_T = \phi_1 \sigma_1 + \phi_2 \sigma_2 \quad (3.1)$$

and a harmonic conductivity sum

$$\sigma_N = (\phi_1 \sigma_1^{-1} + \phi_2 \sigma_2^{-1})^{-1} \quad (3.2)$$

where the two subscripts T and N stand for “tangent” and “normal,” respectively. A first attempt to construct an interface-aware conductivity might look something like

$$\sigma_{eq} \stackrel{?}{=} \sigma_T (I - nn^T) + \sigma_N nn^T. \quad (3.3)$$

However, this does not guarantee that the resulting conductivity tensor will be symmetric. A method for symmetrizing the product of two symmetric tensors  $A, B$  is given by

$$AB \approx \frac{1}{2}(AB + BA). \quad (3.4)$$

With (3.4) in mind, we revise (3.3) and posit that the equivalent element conductivity should be calculated as

$$\sigma_{eq} = \sigma_T + \frac{1}{2} [(\sigma_N - \sigma_T) nn^T + nn^T (\sigma_N - \sigma_T)]. \quad (3.5)$$

In order to compute the conductivity of an element with more than two materials, we will take advantage of the material priority list and the interface reconstruction. Recursing downward from the material of lowest priority to the material of highest priority (since the priority list is stored with the index of the highest priority at 1), we will essentially use (3.5) between each material interface until we have iterated over every material. First, let  $M$  be the number of materials in the cell, let  $n_i$  be the normal to the material interface between materials of priorities  $i$  and  $i + 1$ , let  $\sigma_i$  be the electrical conductivity of the material of priority  $i$ , and let  $\phi_i$  be the volume fraction which the material of priority  $i$  occupies. For compactness, we also define a cumulative volume fraction

$$\kappa_i = \sum_{j=M-i+1}^M \phi_j \quad (3.6)$$

which is constructed such that  $\kappa_M = 1$ . Now we can define the element conductivity  $\sigma_{eq}$  recursively as follows:

$$\begin{aligned} a_1 &= \sigma_M, \\ a_k &= \kappa_k^{-1} (\phi_{M-k+1} \sigma_{M-k+1} + \kappa_{k-1} a_{k-1}) \\ &\quad - \frac{1}{2} \kappa_k^{-1} (\phi_{M-k+1} \sigma_{M-k+1} + \kappa_{k-1} a_{k-1}) n_{M-k+1} n_{M-k+1}^T \\ &\quad + \frac{1}{2} \kappa_k (\phi_{M-k+1} \sigma_{M-k+1}^{-1} + \kappa_{k-1} a_{k-1}^{-1})^{-1} n_{M-k+1} n_{M-k+1}^T \\ &\quad - \frac{1}{2} \kappa_k^{-1} n_{M-k+1} n_{M-k+1}^T (\phi_{M-k+1} \sigma_{M-k+1} + \kappa_{k-1} a_{k-1}) \\ &\quad + \frac{1}{2} \kappa_k n_{M-k+1} n_{M-k+1}^T (\phi_{M-k+1} \sigma_{M-k+1}^{-1} + \kappa_{k-1} a_{k-1}^{-1})^{-1}, \\ \sigma_{eq} &= a_M. \end{aligned} \quad (3.7)$$

Note that for  $M = 2$ , (3.7) reduces to (3.5). Indeed, this formula was derived by repeated application of (3.5). It can be seen that the formula takes the combined conductivity of the last two materials in the priority list, then treats that chunk as a single material and repeats the process with the next material. In the end, we have combined every material in the element into a single chunk.

**3.3. Properties of interface based conductivity.** Here we will outline some properties of the model that are crucial to its usefulness.

PROPERTY 3.3.1 (Symmetry).  $\sigma_M$  is symmetric.

It is easy to see why this would be the case. Our construction of  $\sigma_M$  consists solely of scalar multiplication, element-wise addition, and symmetrized tensor multiplication, all of which are symmetry-preserving operations.

PROPERTY 3.3.2 (Multiple instances of the same material preserve the material's conductivity). Let  $M$  be the number of materials in the element. If

$$\sigma_1 = \sigma_2 = \cdots = \sigma_M,$$

then  $\sigma_{eq} = \sigma_1$ .

An ALEGRA user would expect that an element populated wholly by materials of the same conductivity has the same conductivity as its constituents. Our model would have limited usefulness without this fact, and would in fact break down in the surprisingly common situation of inserting two instances of the same material with different priorities into the same element.

*Proof.* First we must prove that  $a_i = \sigma_1 \forall i \in \{1, 2, \dots, M\}$ . The proof is by induction. Note that  $a_1 = \sigma_1$  by our hypothesis. Now suppose that  $a_{k-1} = \sigma_1$ . Then, using (3.7), we get

$$\begin{aligned} a_k &= \kappa_k^{-1} (\phi_{M-k+1} \sigma_1 + \kappa_{k-1} \sigma_1) \\ &\quad - \frac{1}{2} \kappa_k^{-1} (\phi_{M-k+1} \sigma_1 + \kappa_{k-1} \sigma_1) n_{M-k+1} n_{M-k+1}^T \\ &\quad + \frac{1}{2} \kappa_k (\phi_{M-k+1} \sigma_1^{-1} + \kappa_{k-1} \sigma_1^{-1})^{-1} n_{M-k+1} n_{M-k+1}^T \\ &\quad - \frac{1}{2} \kappa_k^{-1} n_{M-k+1} n_{M-k+1}^T (\phi_{M-k+1} \sigma_1 + \kappa_{k-1} \sigma_1) \\ &\quad + \frac{1}{2} \kappa_k n_{M-k+1} n_{M-k+1}^T (\phi_{M-k+1} \sigma_1^{-1} + \kappa_{k-1} \sigma_1^{-1})^{-1} \\ &= \sigma_1 + \frac{1}{2} (\sigma_1 - \sigma_1) n_{M-k+1} n_{M-k+1}^T + \frac{1}{2} n_{M-k+1} n_{M-k+1}^T (\sigma_1 - \sigma_1) \\ &= \sigma_1 \end{aligned}$$

from which it immediately follows that  $\sigma_{eq} = a_M = \sigma_1$ .  $\square$

In Section 1, we alluded to the fact that the interface-based conductivity model would reduce to the deprecated volume averaged model in the case where the electric field was tangent to the material interfaces, those interfaces were all parallel, and the interface normals were eigenvectors of the conductivity tensors. We present and prove that fact here.

PROPERTY 3.3.3 (Conductivities add arithmetically with parallel interfaces and tangent electric fields). Let  $M$  be the number of materials in the element, and let  $\mathbf{E} \in \mathbb{R}^3$  such that  $\mathbf{E} \cdot \mathbf{n}_1 = 0$ . If  $n_1 = n_2 = \cdots = n_{M-1}$  and  $n_1$  is an eigenvector of  $\sigma_i \forall i \in \{1, 2, \dots, M\}$ , then  $\sigma_{eq} \mathbf{E} = \sum_{j=1}^M \phi_j \sigma_j \mathbf{E}$ .

The hypotheses for this property may seem overly stringent, but they represent a relatively common class of problems. For example, a problem with laminated sheets of materials aligned with a coordinate plane and diagonal conductivity tensors, as in [6, 13], satisfies all but the condition on the electric field.

*Proof.* We must first show that  $n_1$  is an eigenvector of each  $a_i$ . This can be done through induction. Since  $a_1 = \sigma_M$ ,  $n_1$  is an eigenvector of  $a_1$  by hypothesis. Suppose that  $n_1$  is an eigenvector of  $a_{k-1}$ . Let  $a_{k-1} n_1 = \lambda_{k-1} n_1$  and  $\sigma_{M-k+1} n_1 = \gamma_{k-1} n_1$  for some

$\lambda_{k-1}, \gamma_{k-1} \in \mathbb{R}$ . Using (3.7), we get

$$\begin{aligned}
a_k n_1 &= \kappa_k^{-1} (\phi_{M-k+1} \sigma_{M-k+1} + \kappa_{k-1} a_{k-1}) n_1 \\
&\quad - \frac{1}{2} \kappa_k^{-1} (\phi_{M-k+1} \sigma_{M-k+1} + \kappa_{k-1} a_{k-1}) n_1 n_1^T n_1 \\
&\quad + \frac{1}{2} \kappa_k (\phi_{M-k+1} \sigma_{M-k+1}^{-1} + \kappa_{k-1} a_{k-1}^{-1})^{-1} n_1 n_1^T n_1 \\
&\quad - \frac{1}{2} \kappa_k^{-1} n_1 n_1^T (\phi_{M-k+1} \sigma_{M-k+1} + \kappa_{k-1} a_{k-1}) n_1 \\
&\quad + \frac{1}{2} \kappa_k n_1 n_1^T (\phi_{M-k+1} \sigma_{M-k+1}^{-1} + \kappa_{k-1} a_{k-1}^{-1})^{-1} n_1 \\
&= \kappa_k^{-1} (\phi_{M-k+1} \gamma_{k-1} + \kappa_{k-1} \lambda_{k-1}) n_1 \\
&\quad - \frac{1}{2} \kappa_k^{-1} (\phi_{M-k+1} \gamma_{k-1} + \kappa_{k-1} \lambda_{k-1}) n_1 \\
&\quad + \frac{1}{2} \kappa_k (\phi_{M-k+1} \gamma_{k-1}^{-1} + \kappa_{k-1} \lambda_{k-1}^{-1})^{-1} n_1 \\
&\quad - \frac{1}{2} \kappa_k^{-1} (\phi_{M-k+1} \gamma_{k-1} + \kappa_{k-1} \lambda_{k-1}) n_1 \\
&\quad + \frac{1}{2} \kappa_k (\phi_{M-k+1} \gamma_{k-1}^{-1} + \kappa_{k-1} \lambda_{k-1}^{-1})^{-1} n_1 \\
&= \kappa_k (\phi_{M-k+1} \gamma_{k-1}^{-1} + \kappa_{k-1} \lambda_{k-1}^{-1})^{-1} n_1
\end{aligned}$$

which shows that  $n_1$  is an eigenvector. We will now show that

$$\kappa_i a_i \mathbf{E} = \sum_{j=M-i+1}^M \phi_j \sigma_j \mathbf{E} \quad \forall i \in \{1, 2, \dots, M\}.$$

Again, the proof is by induction. Note that  $\kappa_1 a_1 = \phi_M \sigma_M$  by construction, and so  $\kappa_1 a_1 \mathbf{E} = \phi_M \sigma_M \mathbf{E} = \sum_{j=M}^M \phi_j \sigma_j \mathbf{E}$ . Now suppose that

$$\kappa_{k-1} a_{k-1} \mathbf{E} = \sum_{j=M-k+2}^M \phi_j \sigma_j \mathbf{E}.$$

Since  $\mathbf{E} \cdot n_1 = 0$ ,  $n_1 n_1^T \mathbf{E} = \mathbf{0}$ . We will also use the fact that if  $v$  is an eigenvector of a

symmetric tensor  $A$ , then  $Avv^T = vv^T A$ . With (3.7), we have

$$\begin{aligned}
\kappa_k a_k \mathbf{E} &= (\phi_{M-k+1} \sigma_{M-k+1} + \kappa_{k-1} a_{k-1}) \mathbf{E} \\
&\quad - \frac{1}{2} (\phi_{M-k+1} \sigma_{M-k+1} + \kappa_{k-1} a_{k-1}) n_1 n_1^T \mathbf{E} \\
&\quad + \frac{1}{2} \kappa_k^2 (\phi_{M-k+1} \sigma_{M-k+1}^{-1} + \kappa_{k-1} a_{k-1}^{-1})^{-1} n_1 n_1^T \mathbf{E} \\
&\quad - \frac{1}{2} n_1 n_1^T (\phi_{M-k+1} \sigma_{M-k+1} + \kappa_{k-1} a_{k-1}) \mathbf{E} \\
&\quad + \frac{1}{2} \kappa_k^2 n_1 n_1^T (\phi_{M-k+1} \sigma_{M-k+1}^{-1} + \kappa_{k-1} a_{k-1}^{-1})^{-1} \mathbf{E} \\
&= (\phi_{M-k+1} \sigma_{M-k+1} + \kappa_{k-1} a_{k-1}) \mathbf{E} \\
&= \left( \phi_{M-k+1} \sigma_{M-k+1} + \sum_{j=M-k+2}^M \phi_j \sigma_j \right) \mathbf{E} \\
&= \sum_{j=M-k+1}^M \phi_j \sigma_j \mathbf{E}.
\end{aligned}$$

Finally, we notice that  $\sigma_{eq} \mathbf{E} = a_M \mathbf{E} = \kappa_M a_M \mathbf{E} = \sum_{j=1}^M \phi_j \sigma_j \mathbf{E}$ .  $\square$

These two properties exemplify the flexibility of the interface based conductivity model in that it reduces to more familiar models in special circumstances: namely, the circumstances in which the older models excel.

**4. Verification.** A purely mathematical justification does not suffice for full scale implementation. In this section, we incorporate the interface based model in ALEGRA and run a test problem in order to more fully illustrate the superiority of our new model.

**4.1. Problem description.** The problem we will use here is known as “ser.par.” It consists of two elements, one comprised wholly of a conductor and the other of half conducting and half insulating materials, with conductivities  $\sigma_C$  and  $\sigma_I$  respectively, as shown in Figure 4.1.

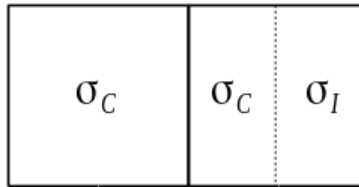


Fig. 4.1: A schematic of the ser.par problem domain. The solid lines denote the element boundaries and the dashed line represents the material interface. Note that  $\sigma_C$  refers to the conductor and  $\sigma_I$  refers to the insulator.

The purpose of the element on the left is to ensure that the interface is reconstructed as we described, since the interface reconstruction algorithm relies on information from surrounding elements to make inferences about the proper location of the boundaries.

In order to drive the simulation, we couple the mesh to an external circuit. In particular, we can choose for the circuit to be configured such that the two materials act either in serial



or in parallel. This is done by coupling a constant potential of  $V_0$  to the left or the bottom of the mesh and a constant potential of 0 to the right or top, respectively.

In the serial case, the current density  $\mathbf{J}$  is expected to be constant throughout the mesh, given by

$$\mathbf{J} = \frac{2V_0}{\frac{3}{\sigma_C} + \frac{1}{\sigma_I}} \hat{\mathbf{i}}. \quad (4.1)$$

In the parallel version, the current should differ between the two elements. The current through the leftmost cell  $\mathbf{J}_1$  is given by

$$\mathbf{J}_1 = V_0 \sigma_C \hat{\mathbf{j}}, \quad (4.2)$$

and through the rightmost cell,  $\mathbf{J}_2$  is given by

$$\mathbf{J}_2 = \frac{V_0}{2} (\sigma_C + \sigma_I) \hat{\mathbf{j}}. \quad (4.3)$$

Inputting the problem into ALEGRA is straightforward. We set  $V_0 = 10^2$  V,  $\sigma_c = (10^2 \text{ S/m}) I$ , and  $\sigma_I = (10^{-4} \text{ S/m}) I$ , where  $I$  is the identity tensor. In addition, we set the simulation termination time to  $t = 1$  s and output the results of the current density at that time. We will run the problem with both the deprecated volume averaged calculation for conductivity and our revised interface based model.

**4.2. Parallel circuit results.** Figure 4.2 shows the output from ALEGRA rendered in ParaView with interface-based conductivity on the left and volume-averaged conductivity on the right. These two pictures are exactly the same. This is because this problem satisfies all of the conditions for Property 3.3.3. More importantly, both solutions are identically close to the analytic solution that we expected to see. This test provides experimental evidence for the correctness of Property 3.3.3. We will move on to the serial circuit, which should exhibit more stark contrasts.

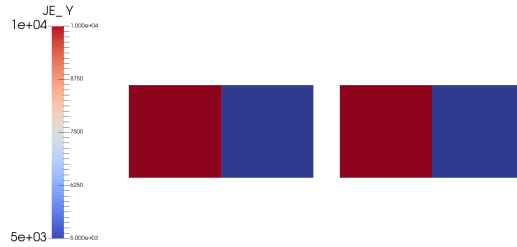
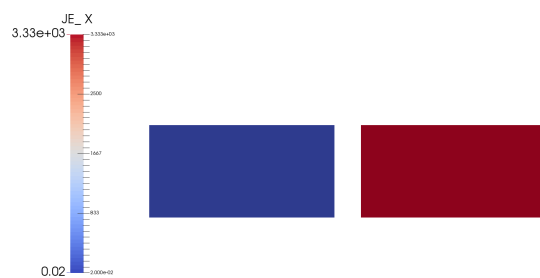


Fig. 4.2: ALEGRA output for the parallel version of the ser\_par test. Left: interface based conductivity. Right: volume averaged conductivity. The analytic solution is  $J_{1,y} = 10^4 \text{ A/m}^2$ ,  $J_{2,y} \approx 5 \times 10^3 \text{ A/m}^2$ .

**4.3. Serial circuit results.** Figure 4.3 shows the output from ALEGRA rendered in ParaView with interface-based conductivity on the left and volume-averaged conductivity on the right. The current density as calculated by the volume averaged model is about five orders of magnitude greater than that of the interface based model. In particular, the volume averaged model is about five orders of magnitude greater than the analytic solution.



**5. Conclusions.** Sandia’s ALEGRA finite element magnetohydrodynamics simulation code is applied often to calculate current densities in complicated problems. Since element-centered electric field values are typically better known than magnetic field values, we have shown that it is prudent to take advantage of the electric field when calculating current density in the form of Ohm’s Law. In addition, we addressed the issue of conductivity in multi-material cells, which is of particular importance in the use of Ohm’s Law, by introducing a new, interface based conductivity model. This formulation reconstructs and considers material interfaces when calculating element conductivity. We then described some properties of the model and demonstrated its flexibility on a contrived but commonly-encountered verification problem.

## REFERENCES

- [1] K. S. BONNELL, M. A. DUCHAINEAU, D. R. SCHIKORE, B. HAMANN, AND K. I. JOY, Material Interface Reconstruction, *IEEE Trans. on Vis. and Comp. Graphics*, 9 (2003), pp. 500–511.
- [2] W. C. CHEW AND W. H. WEEDON, 3D Perfectly Matched Medium from Modified Maxwells Equations With Stretched Coordinates, *Microwave and Optical Tech. Lett.*, 7 (1994), pp. 599–604.
- [3] V. DYADECHKO AND M. SHASHKOV, Reconstruction of Multi-Material Interfaces from Moment Data, *Journal of Computational Physics*, 227 (2008), pp. 5361–5384.
- [4] S. D. GEDNEY, An Anisotropic Perfectly Matched Layer Absorbing Media for the Truncation of FDTD Lattices, *IEEE Trans. Antennas and Propagation*, 44 (1996), pp. 1630–1639.
- [5] F. GIBOU AND C. MIN, A Second Order Accurate Level Set Method on Non-Graded Adaptive Cartesian Grids, *Journal of Computational Physics*, 225 (2007), pp. 300–321.
- [6] X. JIANG AND W. ZHENG, An Efficient Eddy Current Model for Nonlinear Maxwell Equations with Laminated Conductors, *SIAM J. Appl. Math.*, 72 (2012), pp. 1021–1040.
- [7] PARAVIEW DEVELOPMENT TEAM, ParaView, 2016. Version 5.1.2.
- [8] E. G. PUCKETT, A Volume-of-Fluid Interface Reconstruction Algorithm that Is Second-Order Accurate in the Max Norm, *Comm. App. Math. and Comp. Sci.*, 5 (2010), pp. 199–220.
- [9] C. M. RAPPAPORT, Perfectly Matched Absorbing Boundary Conditions Based on Anisotropic Lossy Mapping of Space, *IEEE Microwave and Guided Wave Lett.*, 5 (1995), pp. 90–92.
- [10] A. C. ROBINSON, W. J. RIDER, ET AL., ALEGRA: An Arbitrary Lagrangian-Eulerian Multimaterial, Multiphysics Code, in *Proceedings of the 46th AIAA Aerospace Sciences Meeting*, Reno, NV, Jan. 2008. AIAA-2008-1235.
- [11] Z. S. SACKS, D. M. KINGSLAND, R. LEE, AND J.-F. LEE, A Perfectly Matched Anisotropic Absorber for Use as an Absorbing Boundary Condition, *IEEE Trans. Antennas and Propagation*, 43 (1995), pp. 1460–1463.
- [12] D. WHITE AND J. SOLBERG, Anisotropic Conductivity for Mixed Elements. Unpublished slides from LLNL presentation, Aug. 2015.
- [13] W. ZHENG AND Z. CHENG, An Inner-Constrained Separation Technique for 3-D Finite-Element Modeling of Grain-Oriented Silicon Steel Laminations, *IEEE Trans. Magn.*, 48 (2012), pp. 2277–2283.

## SEMI-LAGRANGIAN RKPM AND ITS IMPLEMENTATION IN THE SIERRA/SOLIDMECHANICS ANALYSIS CODE

MARCO PASETTO\* AND DAVID J. LITTLEWOOD†

**Abstract.** Numerical simulations of problems involving large deformations, material damage, fracture, and material fragmentation are challenging for computational mechanics. Some of the issues related to the use of the Finite Element (FE) method for this class of problems, such as mesh entanglement and mesh distortion, can be avoided by using meshfree approaches for discretization of the problem domain and the construction of the approximation space. The reproducing kernel particle method (RKPM) is a meshfree approach for computational solid mechanics that can obtain an arbitrary order of completeness and smoothness, offering a valid alternative to FE for the aforementioned problems. While a Lagrangian RKPM approach can handle very large deformations, its formulation breaks down when the mapping between the undeformed and the deformed configuration is no longer one-to-one. This is the case for extremely large deformation problems, and for problems when material fragmentation and formation of new surfaces occur. For the solution of these types of problems, the semi-Lagrangian RKPM approach was implemented in *Sierra/SolidMechanics*. An overview of the RKPM semi-Lagrangian method is first presented together with its implementation in the *Sierra/SolidMechanics* analysis code, with main focus on its differences with respect to the Lagrangian RKPM approach and its implementation. The performance and potential of semi-Lagrangian RKPM is then shown through a set of initial benchmark problems. Semi-Lagrangian RKPM is deemed to significantly increase the class of problems solvable with RKPM in *Sierra/SolidMechanics*.

**1. Introduction.** Large deformations, material damage, fracture and material fragmentation are challenging computational mechanics problems. Standard Finite Element (FE) approaches often perform poorly for the aforementioned classes of problems due to mesh distortion and element inversion. Even though the performance of FE can be improved through the use of techniques such as adaptive refinement or, in the presence of material separation, through the use of the extended finite element method (XFEM), this usually results in a significant computation expense [2]. For this reason, Lagrangian RKPM was implemented in the *Sierra/SolidMechanics* analysis code at Sandia National Laboratories for explicit transient dynamic problems [2] [16].

In the RKPM Lagrangian meshfree approach, the variational equations are formulated in the reference undeformed configuration. Because of the meshfree nature of the method, materials undergoing very large deformations can be modeled without encountering mesh distortion or entanglement issues and with less loss of accuracy [7]: the Lagrangian RKPM formulation has therefore been successfully used to model problems involving plasticity [7], hyperelasticity [6], and structural dynamics [18]. However, Lagrangian RKPM breaks down when the mapping between the deformed and the undeformed configuration is no longer one-to-one and the deformation gradient loses positive definiteness. This is the case when modeling extremely large deformation problems, such as high velocity impact and penetration processes, during which new surface formation and material fragmentation occur. To overcome this limitation, the semi-Lagrangian RKPM formulation [15] was developed, where the approximation functions are constructed directly in the current deformed configuration, meaning that the mapping between the undeformed and the deformed configuration is no longer required. Semi-Lagrangian RKPM has been used for a large variety of problems, such as impact problems involving fragmentation [11] [1], slope stability analyses [14] and earth moving simulations [15]. In order to allow the solution of such problems with *Sierra/SolidMechanics*, the implementation of semi-Lagrangian RKPM in its framework was performed.

An overview of the RKPM method, with particular attention to the peculiarities of the

---

\*University of California, San Diego, mpasetto@ucsd.edu

†Sandia National Laboratories, djlittl@sandia.gov

semi-Lagrangian approach and its implementation in *Sierra/SolidMechanics* is presented in Section 2. A concrete perforation problem and simulation of a wave propagating along the length of a bar are shown in Section 3 to present the capabilities of semi-Lagrangian RKPM and to provide an initial verification of its coding implementation.

**2. Methodology.** The semi-Lagrangian RKPM formulation differs from the Lagrangian one mainly in that the equilibrium equation is formulated in the current and in the reference configuration, respectively. This leads to several theoretical and implementation differences; notable ones include the construction of the approximating functions and gradients being performed at different times and the computation of the deformation gradient. Especially the former makes it necessary to severely restructure the computation steps previously used in the *Sierra/SolidMechanics* for the Lagrangian approach [2].

**2.1. Solution Approximation.** Differently from the Finite Element approach, where the shape functions and the related approximation space are strictly linked to element connectivity, in the RKPM approach the approximate solutions are constructed over a meshfree discretization of the considered domain. Let  $\bar{\Omega} \subset \mathbb{R}^3$  be the considered closed domain and let it be discretized by a set of NP nodes  $\{\mathbf{x}_I | \mathbf{x}_I \in \bar{\Omega}\}_{I=1}^{NP}$ . The reproducing kernel (RK) approximation  $u^h(\mathbf{x})$  of a generic function  $u(\mathbf{x})$  in  $\bar{\Omega}$  is defined as:

$$u^h(\mathbf{x}) = \sum_{I=1}^{NP} \Psi_I(\mathbf{x}) u_I, \quad (2.1)$$

where  $\{\Psi_I(\mathbf{x})\}_{I=1}^{NP}$  is the set of RK shape functions and  $\{u_I\}_{I=1}^{NP}$  is the set of nodal coefficients of the approximation [17][7].

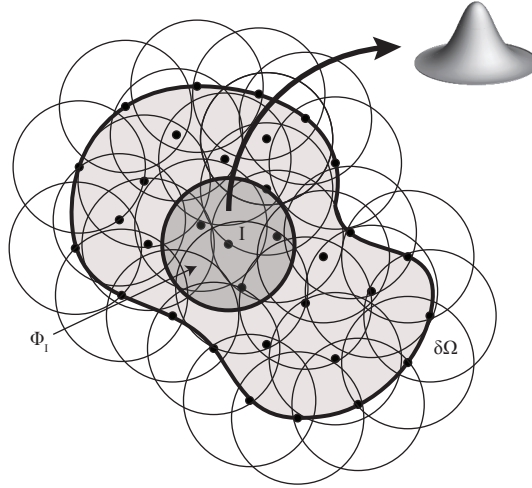


Fig. 2.1: RKPM domain discretization with RK approximation functions

RK shape functions are defined as the product of a weight kernel function  $\Phi_a(\mathbf{x} - \mathbf{x}_I)$  with compact support measure  $a$ , such that  $\Phi_I = \{\mathbf{x} | \Phi_a(\mathbf{x} - \mathbf{x}_I) \neq 0\}$  (Figure 2.1), and of a correction function  $C(\mathbf{x}; \mathbf{x} - \mathbf{x}_I)$ :

$$\Psi_I(\mathbf{x}) = \Phi_a(\mathbf{x} - \mathbf{x}_I) C(\mathbf{x}; \mathbf{x} - \mathbf{x}_I). \quad (2.2)$$

The correction function  $C(\mathbf{x}; \mathbf{x} - \mathbf{x}_I)$  is defined as a linear combination of monomial basis functions:

$$C(\mathbf{x}; \mathbf{x} - \mathbf{x}_I) = \sum_{i+j+k=0}^n (x_1 - x_{I1})^i (x_2 - x_{I2})^j (x_3 - x_{I3})^k b_{ijk} \equiv \mathbf{H}^T(\mathbf{x} - \mathbf{x}_I) \mathbf{b}(\mathbf{x}), \quad (2.3)$$

where  $\mathbf{b}(\mathbf{x})$  is the column vector of the monomial coefficients  $b_{ijk}$  and  $\mathbf{H}^T(\mathbf{x} - \mathbf{x}_I)$  is the row vector of the monomial bases:

$$\mathbf{H}^T(\mathbf{x} - \mathbf{x}_I) = [\{(\mathbf{x} - \mathbf{x}_I)^\alpha\}_{|\alpha| \leq n}], \quad (2.4)$$

where  $\alpha = (\alpha_1, \alpha_2, \alpha_3)$ ,  $|\alpha| = \sum_{i=1}^3 \alpha_i$ ,  $\mathbf{x}^\alpha = x_1^{\alpha_1} x_2^{\alpha_2} x_3^{\alpha_3}$  and  $\mathbf{x}_I^\alpha = x_{I1}^{\alpha_1} x_{I2}^{\alpha_2} x_{I3}^{\alpha_3}$ . The basis order  $n$  defines the order of the shape functions and the reproducing completeness. Smoothness of the approximation functions is determined through the choice of the kernel function  $\Phi_a(\mathbf{x} - \mathbf{x}_I)$ : a cubic B-spline function such as the one shown in (2.5), for example, gives  $C^2$  continuity.

$$\Phi_a\left(\frac{d}{a}\right) = \begin{cases} \frac{2}{3} - 4\left(\frac{d}{a}\right)^2 + 4\left(\frac{d}{a}\right)^3 & \text{for } 0 \leq \frac{d}{a} \leq \frac{1}{2} \\ \frac{4}{3} - 4\left(\frac{d}{a}\right) + 4\left(\frac{d}{a}\right)^2 - \frac{4}{3}\left(\frac{d}{a}\right)^3 & \text{for } \frac{1}{2} \leq \frac{d}{a} \leq 1 \\ 0 & \text{otherwise} \end{cases} \quad (2.5)$$

where

$$d = \|\mathbf{x} - \mathbf{x}_I\| \quad (2.6)$$

The coefficients  $\mathbf{b}(\mathbf{x})$  are determined by substituting (2.2) in the following polynomial reproducing conditions:

$$\sum_{I=1}^{NP} \Psi_I(\mathbf{x}) x_{I1}^i x_{I2}^j x_{I3}^k = x_1^i x_2^j x_3^k, \quad 0 \leq i + j + k \leq n. \quad (2.7)$$

Once  $\mathbf{b}(\mathbf{x})$  is obtained from (2.7), the RK shape functions are constructed as

$$\Psi_I(\mathbf{x}) = \mathbf{H}^T(\mathbf{0}) \mathbf{M}^{-1}(\mathbf{x}) \mathbf{H}^T(\mathbf{x} - \mathbf{x}_I) \Phi_a(\mathbf{x} - \mathbf{x}_I), \quad (2.8)$$

where  $\mathbf{M}(\mathbf{x})$  is called the moment matrix and is defined as follows:

$$\mathbf{M}(\mathbf{x}) = \sum_{I=1}^{NP} \mathbf{H}(\mathbf{x} - \mathbf{x}_I) \mathbf{H}^T(\mathbf{x} - \mathbf{x}_I) \Phi_a(\mathbf{x} - \mathbf{x}_I). \quad (2.9)$$

In order for the reproducing conditions in (2.7) to hold, the moment matrix defined in (2.9) must be invertible. This means that the reproducing equations are linearly independent. This is possible only if there are a sufficient number of kernels of non-coplanar points covering the evaluation point  $\mathbf{x}$  [7]. For this reason, every evaluation point will be covered by multiple kernel supports  $\Phi_I(\mathbf{x})$  and consequently multiple shape functions  $\Psi_I(\mathbf{x})$ . This leads to the RK nodal coefficients (usually called generalized displacements) not being equivalent to the physical displacements at the nodes ( $u_I \neq u^h(\mathbf{x}_I)$ ), i.e the RK shape functions, differently from the Finite Element ones, lack the Kronecker delta property. Because of this, exact imposition of essential boundary conditions in a RKPM framework is not trivial. In *Sierra/SolidMechanics* boundary singular kernel functions are employed [4] [2]; these RK

shape functions are modified to recover the Kronecker delta property and to therefore allow direct imposition of the boundary conditions.

Lastly, via direct differentiation of (2.7), the following property can be derived:

$$\sum_{I=1}^{NP} \nabla \Psi_I(\mathbf{x}) x_{I1}^i x_{I2}^j x_{I3}^k = \nabla (x_1^i x_2^j x_3^k), \quad 0 \leq i + j + k \leq n. \quad (2.10)$$

**2.2. Lagrangian and semi-Lagrangian RKPM formulations.** As discussed in Sections 1 and 2.1, in a RKPM meshfree-type discretization, approximation is achieved through construction of the shape functions in the physical domain and the interaction of neighboring nodes. When modeling deformation problems, a Lagrangian [7] [8] and a semi-Lagrangian [15] [1] [5] formulation have been proposed. As previously stated, in the RKPM Lagrangian meshfree approach, the variational equations are formulated in the reference undeformed configuration and RK functions and their derivatives are thus evaluated only once during the preprocessing stage of the analysis. However, since Lagrangian RKPM breaks down when the mapping between the deformed and the undeformed configuration is no longer one-to-one and the deformation gradient loses positive definiteness, the semi-Lagrangian RKPM formulation was developed. Here, the approximation functions are constructed directly in the current deformed configuration meaning that the mapping between the undeformed and the deformed configuration is no longer required. This conceptual difference between the Lagrangian and the semi-Lagrangian RKPM formulations is illustrated in Figure 2.2. Figure 2.2A shows spherical kernel supports in the undeformed configuration. In Lagrangian RKPM, they deform together with the material body, and every evaluation point  $\mathbf{x}$  is always covered by the same nodal supports (Figure 2.2B), while in the semi-Lagrangian formulation the RK shape functions and the kernels are rebuilt every time step (Figure 2.2C). Therefore, even though the description of motion is Lagrangian in both approaches, the semi-Lagrangian kernels assume an Eulerian flavor and a convective term arises in the formulation. In *Sierra/SolidMechanics*, in order to rebuild kernels, shape functions, and shape function gradients at every time integration step, the nodal neighbors are searched using an artificially increased spherical domain and then the derived neighbor list is reduced through the use of the kernel support size,  $a$ . The idea behind this approach is to create for each node an associated enlarged neighbor list containing all the nodes with which the considered node is expected to interact over a time-window comprised of multiple time-steps. At every time step, the actual neighbor list for the considered node is then built by comparing its kernel support size  $a$  with its distance from all the other nodes in its enlarged neighbor list. This way the comparison is performed over a set of nodes that is smaller with respect to the one containing all the nodes in the domain, resulting in computational saving. The more computationally expensive neighbor search over all the nodes in the domain, in fact, has to be performed only when necessary, that is at the beginning of each considered multi-step time window.

Consider now a body initially occupying a domain  $\Omega_X$  with undeformed boundary  $\Gamma_X$  (Figure 2.3). Let us denote the initial density, body force and the surface traction in the undeformed configuration by  $\rho^0$ ,  $\mathbf{b}^0$  and  $\mathbf{h}^0$  respectively. The material displacement is  $\mathbf{u} = \mathbf{x} - \mathbf{X}$  where  $\mathbf{X}$  are the material reference coordinates and  $\mathbf{x}$  are the current spatial coordinates. The relationship between  $\mathbf{X}$  and  $\mathbf{x}$  is the mapping function defined  $\mathbf{x} = \varphi(\mathbf{X}, t)$ . After deformation occurs, the body domain is  $\Omega_x$  with deformed boundary  $\Gamma_x$ ;  $\rho$ ,  $\mathbf{b}$  and  $\mathbf{h}$  represent the density, body force and the surface traction in this deformed configuration.

The equation of motion in the Lagrangian formulation is:

$$\int_{\Omega_X} \delta u_i \rho^0 \ddot{u}_i \, d\Omega + \int_{\Omega_X} \delta F_{ij} P_{ij} \, d\Omega = \int_{\Omega_X} \delta u_i b_i^0 \, d\Omega + \int_{\Gamma_X} \delta u_i h_i^0 \, d\Gamma, \quad (2.11)$$

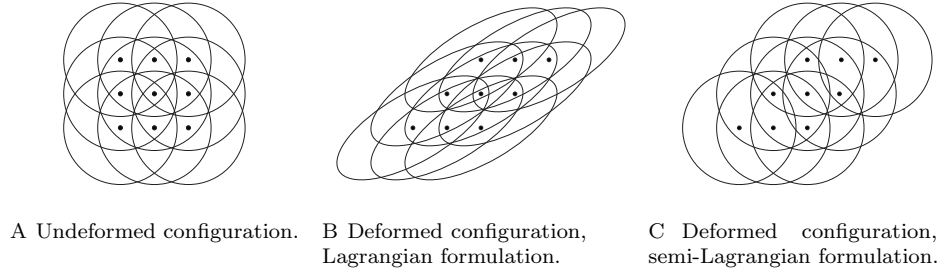


Fig. 2.2: Lagrangian and semi-Lagrangian discretizations.

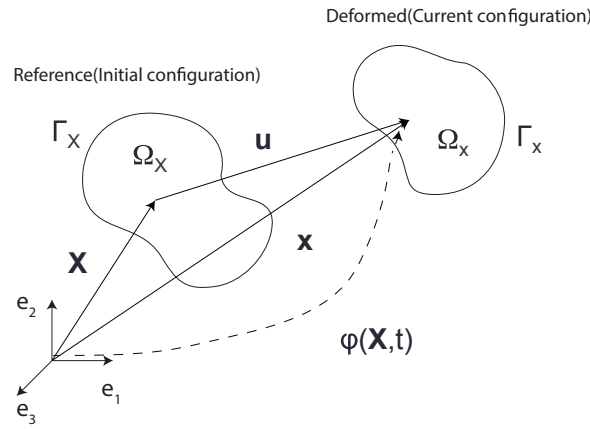


Fig. 2.3: Deformation of a material body

where  $\mathbf{F}$  is the deformation gradient defined as  $F_{ij} = \partial x_i / \partial X_j$ , and  $\mathbf{P}$  is the first Piola-Kirchhoff stress tensor. The Cauchy stress is usually employed as the stress measure for path-dependent material models, which means that it becomes necessary to evaluate the spatial derivatives of the displacement. In the RKPM Lagrangian formulation this is done via the following chain rule:

$$\frac{\partial \Psi_I^L(\mathbf{X})}{\partial x_i} = \frac{\partial \Psi_I^L(\mathbf{X})}{\partial X_j} \frac{\partial X_j}{\partial x_i} = \frac{\partial \Psi_I^L(\mathbf{X})}{\partial X_j} F_{ij}^{-1}, \quad (2.12)$$

where  $\Psi_I^L(\mathbf{X}) = \mathbf{H}^T(\mathbf{0})\mathbf{M}^{-1}(\mathbf{X})\mathbf{H}^T(\mathbf{X} - \mathbf{X}_I)\Phi_a(\mathbf{X} - \mathbf{X}_I)$  is the Lagrangian shape function centered at node  $I$  and  $\mathbf{F}^{-1}$  is the inverse of the deformation gradient  $F$ . As previously mentioned, this approach breaks down when the mapping function  $\mathbf{x} = \varphi(\mathbf{X}, t)$  is no longer one-to-one and thus no longer invertible. The semi-Lagrangian formulation solves this problem by expressing the equation of motion as follows:

$$\int_{\Omega_x} \delta u_i \rho \ddot{u}_i \, d\Omega + \int_{\Omega_x} \delta u_{(i,j)} \sigma_{ij} \, d\Omega = \int_{\Omega_x} \delta u_i b_i \, d\Omega + \int_{\Gamma_x} \delta u_i h_i \, d\Gamma, \quad (2.13)$$

where  $u_{(i,j)} = (\partial u_j / \partial x_i + \partial u_i / \partial x_j) / 2$  and  $\sigma$  is the Cauchy stress. The semi-Lagrangian

shape functions as previously stated are evaluated in the current configuration as  $\Psi_I^{SL}(\mathbf{x}) = \mathbf{H}^T(\mathbf{0})\mathbf{M}^{-1}(\mathbf{x})\mathbf{H}^T(\mathbf{x} - \mathbf{x}_I)\Phi_a(\mathbf{x} - \mathbf{x}_I)$ .

**2.3. Stress update and calculation of the internal force.** As with the Lagrangian RKPM approach described in [2], the semi-Lagrangian approximation of the displacement field described in (2.1) enables the calculation of kinematic quantities at every time step in the simulation. These kinematic quantities are then passed to the constitutive model to compute stresses, which are then in turn used to compute nodal forces and related accelerations. At this point, an explicit time integrator advances the simulation to the next time step.

The *Sierra/SolidMechanics* implementation of both Lagrangian [2] and semi-Lagrangian RKPM interfaces directly with the *Library of Advanced Materials for Engineering*, where a large number of constitutive models is available [19]. The main kinematic quantities to be passed to the material constitutive models are the symmetric part of the deformation tensor  $\bar{\mathbf{D}}$  and the deformation gradient  $\mathbf{F}$ .  $\bar{\mathbf{D}}$  can be computed directly in the current configuration in semi-Lagrangian RKPM as:

$$\bar{D}_{ij} = \frac{1}{2\Delta t}(G_{ij} + G_{ji}), \quad (2.14)$$

where  $\mathbf{G}$  is computed incrementally by using the gradient of the increment of displacement  $\Delta \mathbf{u} = \mathbf{u}^{n+1} - \mathbf{u}^n$  with respect to the half time step configuration  $\mathbf{x}^{n+1/2} = (1/2)(\mathbf{x}^n + \mathbf{x}^{n+1})$  [2]:

$$G_{ij} = \frac{\partial \Delta u_i}{\partial x_j^{n+1/2}}, \quad (2.15)$$

which leads to [2]:

$$\mathbf{G} = \mathbf{D}_F(\mathbf{I} + \frac{1}{2}\mathbf{D}_F)^{-1}, \quad (2.16)$$

where  $\mathbf{I}$  is the identity tensor and where the  $ij$ -th component of the incremental deformation gradient  $\mathbf{D}_F$  is defined as:

$$(\mathbf{D}_F)_{ij} = \sum_{J=1}^{NP} \Psi_{Jj}^{\nabla, SL}(\mathbf{x}) \Delta u_{Ji}, \quad (2.17)$$

where  $\Delta \mathbf{u}_J$  is the incremental generalized displacement at node  $J$  and  $\Psi_{Jj}^{\nabla, SL}(\mathbf{x})$  is the smoothed spatial gradient of the semi-Lagrangian shape function, thus computed in the current deformed configuration. The deformation gradient  $\mathbf{F}$  is not computed directly. To obtain its value when necessary, its inverse  $\mathbf{F}^{-1}$  is computed first as:

$$F_{ij}^{-1} = \delta_{ij} - \sum_{J=1}^{NP} \Psi_{Jj}^{\nabla, SL}(\mathbf{x}) u_{Ji}, \quad (2.18)$$

and  $\mathbf{F}$  is then computed as  $\mathbf{F} = (\mathbf{F}^{-1})^{-1}$ .

Differently from standard Finite Element schemes, and as in the Lagrangian RKPM approach implemented in *Sierra/SolidMechanics* [2], the above quantities are computed at the nodes rather than Gauss points due to the use of a nodal integration scheme. For use with Lagrangian RKPM, two nodal spatial integration schemes had previously been implemented in *Sierra/SolidMechanics* [2]: SCNI (Stabilized Conforming Nodal Integration) [9] and SNNI (Stabilized Nonconforming Nodal Integration) [1].



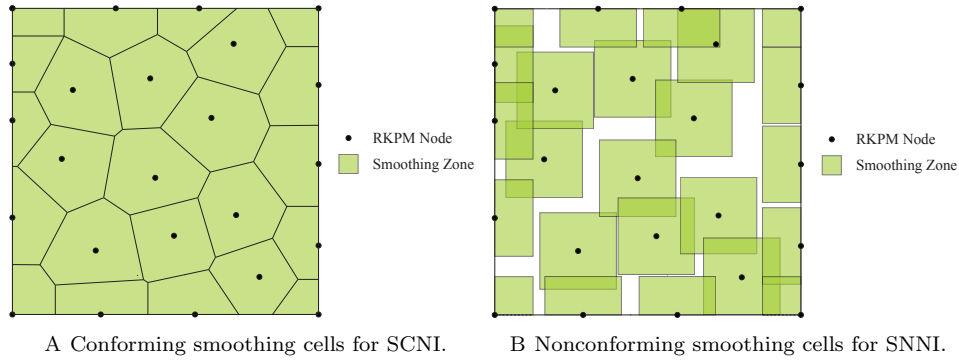


Fig. 2.4: RKPM smoothing cells for SCNI and SNNI.

SCNI was developed to remedy rank instability present in the direct nodal integration scheme and to obtain optimal convergence for linearly complete functions. In order to do so, the domain is partitioned in conforming cells (Figure 2.4A) each associated with a RKPM node; to avoid the rank instability arising from the evaluation of the shape functions gradients at the nodes, gradients are smoothed over the conforming nodal domains. Because of the conforming nature of the nodal cells and of the way the smoothing is performed, the first order variational consistency for Galerkin linear exactness is satisfied, leading to optimal convergence. Unfortunately, SCNI is not a feasible integration scheme for semi-Lagrangian RKPM. For SCNI to be used in the semi-Lagrangian formulation, the conforming nodal domains would have to be reconstructed at every integration time step, making this approach too computationally expensive for practical purposes. For this reason SNNI is used as an alternative to SCNI. In SNNI the smoothing of the shape function gradients is performed over nonconforming cells, represented by box domains constructed around each RKPM node (Figure 2.4B); these smoothing domains maintain their shape for the whole duration of the simulation, making them very suitable for semi-Lagrangian RKPM. The relaxation of the conforming requirement however leads to convergence rates being no longer optimal. To improve the performance of the SNNI scheme an assumed strain method was developed to achieve satisfaction of the first order variational consistent (VC) condition [10]. Together with the semi-Lagrangian formulation, an additional integration scheme has been implemented in *Sierra/SolidMechanics*: NSNI (Naturally Stabilized Nodal Integration). NSNI adds stabilization to the DNI (Direct Nodal Integration) scheme by performing a Taylor-like expansion of the approximate displacement field in each nodal integration cell about the related RKPM node; this leads to the elimination of the zero energy modes associated with DNI through addition of coercivity arising naturally from the formulation and the RKPM domain discretization. The interested reader is referred to [13].

After  $\bar{\mathbf{D}}$  and  $\mathbf{F}$  have been computed according to eqs. (2.14)-(2.18), the values of the Cauchy stress  $\sigma^{n+1}$  are evaluated through a call to the material constitutive model. Lastly the internal force  $\mathbf{f}_{int}^{n+1}$  is determined as:

$$\mathbf{f}_{int}^{n+1} = \sum_{J=1}^{NP} \mathbf{B}^T(\mathbf{x}_J) \Theta^{n+1}(\mathbf{x}_J) V_J, \quad (2.19)$$

where  $\mathbf{B}^T(\mathbf{x}_J)$  and  $\Theta^{n+1}(\mathbf{x}_J)$  are the Voigt notation vector representation of respectively the smoothed spatial gradients and the Cauchy stress, while  $V_J$  is the current nodal volume.

Once the internal forces are available, nodal accelerations are computed and the time integration scheme advances to the next time step. In *Sierra/SolidMechanics* the procedure to estimate the time step to be used in semi-Lagrangian RKPM is currently the same as the one used for Lagrangian RKPM, described in [2].

**3. Verification & Validation.** The semi-Lagrangian RKPM framework in *Sierra/SolidMechanics* is currently undergoing verification and validation and being corrected and refined accordingly to the results obtained from the V&V process. For this reason, only an initial test case of a wave propagation in a bar solved with *Sierra/SolidMechanics* will be presented here. In order to further demonstrate the capabilities of the semi-Lagrangian formulation, a concrete perforation problem solved with NMAP (Nonlinear Meshfree Analysis Program) [12] is presented first. Lastly, the ongoing work being performed to further verify and validate the semi-Lagrangian implementation and performance is described.

**3.1. Impact-Perforation Simulation.** Simulation of the penetration of a spherical steel projectile impacting a CorTuf [3] ultra high-strength concrete panel with a zero degree angle is presented in Figure 3.1. This example problem is taken from the experiments presented in [11], and the corresponding experimental results are used as a reference for comparison with the the numerical results from the RKPM simulation. Specifically, impact of a spherical projectile with an initial velocity of 1114.65 m/s on a 2.54 cm thick CorTuf concrete panel is considered. In the NMAP simulation, the projectile was modeled as a rigid body with a discretization of 2,273 nodes while the concrete panel was discretized using 224,422 nodes and modeled using the MIDM (microcrack informed damage model)-enhanced AFC model; the AFC material model parameters can be found in [11]. Contact between the panel and the projectile was modeled using the smooth penalty contact algorithm [12] with a penalty factor of 0.2e8.

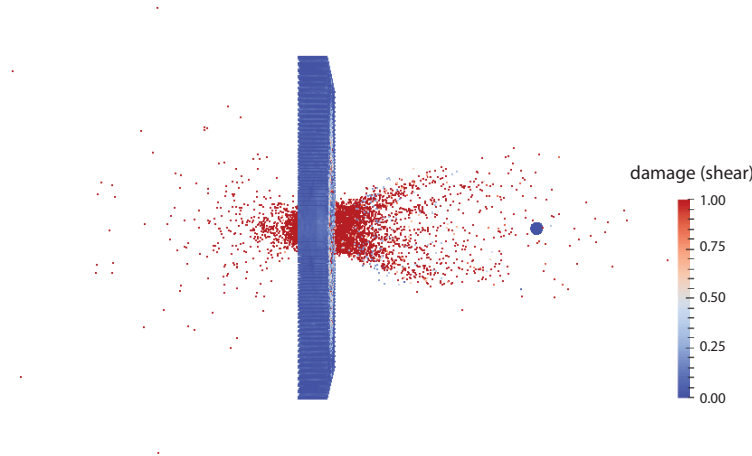


Fig. 3.1: Debris cloud due to concrete perforation

Figure 3.1 shows the debris cloud generated by the fragmentation of the concrete due to impact, while in Figures 3.2A and 3.2B the panel shear and tension damage patterns can be observed.

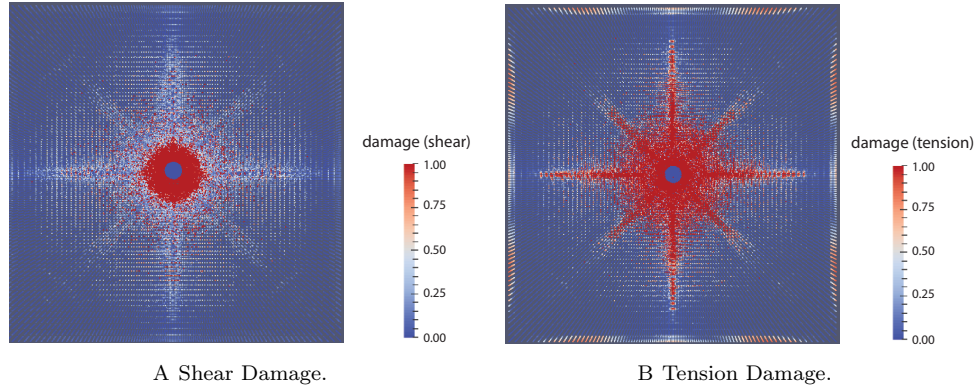


Fig. 3.2: Shear and tension damage.

The numerical result for the projectile velocity over time is compared to the experimental final velocity, represented as a constant line, in Figure 3.3. The semi-Lagrangian simulation projectile final velocity is 578.28 m/s, against an observed experimental one of 544.68 m/s. The numerically predicted reduction in velocity of the projectile is therefore 52% while the experimental velocity reduction is 49%.

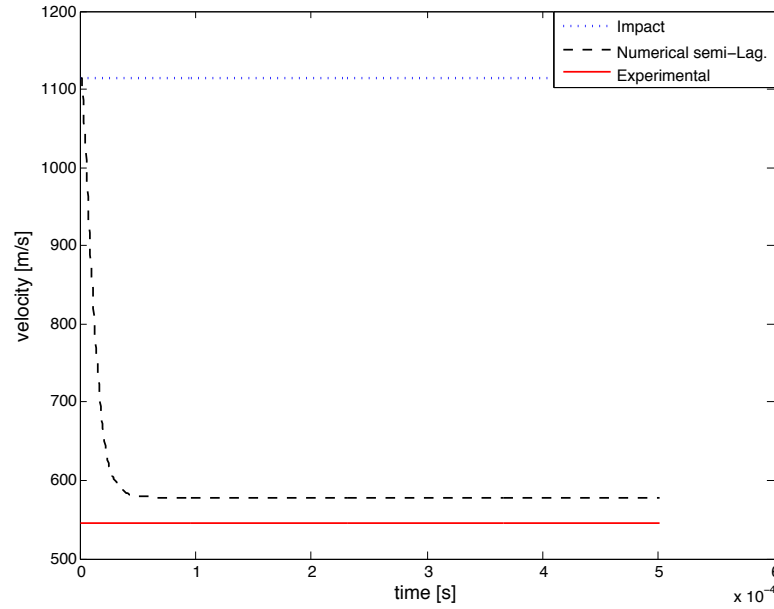


Fig. 3.3: Projectile exit velocity

**3.2. Wave Propagation in a Bar.** In order to verify the implementation of semi-Lagrangian RKPM in *Sierra/SolidMechanics*, propagation of a wave along a long, thin bar is used as a test. A model of a bar with dimensions of 2 cm by 0.02 cm by 0.02 cm was built. The bar was modeled as elastic with a density of 7.8 g/cm<sup>3</sup> and a Young's modulus of 3.0e5 MPa. A Poisson's ratio of zero was used to simulate a one-dimensional solution. For the RKPM formulation the bar was discretized with 404 nodes and a first order basis, a quartic B-spline kernel, SNNI domain integration and a normalized support size of 2.1 were used. The bar was given an initial velocity of 100 cm/s and a fixed boundary displacement condition was applied to the leading edge of the bar. The same type of test was performed with SCNI domain integration to verify the Lagrangian RKPM implementation in *Sierra/SolidMechanics* in [2]. The following analytical expression was used as a reference solution to verify the wave propagation results:

$$u(x, t) = \sum_{n=1}^{\infty} (A_n \sin(\omega_n t) \sin(\frac{(2n-1)\pi}{2L} x)), \quad (3.1)$$

where

$$A_n = \frac{8v_0 L}{(2n-1)^2 \pi^2} \sqrt{\frac{\rho}{E}}, \quad (3.2)$$

and

$$\omega_n = \frac{(2n-1)\pi}{2L} \sqrt{\frac{E}{\rho}}, \quad (3.3)$$

In the above expressions,  $u(x, t)$  represents the displacement at time  $t$  at position  $x$ ,  $L$  and  $v_0$  are respectively the initial length and initial velocity of the bar,  $\rho$  is the bar density and  $E$  its Young's modulus. Figures 3.4 and 3.5 show comparison between the computational and the analytical solution for the considered problem. Further verification and validation is however necessary and is the subject of ongoing work described in the next subsection.

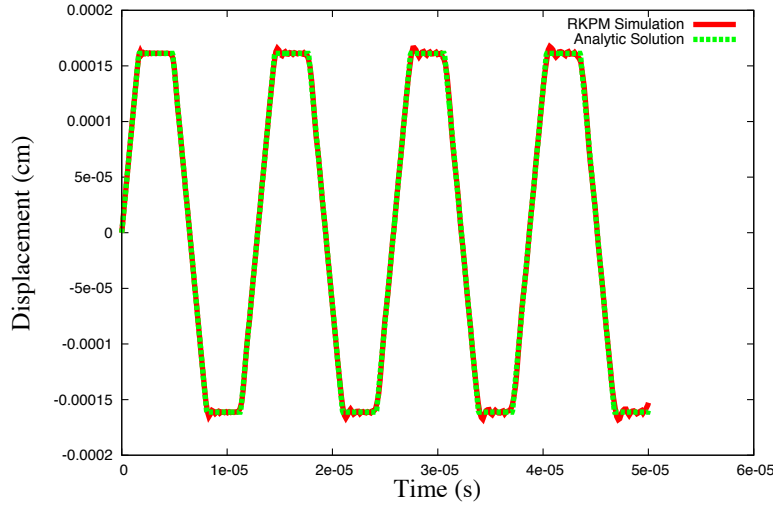


Fig. 3.4: Bar midpoint displacement over time.

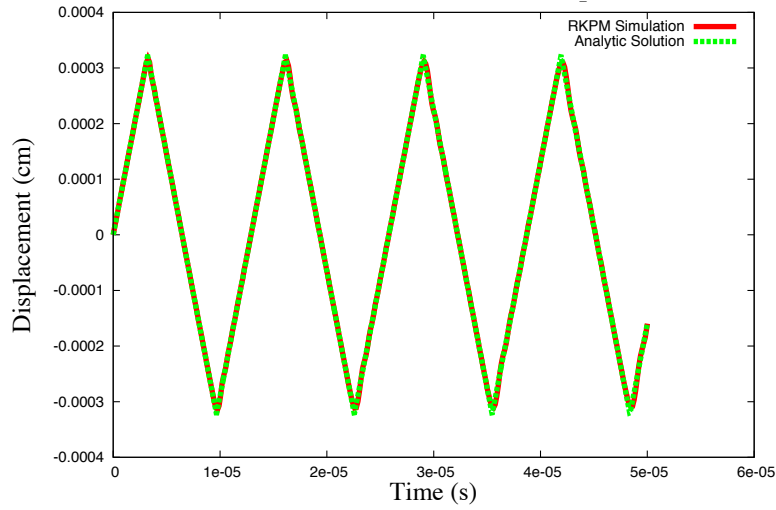


Fig. 3.5: Bar endpoint displacement over time.

**3.3. Current and ongoing work.** As mentioned, the implementation of semi-Lagrangian RKPM in *Sierra/SolidMechanics* is currently undergoing verification and validation and, when necessary, being accordingly corrected. The following points describe the current and ongoing work being performed in order to ensure a correct and effective implementation, and to showcase the capabilities of the semi-Lagrangian RKPM approach:

- Simple test cases with known expected solutions are being solved as a tool for debugging. To this end, the problem of a freely rotating cube with different given initial velocities is being considered. Solutions obtained with the NMAP code are also being used as a sanity check.

- The code implementation of semi-Lagrangian RKPM is being systematically checked through individual inspection of all the related subroutines and their interaction with the *Sierra/SolidMechanics* framework.
- The capabilities of semi-Lagrangian RKPM have started being assessed in order to further define its application space and its performance for significant problems encountered in practice. In this regard, simulations of a cylindrical steel rod impacting a steel plate are currently being performed, providing a starting point for further ongoing effort in this direction.

**4. Conclusions.** The semi-Lagrangian RKPM formulation is being implemented in *Sierra/SolidMechanics* with the goal of providing an effective simulation capability for problems involving extremely large deformations, material fragmentation, and formation of new surfaces. The implementation is currently restricted to explicit transient dynamics and is at the time of writing undergoing improvement and verification and validation. The semi-Lagrangian formulation together with the solution of some initial benchmark problems were presented to show some of the method capabilities and the status of semi-Lagrangian RKPM in *Sierra/SolidMechanics*.

#### REFERENCES

- [1] C. GUAN, S.W. CHI, J.S. CHEN ET ALII, Semi-Lagrangian Reproducing Kernel Particle Method for Fragment-Impact Problems, *International Journal of Impact Engineering*, 38 (2011), pp. 1033–1047.
- [2] D.J. LITTLEWOOD, M. HILLMAN ET ALII, Implementation and Verification of RKPM in the Sierra/-SolidMechanics Analysis Code, *Proceedings of ASME 2015 International Mechanical Engineering Congress and Exposition*, (2015).
- [3] E.M. WILLIAMS, S.S. GRAHAM, P.A. REED AND T.S. RUSHING, Laboratory Characterization of Cor-Tuf Concrete With and Without Steel Fibers, Tech. rep ERDC/GSL, TR-09-22 (2009).
- [4] J.S. CHEN AND H.-P. WANG, New Boundary Condition Treatment in Meshfree Computation of Contact Problems, *Computer methods in Applied Mechanics and Engineering*, 187 (2000), pp. 441–468.
- [5] J.S. CHEN AND Y. WU, Stability in Lagrangian and Semi-Lagrangian Reproducing Kernel Discretizations Using Nodal Integration in Nonlinear Solid Mechanics, *Advances in Meshfree Techniques*, 5 (2007), pp. 55–76.
- [6] J.S. CHEN, C. PAN AND C.-T. WU, Large Deformation Analysis of Rubber Based on a Reproducing Kernel Particle Method, *Computational Mechanics*, 19 (1997), pp. 211–227.
- [7] J.S. CHEN, C. PAN, C.-T. WU AND W.K. LIU, Reproducing Kernel Particle Methods for Large Deformation Analysis of Non-Linear Structures, *Computer methods in Applied Mechanics and Engineering*, 139 (1996), pp. 195–227.
- [8] J.S. CHEN, C. PAN, C.M.O.L. ROQUE AND H.P. WANG, A Lagrangian Reproducing Kernel Particle Method for Metal Forming Analysis, *Computational Mechanics*, 22 (1998), pp. 289–307.
- [9] J.S. CHEN, C.-T. WU, S. YOON AND Y. YOU, A Stabilized Conformal Nodal Integration for Galerkin Mesh-free Methods, *International Journal for Numerical Methods in Engineering*, 50 (2001), pp. 435–466.
- [10] J.S. CHEN, M. HILLMAN AND M. RTER, An Arbitrary Order Variationally Consistent Integration for Galerkin Meshfree Methods, *International Journal for Numerical Methods in Engineering*, 95 (2013), pp. 387–418.
- [11] J.S. CHEN, S. CHI, C. LEE ET ALII, A Multi Scale Meshfree Approach for Modeling Fragment Penetration into Ultra-Strength Concrete, Tech. rep ERDC/GSL, TR-11-35 (2011).
- [12] J.S. CHEN, S.W. CHI, H. LEE ET ALII, User's Manual for Nonlinear Meshfree Analysis Program (NMAP), tech. rep., ERDC/GSL, Geotechnical and Structural Laboratory, US Army Corps of Engineers, 2012.
- [13] M. HILLMAN AND J.S. CHEN, An Accelerated, Convergent and Stable Nodal Integration in Galerkin Meshfree Methods for Linear and Nonlinear Mechanics, *International Journal for Numerical Methods in Engineering*, 107 (2015), pp. 603–630.
- [14] O.-L. KWOK, P.-C. GUAN, W.-P. CHENG AND C.-T. SUN, Semi-Lagrangian Reproducing Kernel Particle Method for Slope Stability Analysis and Post-Failure Simulation, *KSCE Journal of Civil Engineering*, 19 (2015), pp. 107–115.

- [15] P. GUAN, J. CHEN ET ALII, Semi-Lagrangian Reproducing Kernel Particle Formulation and Application to Modeling Earth Moving Operations, *Mechanics of Materials*, 41 (2009), pp. 670–683.
- [16] SIERRA SOLID MECHANICS TEAM, Sierra/SolidMechanics 4.36 User's Guide, tech. rep., SAND Report, 2015-2199, Sandia National Laboratories, 2015.
- [17] W.K. LIU, S. JUN AND Y.F. ZHANG, Reproducing Kernel Particle Methods, *International Journal for Numerical Methods in Fluids*, 20 (1995), pp. 1081–1106.
- [18] W.K. LIU, S. JUN, S. LI ET ALII, Reproducing Kernel Particle Methods for Structural Dynamics, *International Journal for Numerical Methods in Engineering*, 38 (1995), pp. 1655–1679.
- [19] W.M. SCHERZINGER AND D.C. HAMMERAND, Constitutive Models in LAME, tech. rep., SAND Report, 2007-5873, Sandia National Laboratories, 2007.

## EXPLORING THE USE OF THE FOVIO EYETRACKER TO MEASURE COGNITIVE LOAD

GRACE A. THOMPSON\*, LAURA E. MATZEN†, AND MIKA L. ARMENTA‡

**Abstract.** Pupillometry metrics collected by the FOVIO eyetracker have recently been used in conjunction with the Index of Cognitive Activity (ICA) to measure cognitive activity in humans. This measure could be indicative of the cognitive load a person is experiencing. If accurate, the ICA used in conjunction with FOVIO technology could be used in a wide range of field studies. We seek to confirm whether these reports are accurate and consistent across individuals and types of cognitive load. We do so by developing a number of tasks that test the many variables that could affect the eyetracker’s ability to measure cognitive load. Electroencephalography (EEG) is already an established measure of cognitive load, so to determine the accuracy of the eyetracker’s findings, we also collected EEG data during the tasks and will compare results across measurement systems.

**1. Introduction.** In order to evaluate the abilities and efficiency of specialists in national security work domains, an accurate and non-invasive technology is needed to collect data in ecologically realistic situations. The FOVIO eyetracker presents a possible candidate for this task [4]. The FOVIO is an eyetracker that uses an infrared illumination and camera system to capture the eyes’ motion, gaze point, and pupil dilation. When used in conjunction with the Index of Cognitive Activity (ICA) the FOVIO has the potential to unobtrusively measure cognitive load in naturalistic work environments [23]. In testing this function, there are three potential problems that we must consider. First of all, the term “cognitive load” is ambiguous. It is interpreted in many different ways by different scientists and the specific mechanics that cause cognitive load are still largely unknown. Secondly, an established way to measure cognitive load is by using electroencephalography while an individual performs the N-Back task, a working memory task [3]. The main task used to induce cognitive load when recording with the FOVIO has been the Follow-Solve-Listen task which loads multiple cognitive processing functions, including auditory, visual, and executive, in addition to working memory [24]. Finally, there are many potential variables that could affect the FOVIO’s ability to accurately measure pupil diameter, which is a key parameter influencing ICA measurement. Although initial studies using the ICA have been promising, they have usually used small sample sizes with complex analysis of the data. This study aims to gather a broader set of data and gain some additional evidence either for or against the use of the FOVIO to measure cognitive ability in the field.

**1.1. Cognitive Load.** The term cognitive load is fairly ambiguous and its definition is not widely agreed upon. The idea of cognitive load was first introduced by Dr. John Sweller in his 1988 paper *Cognitive Load During Problem Solving: Effects on Learning* [29]. Sweller defined cognitive load as mental effort. Performing harder tasks or doing multiple tasks at once requires more mental effort and increases cognitive load while performing easier tasks requires less mental effort and decreases cognitive load. Many studies have been conducted on how different difficulty levels of tasks affect cognitive load and how the level of cognitive load affects levels of awareness, attention, and multitasking [18][19][31].

Cognitive load has no universal metric assigned to it so researchers have developed a variety of ways to characterize it. The first way is through feedback from participants. Self-reporting is helpful in understanding overall load, but is severely lacking when more minuscule changes need to be observed [6]. The second way is by changing the difficulty

---

\*University of Texas at Austin, grace.thompson@utexas.edu

†Sandia National Laboratories, lematze@sandia.gov

‡Sandia National Laboratories, mlarmen@sandia.gov



in a certain task and collecting data on response times, accuracy, and error rates. Instead of changing the difficulty of one task, researchers can also add secondary tasks to increase cognitive load. Finally, researchers can also collect physiological data to assess cognitive load.

There are three main factors that affect cognitive load: perceptual load, working memory, and cognitive fatigue. Perceptual load increases or decreases depending on the visual complexity of the stimulus. Higher levels of perceptual load correlate with lower levels of awareness of distractors [18]. During tasks with more visual complexity, participants were less likely to notice non-task-related distractors. Working memory is the ability to hold information in the mind for short periods of time. Many studies on working memory require participants to remember a “memory set”, typically consisting of images, information, or words, while doing different tasks. Unlike perceptual load, higher levels of induced working memory loads lead to more awareness of distractors [19] [31]. Finally, cognitive load is affected by cognitive fatigue. Cognitive fatigue refers to the exhausted feeling one gets after doing a long and/or difficult task. Cognitive fatigue is less objectively measurable and it is not entirely clear what causes it. Different combinations of these three main variables create different levels in cognitive load. These changes can be measured through a variety of different technologies.

**1.2. EEG.** The main physiological measure used to quantify cognitive load is electroencephalography (EEG). Other measures like Functional Magnetic Resonance Imaging (fMRI), Positron Emission Tomography (PET) scans, eye metrics, and hormone monitoring are inconvenient for a variety of reasons, such as being too costly, invasive, unreliable, and/or slow [6]. EEG, however, is non-invasive, relatively inexpensive, and has high temporal resolution. There are also several studies showing its effectiveness in accurately measuring cognitive load [8].

EEG is able to pick up four different brain rhythms: delta, theta, alpha, and beta waves, each with a different frequency. Both theta and alpha waves can be affected by changes in cognitive load [14][15][17]. One study showed that as tasks become more difficult, theta activity increases while alpha activity decreases [13]. Individual Alpha Frequency (IAF) was also shown to drop with performance during memory tasks [16]. When measuring cognitive load with EEG, it is better to measure the changes in the EEG signal rather than the absolute value of a frequency band because of the variations in brain waves from one person to another [15].

Despite being one of the best tools to measure cognitive load, the EEG has some shortcomings. One such shortcoming is that almost all EEG studies take place in highly controlled environments. Also, while the EEG possesses great temporal resolution, it is severely lacking in spatial resolution. Another problem is that when measuring cognitive load, EEG is almost used exclusively with the N-Back task. Because of this, cognitive load is measured mostly when working memory is used. A technique is needed that can measure all variables of cognitive load. Finally, the EEG is easily affected by muscle movement or other sources of electricity (e.g. lights, computers, phones), which can mask the actual brain waves [8]. The EEG is not very effective because of these shortcomings and does not suit our needs outside of controlled environments. However, because it is so effective in measuring cognitive load, we can use it to test other technologies that could potentially be used outside such a controlled experiment.

**1.3. Pupillometry.** Pupillometry is the measurement of pupil diameter. The FOVIO eyetracker, when used in conjunction with the Index of Cognitive Activity (ICA), has been reported to measure cognitive load using pupillometry metrics. The FOVIO can measure pupil size and track eye gaze by using an infrared illumination and a camera system directed

at the eyes. Pupil dilation is mostly affected by changes in light. These changes are usually very large and can be seen by the naked eye. However, when all other factors are held constant, minor pupil dilation caused by mental activity can be observed [7]. The ICA takes the data recorded by the FOVIO and measures the frequency of these small pupil dilations [21]. There have been multiple studies using eyetrackers to measure cognitive load while testing, performing surgery, and making complex decisions [5][28][22].

Because the changes in pupil size due to cognitive load are so small, they can be easily masked by a variety of other factors that affect pupil size. For example, changes in luminosity of a screen or brightness of the environment can cause large changes in pupil size. Also, the position of the eye can change the eyetracker's perceived size of the pupil [9][12]. Also, the techniques that ICA uses for measuring the frequency of these dilations are not as widely studied as other pupillometry metrics. In fact, before the ICA, most eye-based measures of cognitive load measured the change in pupil size rather than the frequency of those changes. If the FOVIO eyetracker and ICA can successfully perceive and analyze the small changes in pupil size due to cognitive load without being affected by all the other larger changes in pupil diameter, then it has the potential for being a very powerful tool.

## 2. Materials and Methods.

**2.1. Participants.** All participants were volunteers over the age of 18 recruited from Sandia National Laboratories. All participants had normal or corrected-to-normal vision. Volunteers were excluded if they were not right-handed, had a history of neurological disease (seizures, stroke, tumor etc.), or had a history of skull fractures, brain injury, or brain surgery. All the participants signed a consent form and filled out a demographic form upon arriving at their appointment.

**2.2. EEG Setup.** EEG caps in different sizes from the company ANT Neuro were used. The electrode layout on the caps was the equidistant hexagonal layout [1] with 128 electrodes. However, a montage was created so that only 32 of those electrodes were being used. The decision to only use 32 electrodes was made because this study does not require spatial accuracy and using fewer electrodes saves time during the preparation of the participant.

**2.3. Eyetracking Setup.** The eyetracker used was the FOVIO distributed by Eyetracking, Inc. It was positioned 22.0 cm in front of the base of the monitor, with a vertical offset of 2.5 cm. The eyetracker maintained an angle of about 28°, while the monitor maintained an angle of about 5°. Both angles were remeasured daily and fluctuated  $\pm 0.2^\circ$ . The eyetracker was calibrated before each task.

**2.4. Stimuli.** Five different tasks were created to test different variables that could affect the eyetracker's measure of cognitive load. All tasks were either made on or converted to E-Prime, an application suite created by Psychology Software Tools, Inc. designed to make computer tasks [30]. The five tasks were counterbalanced so equal numbers of participants went through the tasks in different orders. The participants were seated approximately 90 cm away from the 24 inch monitor inside an electrically shielded booth. The door was left open, however as it got very hot inside the booth and perspiration interrupts the EEG recordings. Timing triggers that recorded the beginning and end of each block of every task were inserted so the EEG data could be accurately correlated with the eyetracking data. Fixation points (5000 ms duration) with colors corresponding to each task were also added before each block to verify the calibration of the eyetracker during the task and to serve as an additional landmark for the beginning of the block in the EEG data. A complete calibration of the eyetracker was also conducted before each task.

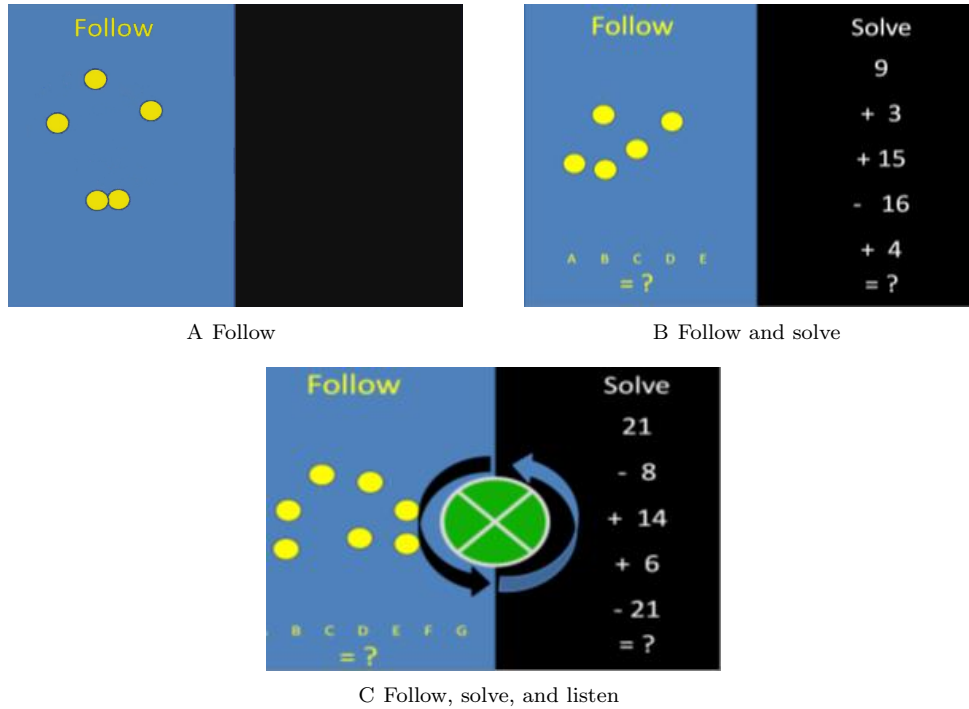


Fig. 2.1: The three difficulty levels of the Follow, Solve, Listen task.

**2.4.1. Task 1 – Follow, Solve, Listen (FSL).** The first task was provided by EyeTracking, Inc and is the task they used to verify their reports that the ICA is indicative of cognitive load. The task was originally created by EyeTracking, Inc on EyeWorks, an eyetracking software suite [2]. The task is divided into three parts increasing in difficulty level. The first part requires participants to follow a yellow dot among other yellow dots on a blue background on the left half of the screen (Figure 2.1A). The second block repeats the same task but also asks the participants to do a progressive math problem on the right half of the screen (Figure 2.1B). Finally, in addition to the first two tasks, the last block has the participant watch a spinning green circle and say what direction it is spinning when they hear a beep (Figure 2.1C). Each block takes approximately 45 seconds.

**2.4.2. Task 2 – FSL No-task.** The FSLT NoTask mimics a gaze trajectory performed during the Follow-Solve-Listen Test [25] without the task demands. In other words, participants are presented with visual stimuli comparable to the FSLT and for the same time amount of time (roughly 45s per block), however they are not asked to track the moving target among distractors, compute the arithmetic problem, or report the direction of the rotating wheel when the tone sounds. Instead, participants follow a white dot that moves along the same trajectory of a real participant performing the FSLT. Stimuli in the FSLT NoTask were matched for color, luminosity and size to their comparable task-on FSLT stimuli. For example, the backgrounds of both tasks are black, and the green disk in the third block of both tasks is centered (see Figure 2.2).

This task was created using one participant's gaze coordinate data from the FSLT. Three different individuals were recorded doing the FSL Task at sampling rate of 60Hz by

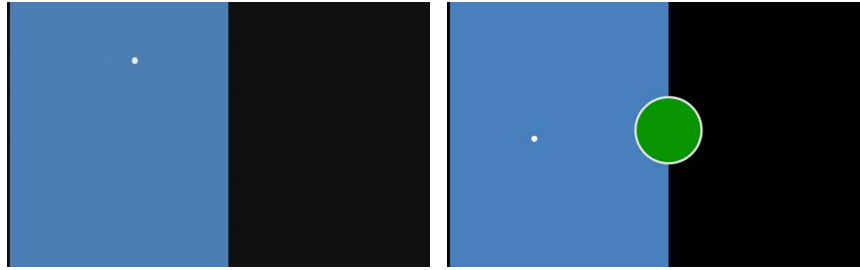


Fig. 2.2: Follow-Solve-Listen NoTask stimuli.

a FOVIO eye-tracker. The data was then run through Marshall's Index of Cognitive Load (ICA) algorithm in EyeWorks Analyze. The individual's gaze data set that was chosen showed an increase in ICA so the task could mimic the eye movements that correlated with an increase in cognitive load. Missing data points were linearly interpolated. Following this, point-to-point distances (p-to-p) between consecutive gaze points were used to distinguish significant saccades (quantified as a distance greater-than-or-equal-to 100 pixels) from minute microsaccades, jitter and other noise. Groups were created based on these significant saccades. All samples up to a p-to-p distance of 100 pixels or greater constituted a group. The coordinates associated with a distance of 100 pixels or greater began a subsequent group, which concluded with the coordinates associated with a distance just prior to a p-to-p greater-than-or-equal-to 100 pixels. To make the FSLT No-Task comfortable for participants to view, further smoothing was necessary and we applied a moving window average (8 samples per window) within each of the groups. All analyses were performed using MATLAB R2015A [26].

**2.4.3. Task 3 – N-back Combined Task.** The N-back task is a common EEG task used to create different levels of cognitive load. The N-back task requires participants to utilize their working memory by remembering a letter or orientation “n” number of steps back in the sequence. For our task, participants were required to determine if the stimulus shown was the same as the stimulus shown 3(n=3) slides back.

To create different levels of cognitive load, we created three different versions of the task. The first two were meant to induce lower levels of cognitive load while the last one was meant to induce a higher level of cognitive load. The first block required the participant to remember the position of a white box. The second block required the participant to remember the letter inside the white box. The last block required the participant to remember both the position and the letter inside of the white box. To familiarize them with the task, we had participants do a short practice round of each block before beginning the task.

All of the stimuli were presented on a black background in white. Before each stimulus, a 0.5 second fixation cross at the center of the screen surrounded by eight empty boxes was presented. The stimulus containing the eight boxes, one completely white with a black letter in its center, was then presented for 0.5 seconds (see Figure 2.3). Finally, the eight empty boxes were shown again for 1.5 seconds. Participants were told to press the “1” key if the current slide matched the slide three trials back and the “2” key if it did not.

**2.4.4. Task 4 – Hebrew Letter Task.** This task was originally described by Sophie Forster and Nilli Lavie in their 2009 paper [11]. They modeled it after a similar Perceptual

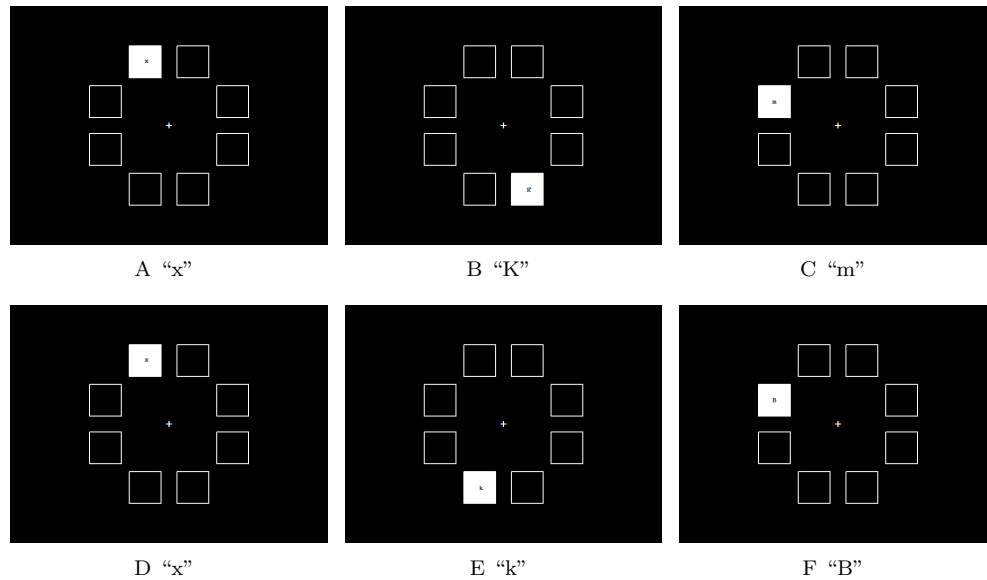


Fig. 2.3: An example sequence of stimuli from the N-Back Task. **A**, **B**, and **C** are all incorrect for all three versions of the task; **D** is correct for all three versions (verbal, spatial, and combined) of the task; **E** is correct for the verbal version of the task; **F** is correct for the spatial version of the task

Load task that they created in the same experiment that displayed circles of letters and dots in quick succession. However, instead of using the English alphabet, this task uses Hebrew letters. The participants were asked to look for the two Hebrew Letters Gimel (Figure 2.4A) and Final Pe (Figure 2.4B) and to press the “0” and “2” keys respectively if they saw them. We used this task to induce perceptual load through visual search. The participants were asked before this task if they knew how to read Hebrew or if they knew any Hebrew letters. The purpose of this was to make sure no verbal/lingual input affected how the participants went through the task.

This task had both a low load and high load version. For both versions, the stimuli were preceded by a 500 ms blank interval and 500 ms fixation. The stimulus was then presented for 100 ms. In the low load version, each stimulus consisted of five dots and one target letter arranged randomly in a circle (with the radius subtending 2° degrees of visual angle) (Figure 2.5A). In the high load version, each stimulus replaced three of the dots with three different non-target letters (Figure 2.5B). All stimuli, fixations, and instructions were presented on a black background. The stimuli and the 500 ms fixation points were gray text while the instructions were presented in white text.

**2.4.5. Task 5 – Raven’s Task.** Our fifth task was a variation of the Raven’s Progressive Matrices, which is a widely used task to measure analytical intelligence [10]. It was originally developed as a booklet of tests called *Standard Progressive Matrices* in 1938 by John Carlyle Raven [20]. The task we used was developed by Laura E. Matzen et al. in 2010 [27]. The task was divided into two blocks. The first block contained only the matrix problems while the second block had an additional auditory task for the participants to complete at the same time as the matrix problems.

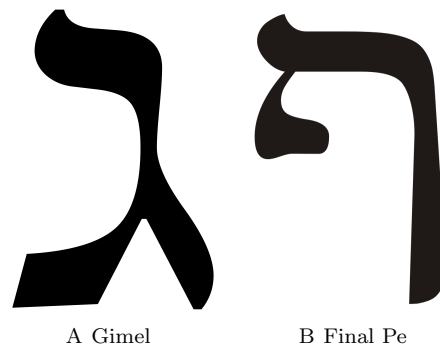


Fig. 2.4: The two target letters for the Perceptual Load task.

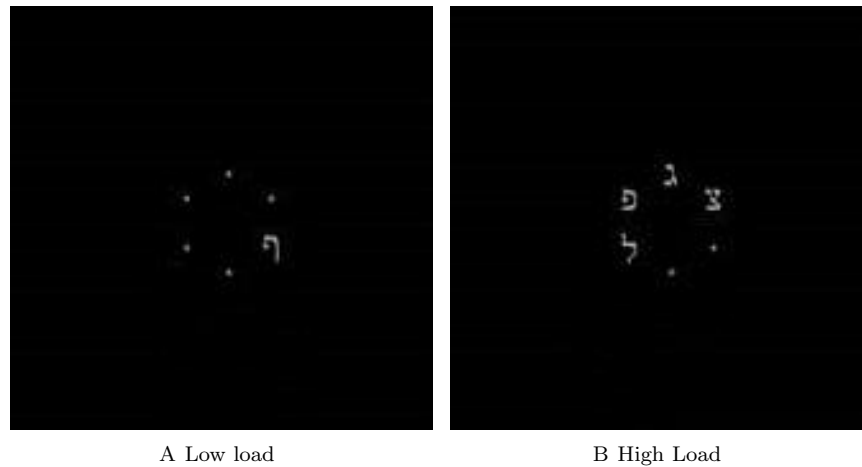


Fig. 2.5: Hebrew Letter Task stimuli. The images shown are zoomed in for viewing purposes; the actual stimuli appear smaller in relation to the screen.

Each block contained five different matrix problems. Each slide contained a 3x3 matrix above eight numbered answer choices (Figure 2.6). The matrices and answer choices were all black and gray over a white background and the answer choices were numbered with red characters. Each matrix was presented for a maximum of 90 seconds or until the participant answered. During the second block, participants listened to spoken sentences while solving the matrices. The participants were asked to determine whether or not each sentence played made sense or not. If the sentence made sense, the participant pressed the “Y” key and if it did not, the participant pressed the “U” key.

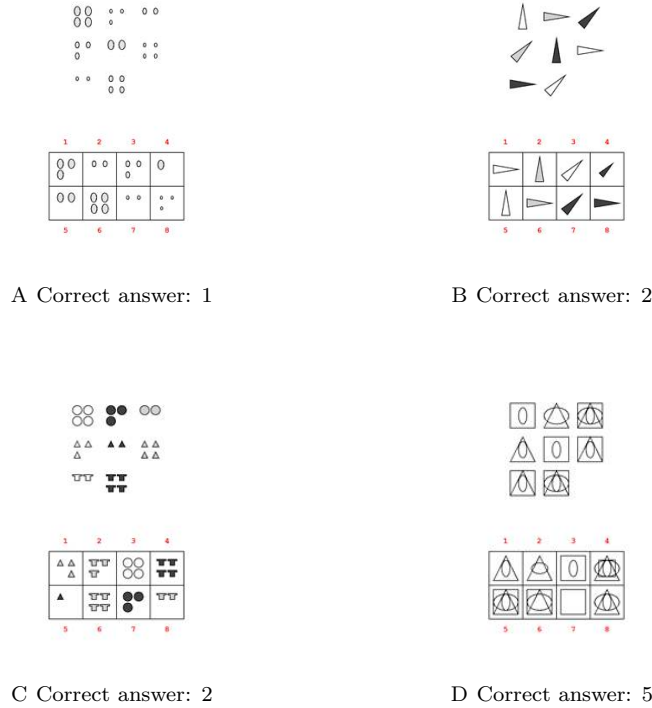


Fig. 2.6: Example stimuli from the Raven's Task.

**3. Results and Discussion.** At the time of the writing, data collection had just been completed, but data analysis had just begun. Preliminary behavioral results show a decrease in accuracy as difficulty increases across tasks as expected. The EEG and ICA analyses are not yet complete.

**4. Conclusions.** Depending on the results of this experiment, the FOVIO eyetracker and the ICA could be a great help in understanding how analysts work and how to improve their efficiency. If these technologies were able to accurately measure cognitive load, researchers could perform experiments on cognitive load in more complex environments. The EEG will probably still remain the primary measure of cognitive load, but in non-memory-related tasks, the eyetracker could really broaden the field. Even if the data does not show the the FOVIO eyetracker and ICA can measure cognitive load, there is still a lot of research to be done, and eyetracking could still become a useful method for assessing cognitive load.

**Acknowledgments.** We would like to thank a couple key people without whom this paper would not have been possible. The cognitive load literature review summary that Mallory Stites wrote was infinitely helpful for finding references on cognitive load and pupilometry. Aidan Thompson was a great help in figuring out how to use  $\text{\LaTeX}$ . Michael Haass provided great feedback and vastly improved the quality of the writing. Finally, we would like to thank Sandia National Laboratories for providing us with such a great opportunity to do this research.

## REFERENCES

- [1] ASA User Manual, Version 4.7, September 2009. ANT BV. Enschede. <http://www.ant-neuro.com>.
- [2] EyeWorks: Software from Eyetracking Inc., 2009.
- [3] *N-Back Paradigm*, Springer New York, 2011, pp. 1718–1719.
- [4] FOVIO, 2014.
- [5] S. AHERN AND J. BEATTY, Pupillary Responses During Information Processing Vary with Scholastic Aptitude Test Scores, *Science*, 205(4412) (1979), pp. 1289–1292.
- [6] P. ANTONENKO, F. PAAS, R. GRABNER, AND T. VAN GOG, Using Electroencephalography to Measure Cognitive Load, *Education Psychology Review*, 22 (2010), pp. 425–438.
- [7] J. BEATTY, Task-Evoked Pupillary Responses, Processing Load, and the Structure of Processing Resources, *Psychological Bulletin*, 91(2) (1982), pp. 276–292.
- [8] C. BERKA, D. LEVENDOWSKI, R. OLMSTEAD, M. POPOVIC, M. CVETINOVIC, AND M. PETROVIC, Real-time Analysis of EEG Indices of Alertness, Cognition, and Memory with a Wireless EEG Headset, *International Journal of Human-Computer Interaction*, 17 (2004), pp. 151–170.
- [9] J. BRISSON, M. MAINVILLE, D. MALLOUX, C. BEAULIEU, J. SERRES, AND S. SIRIOS, Pupil Diameter Measurement Errors as a Function of Gaze Direction in Corneal Reflection Eyetrackers, *Behavior Research Methods*, 45(4) (2013), pp. 1322–1331.
- [10] P. CARPENTER, M. JUST, AND P. SHELL, What One Intelligence Test Measures: A Theoretical Account of the Processing in the Raven Progressive Matrices Test, *Psychological Review*, 97 (1990), pp. 404–431.
- [11] S. FORSTER AND N. LAVIE, Harnessing the Wandering Mind: The Role of Perceptual Load, *Cognition*, 111 (2009), pp. 345–355.
- [12] B. GAGL, S. HAWELKA, AND F. HUTZLER, Systematic Influence of Gaze Position on Pupil Size Measurement: Analysis and Correction, *Behavioral Research Methods*, 43(4) (2011), pp. 1171–1181.
- [13] A. GEVINS AND M. SMITH, Neurophysiological Measures of Working Memory and Individual Differences in Cognitive Ability and Cognitive Style, *Cerebral Cortex*, 10 (2000), pp. 829–839.
- [14] ———, Neurophysiological Measures of Cognitive Workload During Human-Computer Interactions., *Theoretical Issues in Ergonomic Science*, 4 (2003), pp. 113–131.
- [15] W. KLIMESCH, EEG Alpha and Theta Oscillations Reflect Cognitive and Memory Performance: A Review and Analysis, *Brain Research Reviews*, 29 (1999), pp. 169–195.
- [16] W. KLIMESCH, H. SCHIMKE, AND G. PFURTSCHELLER, Alpha Frequency, Cognitive Load and Memory Performance, *Brain Topography*, 5(3) (1993), pp. 241–250.
- [17] W. KLIMESCH, B. SCHACK, AND P. SAUSENG, The Functional Significance of Theta and Upper Alpha Oscillations for Working Memory: A Review., *Experimental Psychology*, 52 (2005), pp. 99–108.
- [18] N. LAVIE, Perceptual Load as a Necessary Condition for Selective Attention, *Journal of Experimental Psychology*, 21(3) (1995), pp. 451–468.
- [19] N. LAVIE, A. HIRST, J. DE FOCKERT, AND E. VIDING, Load Theory of Selective Attention and Cognitive Control, *Journal of Experimental Psychology*, 133(3) (2004), pp. 339–354.
- [20] V. M. LEAVITT, *Standard Progressive Matrices*, Springer New York, 2011, pp. 2368–2368.
- [21] S. MARSHALL, The Index of Cognitive Activity: Measuring Cognitive Workload, *IEEE Human Factors Meeting*, 7 (2002), pp. 5–8.
- [22] ———, Identifying Cognitive State from Eye Metrics, *Aviation, Space, and Environmental Medicine*, 78(5) (2007), pp. 165–175.
- [23] ———, Mental alertness level determination, Mar. 18 2008. US Patent 7 344 251.
- [24] S. MARSHALL, Follow-Solve-Listen Test, 2012.
- [25] S. MARSHALL, Follow-Solve-Listen Test, 2012. Retrieved from <http://www.eyetracking.com/tests/Follow-Solve-Listen>.
- [26] MATLAB, *version 7.10.0 (R2015A)*, The MathWorks Inc., Natick, Massachusetts, 2010.
- [27] L. E. MATZEN, Z. O. BENZ, K. R. DIXON, J. POSEY, J. K. KROGER, , AND A. E. SPEED, Recreating Raven's: Software for Systematically Generating Large Numbers of Raven-like Matrix Problems with Normed Properties, *Behavior Research Methods*, 42(2) (2010), pp. 525–541.
- [28] L. RICHSTONE, M. SCHWARTZ, C. SEIDEMAN, J. CADEDDU, S. MARSHALL, AND L. KAVOUSSI, Eye Metrics as an Objective Assessment of Surgical Skill, *Annals of Surgery*, 252(1) (2010), pp. 177–182.
- [29] J. SWELLER, Cognitive Load During Problem Solving: Effect on Learning, *Cognitive Science*, 12(2) (1988), pp. 257–285.
- [30] P. S. TOOLS, E-Prime 2 (Version 2.0), 2012.
- [31] W. ZHANG AND S. LUCK, Opposite Effects of Capacity Load and Resolution Load on Distractor Processing, *Journal of experimental psychology Human perception and performance*, 41(1) (2015), pp. 22–27.



## FILE FRAGMENT CLASSIFICATION THROUGH SPARSE CODING

FELIX WANG\*, TU-THACH QUACH†, AND FRED ROTHGANGER ‡

**Abstract.** File fragment classification is an important step in the task of file carving in digital forensics. In file carving, a file must be reconstructed based on its content as a result of its fragmented storage on disk or in memory. Existing methods for automatic file type classification from file fragments typically use hand-engineered features such as byte histograms. In this paper, we propose an approach for feature extraction from file fragments based on sparse coding. The advantage of using this approach is two-fold. First, sparse coding is capable of learning features automatically based on how well those features may then be used to reconstruct the original data. In comparison to hand engineering, crafting features to be used in a classifier can be difficult, even with domain expertise, and may still miss differentiating characteristics. Second, by using a sparse basis for our feature space, we are more efficiently able to train state-of-the-art classification algorithms such as recurrent neural networks (RNNs) with Long Short-Term Memory (LSTM) architecture. We demonstrate the capability of this approach by presenting competitive classification results to existing methods without the use of hand-engineered features.

**1. Introduction.** File fragment classification is an important step in the task of file carving in digital forensics. During data recovery, the fragmentation of files on damaged media or memory dumps results in files that appear corrupted or missing even though the data is present [6]. For file carving, in order to reconstruct the complete file from these fragments, analysts must determine which fragment might go with which file. Because the search space of where each fragment may potentially belong to is so large, automated tools are essential to narrowing down what an analyst might process by hand.

The main goal of automatic file fragment classification is to determine which *type* of file (e.g. doc, jpg, pdf) a fragment belongs to depending on its content. In the literature, support vector machines (SVMs) are a powerful machine learning method to classify file fragments [5]. One major downside to SVMs is that the approach typically requires development of a large number of hand-engineered features. Example features are histograms of the bytes (unigrams) within a file fragment, as well as histograms of pairs of bytes (bigrams). More global features include the Shannon entropy over these unigrams and bigrams, or the compressed length of the file fragment [3, 20].

We propose an approach for feature extraction from file fragments based on sparse coding (details in Section 2.1). This approach has a number of advantages over traditionally hand-engineered features. Primarily, the features that are extracted through sparse coding are found automatically, in contrast to features that might need to be laboriously hand-crafted to be suitable for a particular domain [15]. Due to its minimization with respect to reconstruction error, the features found through sparse coding methods furthermore capture a significant amount of information about the domain without needing prior domain knowledge.

Using this approach, we are also able to transform the digital information from its “natural” representation of densely packed bytes into a feature space that is more easily processed by machine learning algorithms. In particular, by decomposing a file fragment into a series of sparse vectors we may more effectively train state-of-the-art classification architectures such as recurrent neural networks (RNNs) with Long Short-Term Memory (LSTM) [7, 8]. By using sparse coding in conjunction with RNNs, we are able to both capture more local features of a file fragment, through the sparse decomposition, as well as more global relationships among these features, through sequence learning by the neural

---

\*University of Illinois at Urbana-Champaign, fywang2@illinois.edu

†Sandia National Laboratories, tong@sandia.gov

‡Sandia National Laboratories, frothga@sandia.gov

network.

The remaining sections of the paper are organized as follows: in Section 2 we describe sparse coding methods as well as the Long Short-Term Memory network architecture; in Section 3 we describe the experimental data and how we used it to train our classifier; in Section 4 we provide and discuss the results from our experiments; and in Section 5 we make concluding remarks.

## 2. Background.

**2.1. Sparse Coding.** Sparse coding is a way of modeling data by decomposing it into sparse linear combinations of elements of a given basis set [12]. That is, a data vector  $y \in \mathbb{R}^m$  may be approximated as multiplying a dictionary matrix  $D \in \mathbb{R}^{m \times k}$  with a sparse vector  $x \in \mathbb{R}^k$ :  $y \approx Dx$ . A vector is said to be  $s$ -sparse when there are at most  $s$  nonzero entries in the vector. Sparse coding, when a dictionary is known (e.g. a wavelet basis), may be formulated as the  $L_1$ -regularized optimization problem,

$$l(y, D) = \min_{x \in \mathbb{R}^k} \frac{1}{2} \|y - Dx\|_2^2 + \lambda \|x\|_1 \quad (2.1)$$

where the cost may be understood as the contributions of the reconstruction error  $\frac{1}{2}\|y - Dx\|_2^2$  and a sparsity penalty  $\lambda\|x\|_1$ , and  $\lambda$  is a regularization parameter. This particular formulation is also known as the *lasso* [19].

When a dictionary is not known, or it is more desirable to learn a dictionary more representative of the data, an extended formulation to the above optimization problem is presented as

$$f_n(Y) = \min_{D \in \mathbb{R}^{m \times k}, X \in \mathbb{R}^{k \times n}} \frac{1}{n} \sum_{i=1}^n \left( \frac{1}{2} \|y_i - Dx_i\|_2^2 + \lambda \|x_i\|_1 \right) \quad (2.2)$$

where  $Y = \{y_1, \dots, y_n\}$  are the data vectors from which we want to learn a dictionary. Because the optimization occurs over both the dictionary  $D$  and the set of sparse vectors  $X = \{x_1, \dots, x_n\}$ , most approaches to dictionary learning iteratively fix one variable while minimizing the other. For learning dictionaries from large input data sets, online or streaming methods have been developed [14].

Sparse coding is typically used in domains where the information being coded contains a great deal of redundancy, such as representing images and other natural signals [14]. In the domain of digital information, one caveat is that the data we wish to analyze and represent is often already in a compressed format. We find that in spite of this, sparse coding is still capable of providing a good decomposition measured with respect to reconstruction error. That is, even in the digital domain, the sparse coding approach is able to extract representative byte patterns to provide a rich basis set with respect to the original data.

A comparison of the application of sparse coding, specifically dictionary learning, of natural images to sparse coding of digital information is shown in Fig. 2.1. In contrast to image patches, where pixels are taken as 2-dimensional space, byte patches may be provided as a 1-dimensional sequence (resized in Fig. 2.1 to 2D for ease of visualization). Once the patches are extracted from the original data, the algorithm for dictionary learning is the same in either domain. Differences are made apparent in the entries of the dictionaries that are learned, corresponding to the differences in how data is structured between the different domains.

While the byte patterns that are extracted using a sparse coding approach potentially span across a relatively large space, the total number of patterns extracted from this space

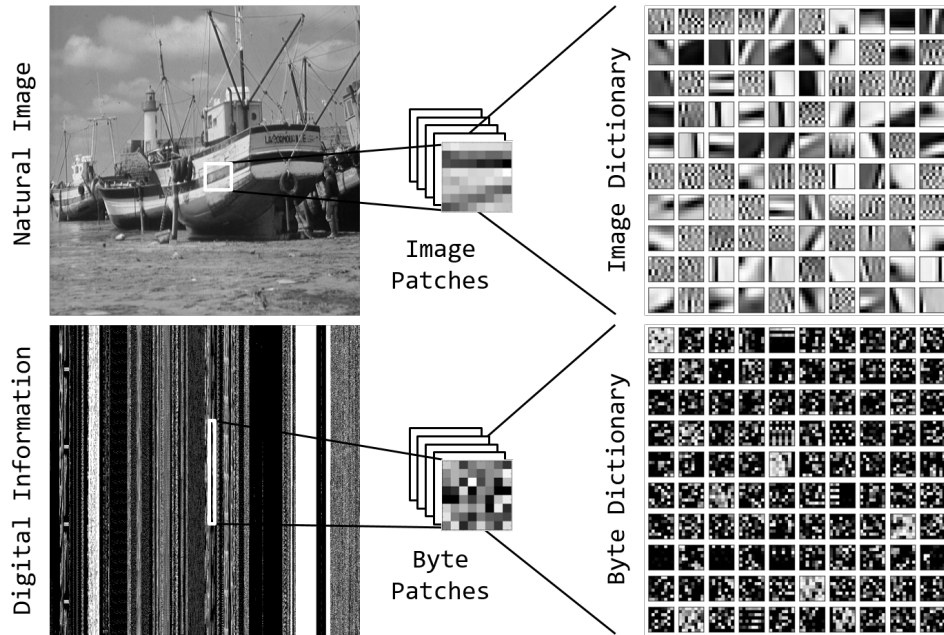


Fig. 2.1: Comparison of representative dictionaries learned from patches of natural images to representative dictionaries learned from digital information

is relatively small. Even for overcomplete dictionaries in image reconstruction applications, the number of entries in the dictionary is only greater by a moderate ratio to the number of pixels in the image patch [12]. In contrast, a common hand-engineered feature of digital information is the byte histogram, consisting of 256 elements. In order to capture pairs of bytes, an engineered feature space is typically expanded by  $256^2 = 65,536$  features, and for triplets of bytes, it is even more. For a modest patch size of  $8 \times 8 = 64$  pixels that may be used in the sparse coding of images, the hand engineering of digital features spanning the same spatial scale is intractable.

Using sparse coding to extract features has some advantages over hand-engineered features in the training of state-of-the-art neural network architectures as well. For the back-propagation algorithm to adjust the weights in a neural network, it is desirable to have the inputs transformed into a space such that the average of each input variable over the training set is close to zero, the covariance of the input variables is scaled to about the same, and that the input variables are uncorrelated if possible [11]. Although sparse coding does not make guarantees as to the covariance of the inputs, due to the sparsity constraint, sparse coding favors features that are uncorrelated. This allows for only a small subset of the features to be active when reconstructing the original data vectors.

**2.2. Long Short-Term Memory.** Neural networks are a powerful method for computing the nonlinear function approximation that is characteristic of classification tasks. Compared to other machine learning algorithms, such as support vector machines (SVMs), neural network architectures tend to scale better when it comes to learning complex high-dimensional functions [1]. This is largely due to the hierarchical nature of “deep” architectures that enable the generalization of data over larger spatial scales while maintaining their ability to perform credit assignment across different levels of representation.

In order to learn sequence data, generalization over temporal scales is needed, and a commonly used architecture in this domain is Long Short-Term Memory (LSTM) [13, 8]. In order for a network to process the history of inputs rather than a snapshot in time, the outputs of neurons are fed back into the network for the next time step. This is the basic idea of a recurrent neural network (RNN). On top of this basic idea, LSTM introduces the concept of a memory cell augmenting the neuron. This allows for the network to retain contextual information over longer temporal scales during training than traditional RNNs without the exponential attenuation of error [10]. In addition to the memory cell, RNNs with LSTM also contain a number of gates that control the state of the memory cell as a function of the input. Together, the basic unit of the LSTM architecture is known as a memory block. For an in depth analysis of the different components and variants of the LSTM architecture, we encourage the reader to review [8].

### 3. Experimental Setup.

**3.1. Dataset.** We use a dataset of 9 common file types that was originally obtained from crawling various internet sites for the purpose of classifying files for identifying data exfiltration [4]. The file types are `html`, `png`, `jpg`, `gif`, `pdf`, `doc`, `elf` (x86-64 Linux executables), `gz` (gzip compressed files), and `aes` (AES-256 encrypted files). There are a total of 4,500 files, with 500 files per file type or class.

From these files, file fragments with length of 512 bytes were extracted by uniformly randomly sampling locations. A fragment size of 512 bytes was selected in order to compare our approach with similar studies in the literature [5]. Specifically, 512 bytes were read starting from a sampled location relative to the start of the file, and the locations were byte-aligned to 64 bytes, which was the size of the byte patches used in the dictionary learning stage (further details in Section 3.1.1). Furthermore, the first 512 bytes of the file were omitted as they frequently contain header information that identifies the file type. We obtain an equal number of file fragments for each file type by extracting 20 file fragments per file for a total of 90,000 file fragments across all of the file types.

**3.1.1. Dictionary Learning.** In sparse coding, the general goal of dictionary learning is to obtain an overcomplete basis set from the data vectors [12]. Similar to image patches or slices of other natural signals, we learn a dictionary over byte patches. Unlike natural signals however, digital information is often already in a compressed format, so the ratio of the elements in the dictionary to the size of the data vector is significantly higher than what is more commonly found in the literature. We chose to use a dictionary of size 1,024 for byte patches of size 64 for a ratio of 16 : 1.

The dictionary was trained using the *spams* sparse coding library which provides a scalable online algorithm for dictionary learning based on stochastic gradient descent [14]. A sample trained dictionary is shown in figure Fig. 3.1. Although there are a few elements with identifiable patterns, the vast majority of the elements appear to be random. This result is not surprising as the dictionary is learned from byte patches from files that have both low entropy (e.g. `html`) and high entropy (e.g. `aes`) with respect to their byte histograms.

**3.1.2. LSTM Classifier.** Using a patch size of 64 bytes, each file fragment may be decomposed into 15 overlapping patches (50% overlap between patches). From these patches, we use the standard *lasso* method (see Section 2.1) to obtain a sequence of sparse vectors for each file fragment. This sequence of sparse vectors is then used as the input to our classifier.

We use a multi-layer LSTM neural network with dropout regularization for classification. Implementation of the network was done using the *Keras* library for Python on top of the *Theano* backend [2, 18]. The input layer is simply composed of the values provided by the

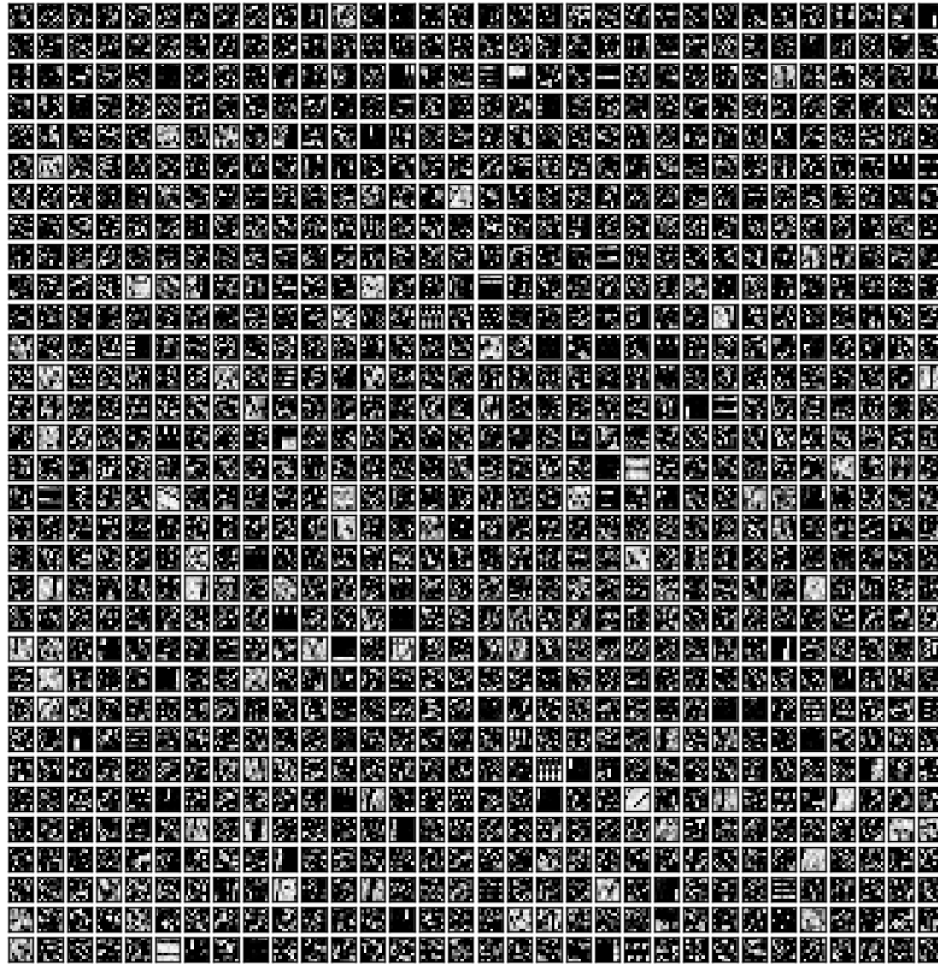


Fig. 3.1: Learned dictionary for byte patches (64-bytes by 1,024 elements)

sparse vectors, where each file fragment provides to the network a sequence of 15 sparse vectors for each of its overlapping byte patches. The output layer is a standard softmax over the 9 different file types such that the output of the network can be distinguished as one of 9 classes. We use three hidden layers with the LSTM recurrent architecture (see Section 2.2) with layer sizes of 128 neurons, 64 neurons, and 32 neurons going from the input layer to the output layer, respectively. The activation function for the hidden layers is the hyperbolic tangent (tanh). All layers are fully connected to the next layer. An illustration of this design is provided in Fig. 3.2.

Between each of the hidden layers and between the output and final hidden layer we employ 50% dropout during training as a regularization method for the network [17]. The mechanism of dropout during training is to “drop out” or zero a percentage of the neurons such that they have no effect on the subsequent layers. The effect of dropout on the network is to encourage learning of redundant paths such that classification accuracy is maintained in spite of this unreliability. During testing, the network is fully connected, and the weights

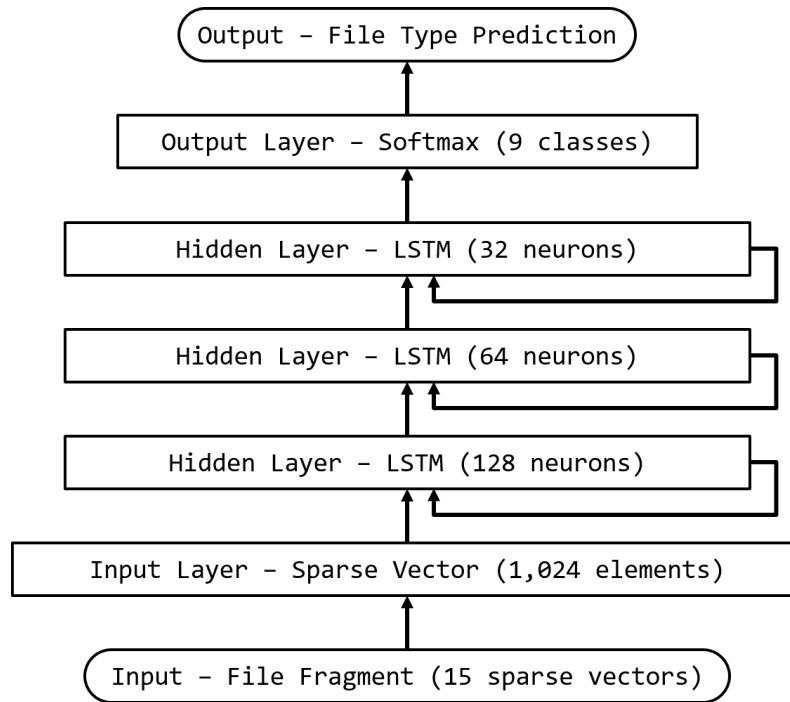


Fig. 3.2: Design of recurrent neural network used for classification

are renormalized with respect to the percentage of dropout used during training.



**4. Results.** For each file, we extracted 20 file fragments for a total of 90,000 file fragments across all of the file types. Training was performed on 85,500 file fragments and testing was performed on the remaining 4,500 fragments. To prevent bias during classification, the file fragments used for testing were drawn from different files altogether than those file fragments that were used for training. Dictionary learning was performed by iterating over randomly shuffled and balanced batches of the training data using a sparsity penalty of  $\lambda = 0.15$  over two epochs with a batch size of 13,500 byte patches. Training was performed with respect to categorical cross-entropy loss over 15 epochs with a batch size of 100 file fragments. We used the RMSProp variant of gradient descent, which adaptively decays the learning rate during training proportional to the average of the squared gradients [9].

We found that our approach produced good results. The average prediction accuracy from file fragments to file types using this approach was 53.31% over 10 separate runs, making it significantly better than random chance ( $1/9 \approx 11.11\%$ ). Using the macro-averaged  $F_1$  metric [16], which provides a weighted average of the precision and recall of a classifier, we obtained an  $F_1$ -score of 53.12%. The confusion matrix of a selected run (accuracy = 53.56%) is given in Table 4.1. The prediction accuracy of 53.31% is competitive with the results of existing approaches to file type classification based on file fragments without the need to hand engineer features. A study by Veenman using Fisher’s linear discriminant achieved an average prediction accuracy of 45% on 11 file types, and a study by Fitzgerald et al. using support vector machines achieved a prediction accuracy of 49.1% over 24 file types [20, 5].

		Predicted class								
		elf	html	png	gif	gz	pdf	jpg	aes	doc
Actual class	elf	<b>86.2</b>	2.0	1.2	2.0	0.2	1.4	0.0	0.0	7.0
	html	0.0	<b>95.0</b>	0.0	0.0	0.0	0.6	0.0	0.0	4.4
	png	1.8	0.0	<b>36.0</b>	7.2	13.2	2.4	10.6	28.2	0.6
	gif	1.2	0.4	8.2	<b>69.2</b>	2.0	2.0	7.2	9.0	0.8
	gz	0.2	0.0	13.8	3.4	<b>36.4</b>	4.2	13.4	28.6	0.0
	pdf	1.0	3.2	17.2	7.0	17.8	<b>21.4</b>	12.0	19.0	1.4
	jpg	0.6	0.2	22.0	7.8	20.8	5.6	17.8	<b>23.0</b>	2.2
	aes	0.0	0.0	21.4	2.6	20.6	0.8	9.8	<b>44.6</b>	0.2
	doc	2.4	5.6	2.4	1.4	4.4	1.8	3.2	3.4	<b>75.4</b>

Table 4.1: Confusion matrix for file fragment classification for one run of the experiment

From these results, we see that fragments more likely to contain identifying patterns, such as the tags in `html` or special formatting in `doc`, were those most easily classified. In the feature space obtained through sparse coding, the fragments of these file types were more likely to be reconstructed using repeating subsets of features, respectively. At the other end, the fragments that gave the classifier the most difficulty were those that possessed relatively high entropy with respect to the byte distribution (e.g. `jpg`, `png`, `gz`, `aes`). Falling into a broader high entropy or “compressed data” category, these fragments were more often confused with each other than confused with those more easily classified file types. A noticeable bias made by our classifier for these fragments was to default to classifying as `aes` when otherwise uncertain. This general behavior when confronted with high entropy fragments is similar to what has been shown in previous studies [4, 5].

Drawing comparisons per file type, we look to the study performed by Fitzgerald et al., which, apart from providing state-of-the-art results, shares the greatest overlap in file types

with ours [5]. Similar to what we provide in Table 4.1, Fitzgerald et al. also provides a representative confusion matrix for a single run of the experiment. We find that those file types that were easily classified through our approach, such as `html` or `doc`, were classified as well or better than the existing method of hand-engineered features. In the case of `doc`, our prediction accuracy was 75.4%, compared to 58.0% in Fitzgerald et al. There were also file types, generally in the broader high entropy category, where our approach performed less well. For example, our prediction accuracy for `png` files was only 36.0%, compared to 62.5% in Fitzgerald et al. At the same time, we had slightly better results for `gz` files, 36.4% compared to 24.8%. Incidentally, file fragments such as `gz` were more commonly mistaken for `png` in Fitzgerald et al. whereas they were more commonly mistaken for `aes` in our study. Overall, both approaches provided comparable results.

**5. Conclusions.** File fragment classification is an important step in the task of file carving in digital forensics. We have presented a sparse coding approach to feature extraction from file fragments for the goal of classification among different file types based on their content. With respect to training a classifier, we found that the features automatically extracted using this approach were competitive with the hand-engineered features of existing approaches. Furthermore, sparse coding leads to efficient training of state-of-the-art classification algorithms such as with LSTM by adhering to appropriate transformations of the input as discussed in Section 2.1. Although sparse coding may be used for automatic feature extraction without the need for hand-engineered features, extensions to the current approach by leveraging domain expertise may be valuable in producing still more accurate classifiers.

## REFERENCES

- [1] Y. BENGIO AND Y. LECUN, Scaling Learning Algorithms towards AI, in Large Scale Kernel Machines, L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, eds., MIT Press, 2007.
- [2] F. CHOLLET, Keras. <https://github.com/fchollet/keras>, 2015.
- [3] G. CONTI, S. BRATUS, A. SHUBINA, B. SANGSTER, R. RAGSDALE, M. SUPAN, A. LICHTENBERG, AND R. PEREZ-ALEMANY, Automated Mapping of Large Binary Objects Using Primitive Fragment Type Classification, Digital Investigation, 7, Supplement (2010), pp. S3 – S12. The Proceedings of the Tenth Annual DFRWS Conference.
- [4] J. A. COX, C. D. JAMES, AND J. B. AIMONE, Complex Adaptive Systems San Jose, CA November 2-4, 2015 A Signal Processing Approach for Cyber Data Classification with Deep Neural Networks, Procedia Computer Science, 61 (2015), pp. 349 – 354.
- [5] S. FITZGERALD, G. MATHEWS, C. MORRIS, AND O. ZHULYN, Using NLP Techniques for File Fragment Classification, Digital Investigation, 9, Supplement (2012), pp. S44 – S49. The Proceedings of the Twelfth Annual DFRWS Conference.
- [6] S. L. GARFINKEL, Carving Contiguous and Fragmented Files with Fast Object Validation, Digital Investigation, 4, Supplement (2007), pp. 2 – 12.
- [7] A. GRAVES AND J. SCHMIDHUBER, Framewise Phoneme Classification with Bidirectional LSTM and Other Neural Network Architectures, Neural Networks, 18 (2005), pp. 602 – 610. IJCNN 2005.
- [8] K. GREFF, R. K. SRIVASTAVA, J. KOUTNÍK, B. R. STEUNEBRINK, AND J. SCHMIDHUBER, LSTM: A Search Space Odyssey, CoRR, abs/1503.04069 (2015).
- [9] G. HINTON, N. SRIVASTAVA, AND K. SWERSKY, RMSProp Gradient Optimization. Lecture Notes, CSC321, University of Toronto, 2014.
- [10] S. HOCHREITER AND J. SCHMIDHUBER, Long Short-Term Memory, Neural Computation, 9 (1997), pp. 1735–1780.
- [11] Y. A. LECUN, L. BOTTU, G. B. ORR, AND K.-R. MÜLLER, *Efficient BackProp*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 9–48.
- [12] H. LEE, A. BATTLE, R. RAINA, AND A. Y. NG, Efficient Sparse Coding Algorithms, in Advances in Neural Information Processing Systems 19, B. Schölkopf, J. Platt, and T. Hoffman, eds., MIT Press, Cambridge, MA, 2006, pp. 801–808.
- [13] Z. C. LIPTON, J. BERKOWITZ, AND C. ELKAN, A Critical Review of Recurrent Neural Networks for Sequence Learning, CoRR, abs/1506.00019 (2015).



- [14] J. MAIRAL, F. BACH, J. PONCE, AND G. SAPIRO, Online Learning for Matrix Factorization and Sparse Coding, *Journal of Machine Learning Research*, 11 (2010), pp. 19–60.
- [15] V. M. PATEL AND R. CHELLAPPA, Sparse Representations, Compressive Sensing and Dictionaries for Pattern Recognition, in *Asian Conference on Pattern Recognition (ACPR)*, 2011.
- [16] C. J. V. RIJSBERGEN, *Information Retrieval*, Butterworth-Heinemann, Newton, MA, USA, 2nd ed., 1979.
- [17] N. SRIVASTAVA, G. HINTON, A. KRIZHEVSKY, I. SUTSKEVER, AND R. SALAKHUTDINOV, Dropout: A Simple Way to Prevent Neural Networks from Overfitting, *Journal of Machine Learning Research*, 15 (2014), pp. 1929–1958.
- [18] THEANO DEVELOPMENT TEAM, Theano: A Python Framework for Fast Computation of Mathematical Expressions, arXiv e-prints, abs/1605.02688 (2016).
- [19] R. TIBSHIRANI, Regression Shrinkage and Selection Via the Lasso, *Journal of the Royal Statistical Society, Series B*, 58 (1994), pp. 267–288.
- [20] C. J. VEENMAN, Statistical Disk Cluster Classification for File Carving, in *Third International Symposium on Information Assurance and Security*, Aug 2007, pp. 393–398.

## Software and High Performance Computing

The articles in this section discuss the implementation of high performance computing (HPC) and productivity software. In many cases, performance improvements and portability are demonstrated for many-core architectures, such as conventional multicore CPUs, the Intel Many Integrated Core coprocessor (MIC), and graphical processing units (GPU).

*Bello-Maldonado, Ober, and Belcourt* use the Kokkos library to implement task parallelism for two computational kernels of an existing MPI-parallel Discontinuous Galerkin code, targeting high-throughput architectures based on GPUs. They demonstrate speedup for both kernels over a wide range of problem sizes and basis polynomial orders.

*Bookey, Holmen, and Hu* create an algebraic multigrid solver designed for problems discretized on structured meshes. The solver is implemented in Kokkos in order to achieve performance portability, and a new Kokkos data structure is also implemented to take advantage of the assumed structure of the matrices. Scaling results are presented and compared with scaling achieved using the MueLu solver, which does not take advantage of grid structure.

*Bourgeois and Wolf* use the High Performance ParalleX runtime system to develop a distributed-memory, task-parallel version of miniTri, a miniapp that performs linear algebra based triangle enumeration, which is a common graph analytics algorithm. They focus on the interplay between task parallelism and data partitioning structure and achieve good scaling to 16 nodes with 24 cores per node.

*Garcia de Gonzalo, Hammond, and Trott* enhance Kokkos' runtime capabilities by developing a dynamic autotuner tool that attempts to find the optimal set of runtime parameters for a particular hardware architecture and a particular matrix sparsity structure. The autotuner achieves this goal by cycling through all possible parameter combinations during the first invocation of the kernel, so many kernel applications are required to overcome this overhead. The automation of this process significantly reduces the burden of the application programmer in achieving performance portability. They present results for several architectures and many matrices, and they demonstrate that in many cases the autotuner significantly increases overall performance.

*Held and Bradley* use Kokkos to implement thread parallelism for the Green-Gauss gradient calculation algorithm, which is a key kernel for the Sandia Parallel Aerodynamics and Reentry Code (SPARC). The algorithm involves looping over all faces in the mesh, and they explain the implementation differences for three different kinds of faces: interior faces, faces on element block boundaries, and faces on the domain boundary. Performance results show superior GPU performance for solve-dominated problems.

*Hota and Moreland* investigate the effects of different optimization strategies on the vectorization performance of illustrative algorithms for visualization. They propose a general method for achieving efficient vector performance with the VTK-m framework in a way that is invisible to the algorithm developer. In some cases, 2x speedup is achieved.

*Juedeman and Cook* use the PerfMiner hardware performance counter collection utility to verify previous findings about bottlenecks in the performance of the LULESH shock hydrodynamics mini-application using the MuMMI application performance profiling and optimization tool. They verify that MuMMI correctly identifies areas of interest for optimization and expand on the previous study by using PerfMiner to keep track of events not recorded by MuMMI in typical runs.

*Kelley, Hoemmen, and Siefert* refactor and optimize the implementation of the block relaxation preconditioning algorithm in Ifpack2, a Trilinos package. They change the memory layout of block matrices to make better use of cache and replace Tpetra MultiVectors with

Kokkos Views, and achieve significant speedup. They also add YAML as a user-friendly alternative to XML for Trilinos ParameterLists.

*Perez and Littlewood* develop a fast recursive descent equation parser in C++, which can be used to apply initial conditions, time-dependent boundary conditions, and other expressions not known at compile time to simulations on high-performance computing platforms.

*Stevens and Edwards* extend Kokkos' task parallel capabilities by implementing task team collective operations, such as reduce and scan, for CUDA, OpenMP, and serial back ends. In doing this, they allow teams of threads to take advantage of internal data parallelism that may exist within a task. Such intra-task data parallelism arises in Cholesky factorization and other commonly used numerical methods. The new implementation utilizes the GPU more fully than native CUDA, since it allows smaller task team sizes and avoids tasks that do not have enough parallel work.

J.B. Carleton

M.L. Parks

December 15, 2016

## ACCELERATING DGM WITH GRAPHICS PROCESSING UNITS AND KOKKOS

PEDRO D. BELLO-MALDONADO\*, CURTIS OBER†, AND KENNETH BELCOURT‡

**Abstract.** The use of numerical computing has seen a rapid increase in the past several years as simulations became the go to approach for product development and scientific discovery. Implemented in the code named DGM, the Discontinuous Galerkin method is one of the most popular computing tools due to its versatility in adapting to meshes with mixed elements, as well as its better accuracy and general numerical properties. A GPU parallelization with Kokkos[5] was added to DGM, achieving improvements of twice the speed when considering the communication time and 20x the speed without communication. This initial implementation aims to serve as proof of concept for further development in the future in order to demonstrate how GPU computing can significantly improve the execution time of DGM.

**1. Introduction.** Given the rise in computing power over the last 50 years, computer simulations have become a key component in the development of new products and services. Companies save millions of dollars every year by testing their products using computers instead of building disposable prototypes. Furthermore, better and better simulation tools and methods have been developed by scientists and engineers to improve the quality of the results and reduce the time to completion. This trend has been seen in all sectors of the economy from engineering to finance, and the demand for skilled workers in these fields is expected to grow significantly in the next years.

While the computing resources are mostly available to anyone through funding agencies like the National Science Foundation (NSF) and the Extreme Science and Engineering Discovery Environment (XSEDE) program or private clusters, utilizing these resources can be daunting for non-computing experts. Furthermore, with newer computing architectures emerging every year it is hard to determine what the best option is when choosing a platform for running a particular simulation. Different systems offer different features that may be well suited for a specific application, while being the wrong choice for a different one. Even if it is clear that a certain platform is the right choice for an application, developing code to maximize resource usage often requires significant effort. For this reason, new programming tools that aim to overcome this issue are being developed to maximize the performance of applications while keeping them simple for non-experts to use. Compilers are also getting smarter and smarter at determining how code translates to machine instructions that make use of the resources effectively. In this work we focus on one of these tools named Kokkos, which is used for on-node parallelism, and a new computing method known as the Discontinuous Galerkin (DG) method for the solution of partial differential equations. Our main objective is to reduce the execution time of a DG code using graphics processing units (GPUs).

The rest of this paper is organized as follows. In Section 2 we introduce the DG method at a very high level and a code named DGM that implements it. Also a survey of GPU computing tools is listed so as to convey where Kokkos lies in the spectrum of programming tools for GPUs. We present here the basics of the Kokkos library and show a simple example so the reader appreciates how easy it is to use. Section 3 describes the approach used to accelerate a portion of DGM that fits well in the GPU programming paradigm. Results of our efforts are summarized in Section 4 with plots of the speedups achieved. Finally we conclude our work in Section 5.

---

\*Dept. of Computer Science, University of Illinois at Urbana-Champaign, belloma2@illinois.edu

†Sandia National Laboratories, ccober@sandia.gov

‡Sandia National Laboratories, kbelco@sandia.gov

**2. Background.** In the realm of scientific computing, the numerical solution of partial differential equations (PDEs) is at the core of scientific discovery and product development. These equations describe many physical systems involving multiple independent and dependent variables, making them almost impossible to solve in closed form (i.e. the solution is given by a function or set of functions satisfying the equations exactly). Many numerical methods have been proposed to address this issue. The general approach in these methods is to decompose the domain (i.e. the physical space where the physics is happening) into many well-defined, disjoint elements and to use them to solve the equations at a set of discrete points (e.g. nodal methods). Other approaches aim to approximate the solution using interpolation where the unknowns become the coefficients in the interpolating function (e.g. modal methods) [13, 2, 3].

In either case the computational complexity of the method can be very expensive if the solution is to be very accurate. Accuracy can be improved by increasing the number of elements, known as  $h$ -refinement, or increasing the polynomial degree of the basis functions in each element, known as  $p$ -refinement. Furthermore, for time evolving PDEs, different time-stepping approaches are possible that can be of explicit or implicit nature, each with its own advantages and disadvantages [8, 10].

**2.1. The Discontinuous Galerkin Method and DGM.** One of the numerical methods for the solution of PDEs is the Discontinuous Galerkin Method (DG). DG methods combine features of the Finite Element method (FE) and the Finite Volume method (FV) to overcome the challenges each one has with respect to accuracy,  $hp$ -adaptivity, and explicit/implicit time-stepping. Solutions to the PDE are locally approximated in the Galerkin sense (i.e. the residual is minimized using the weighted residual formulation at each element), which causes discontinuous solutions at the element interfaces. In order to pick one solution, fluxes are used as in the FV method and in this way the discretization is complete and the solution unique [6].

Many codes are available for the discretization of PDEs with DG. In this work we focus on DGM [15]. The code features discretization in space using mixed nodal and modal representations, as well as a suite of time integrators with many examples of different classes of PDEs (e.g. Navier-Stokes, advection-diffusion, Poisson, etc.).

**2.2. Parallel Computing and Accelerators.** Once the problem being solved gets too large to fit on a single machine or too expensive to run in a timely manner, high performance computing becomes the go to tool. In this computing paradigm, the problem is broken down and distributed across multiple processors that compute the solution in parallel. Significant speed improvements can be achieved with enough computing units. Furthermore, specialized hardware designed to maximize computational throughput can be used to further accelerate the execution time of the application. Currently, two architectures are the most widely adopted in supercomputing clusters: graphics processing units (GPUs) and Intel Xeon Phi coprocessors (the newest version is not a coprocessor anymore. See Intel Knights Landing).

Different programming tools are available for GPU computing. For NVIDIA GPUs the Compute Unified Device Architecture (CUDA) programming model is the standard for programming the accelerators [7]. CUDA is an extension to the C/C++ language that supports GPU computing and provides routines to manage memory and run functions (called kernels) on the GPU. A similar tool that supports other GPU vendors, as well as NVIDIA, is the OpenCL library. Both of these tools work very close to the hardware and require the user to carefully map computational work to the manycore architecture on the GPU. Higher level programming libraries such as OpenACC and OpenMP 4.0 make it easier and faster for programmers to write GPU code without worrying about the underlying architecture at the

expense of performance [14, 12]. In these cases the programmer instruments the sequential code with compiler directives that aid the compiler in deciding how to port the code to the GPU and run it.

A third category of programming models for GPUs and other multithreaded architectures aims to use abstractions to describe the parallel computation so it can be run on different architectures. These tools provide a “write once, run everywhere” approach that enables parallelism without writing code for each different platform. Some examples include the Kokkos library in Trilinos [11], VTK-m [9], and Loo.py [1].

**2.3. The Kokkos Library.** The Kokkos library implements a programming model in C++ for writing performance-portable applications targeting all major high performance computing (HPC) platforms [5]. Architectural features are abstracted so the user need not worry about the specifics of the platform the code is to be run on.

Abstractions in Kokkos have the following forms:

- *Memory Space*: Describes where the data resides (e.g. device memory, host memory)
- *Execution Space*: Describes where the functions are executed (e.g. GPU, XeonPhi, CPU)
- *Execution Policy*: Describes how and where a user function is executed (e.g. concurrently call function  $f(i)$  for  $i = [0, n)$ )
- *Pattern*: Describes the execution pattern (e.g. parallel for, reduce, scan, or task)

In Listing 1 a simple example of Kokkos code is presented. The `Kokkos::View` object is a smart pointer that allocates memory on the memory space defined at compile time or specified in one of the template parameters. Then we launch a kernel on the GPU using `Kokkos::parallel_for` with `num_elem` threads and use the indexing to assign an initial value to each element in the array. In this way, we are letting Kokkos define the underlying distribution of thread blocks (or teams depending on the architecture) on the device. More specific distributions of threads can be possible for finer control, as well. On the other hand, it is possible to change the memory space and execution space by just recompiling the code with the appropriate flags without changing the source code at all. For further details please refer to the Kokkos documentation.

Listing 1: Kokkos example

```
// Create a view object that allocates data on the device
Kokkos::View<double*> array("array", num_elem);

// Launch a kernel and initialize the data
Kokkos::parallel_for(num_elements, KOKKOS_LAMBDA (const int i)
{
    array[i] = initial_value;
});
```

**3. Methodology.** In order to support a large number of elements during simulation, DGM was implemented in parallel using MPI [4]. The code has been run effectively on hundreds of thousands of processors and has been tested on different platforms. However, little effort has been put into migrating portions of the code to GPUs. Under the right circumstances, significant improvements can be achieved with GPUs, and in this section we explain our approach to parallelizing DGM with GPUs and Kokkos.

**3.1. Element Blocks.** The way a domain is meshed significantly impacts the performance of any numerical method for PDEs. Good meshes represent the physical domain as closely as possible, but using some element types may be too expensive or too hard to mesh. DGM supports mixed elements, enabling a large number of possibilities for the modeler to

achieve a good representation. This feature in DGM allows for meshing portions of the domain with the same element shape, while other portions of the mesh may be meshed with elements of different shapes. Computationally speaking this is very efficient since many properties of the element are shared among identical elements thus reducing recomputing of such properties. An example of this is the Jacobian of the element, which is used to map the local element to the reference element. In DGM, an `ElementBlock` class is provided for this purpose. All the elements in an `ElementBlock` object are aligned in memory as to facilitate accessing the data of all the elements with a single pointer. This is a key feature in the parallelization with GPUs since a single copy is possible from the host to the device without worrying about padding the field data of each element in a contiguous array on the device.

**3.2. Functions of Interest.** Maximizing efficiency on GPUs in order to achieve excellent speedups is not always possible if the task being parallelized doesn't fit the computing paradigm. A good understanding of the code and the problem it solves is necessary before parallelization efforts are put in place. Here, we show the functions of interest that we propose to port to the GPU and their mathematical meanings.

**3.2.1. The `inner_product` Function.** For integration reasons, field data of each element needs to be multiplied by the basis functions in the discretization process. This is accomplished by the `inner_product` function in DGM. Mathematically speaking the following steps achieve this. Let  $q_a$ ,  $q_b$ , and  $q_c$  be the number of quadrature points in the  $x$ ,  $y$ , and  $z$  directions, respectively. Also based on these quadrature points, the polynomial degrees of the basis functions take the form  $L_a = q_a - 1$ ,  $L_b = q_b - 1$ , and  $L_c = q_c - 1$ . Let  $\mathbf{u}^{(e)}$  be a vector of size  $q_a q_b q_c$  with the field values for element  $e$  at the quadrature points, and  $e = 1, \dots, E$  with  $E$  being the total number of elements in the block. Let  $\hat{\mathbf{u}}^{(e)}$  be the vector of size  $L_a L_b L_c$  with field values at the modes for each element. Finally, let  $\mathbf{J}$  be the weighted Jacobian vector at the quadrature points and,  $\mathbf{B}_a$  (size  $L_a \times q_a$ ),  $\mathbf{B}_b$  (size  $L_b \times q_b$ ), and  $\mathbf{B}_c$  (size  $L_c \times q_c$ ) be the matrices used to project onto the basis functions. Algorithm 3.1 implements this procedure.

**3.2.2. The `backward_transform` Function:.** While projecting field data onto the basis modes is accomplished with the `inner_product` function, the inverse operation is possible, as well. This is what the `backward_transform` function does. Using the same notation as before, Algorithm 3.2 shows the implementation in this case.

**3.3. Parallelization.** As described, both Algorithm 3.1 and 3.2, can be parallelized in several ways. For starters, element computations are independent of each other. Each individual element could potentially be assigned to a thread and computed in parallel. Furthermore, the individual computations of each element are basic linear algebra routines that benefit from manycore architectures. Listing 2 and 3 show the implementation of two key portions of Algorithm 3.1: multiplying the Jacobian by the field (an element-to-element vector multiplication that is represented as a diagonal matrix multiplied by a vector) and a simple matrix-matrix multiplication. A very similar code is used for `backward_transform` since the math is effectively the same.

In Listing 2 we see a couple of new Kokkos features not shown before. Mainly the `TeamPolicy` object. This object defines how threads are distributed and organized in teams. The number of teams and the number of threads per team is given in the constructor of the object. For each team, a `TeamThreadRange` object organizes the threads and gives them IDs to be referenced in the body of the function where they are used. The CUDA equivalent of teams and thread range, in this case, is thread blocks and threads in a one-dimensional configuration. Kokkos manages the mapping automatically and takes care of

**Algorithm 3.1** inner\_product function description

**Input:** Number of quadrature points in each dimension,  $q_a$ ,  $q_b$ , and  $q_c$ . Number of modes in each dimension,  $L_a$ ,  $L_b$ , and  $L_c$ . Weighted Jacobian of the element,  $J$ . Matrices to project onto the basis functions,  $B_a$ ,  $B_b$ , and  $B_c$ . Field,  $u$ , for each element. Number of elements,  $E$ . All matrices and vectors are assumed to be laid out in memory in row-major ordering for reshaping purposes.

**Output:** Projected field onto the basis modes,  $\hat{u}$ , for all the elements

```

1: procedure INNER_PRODUCT
2:   for  $e = 1$  to  $E$  do
3:      $u_w = \text{diag}(J) u^{(e)}$ 
4:      $U_w = \text{reshape } u_w \text{ from } q_a q_b q_c \text{ to } q_a \times q_b q_c$ 
5:      $H' = B_a U_w$ 
6:      $H'' = \text{zero matrix of size } L_a \times L_b q_c$ 
7:     for  $l_a = 1$  to  $L_a$  do
8:        $H^{(l_a)} = \text{reshape } H'^{(l_a)} \text{ from } q_b q_c \text{ to } q_b \times q_c$ 
9:        $R = B_b H^{(l_a)}$ 
10:       $H''^{(l_a)} = \text{reshape } R \text{ from } L_b \times q_c \text{ to } L_b q_c$ 
11:       $H'' = \text{reshape } H'' \text{ from } L_a \times L_b q_c \text{ to } L_a L_b \times q_c$ 
12:       $\hat{U} = H'' B_c^\top$ 
13:       $\hat{u}^{(e)} = \text{reshape } \hat{U} \text{ from } L_a L_b \times L_c \text{ to } L_a L_b L_c$ 
14:   return  $\hat{u}$ 

```

Listing 2: Element-to-element multiplication of the field,  $u$ , by the weighted Jacobian,  $J$

```

typedef Kokkos::Cuda Space;
typedef Kokkos::TeamPolicy<Space> TeamPolicyExec;
typedef Kokkos::TeamPolicy<Space>::member_type MemberTypeExec;

Kokkos::parallel_for(TeamPolicyExec(num_elem, num_threads),
  KOKKOS_LAMBDA (const MemberTypeExec& team_member)
  {
    const int e = team_member.league_rank();

    Kokkos::parallel_for(Kokkos::TeamThreadRange(team_member, q_a * q_b * q_c),
      [=] (const int i)
      {
        u[e * (q_a * q_b * q_c) + i] *= wJ[i];
      });
  });

```

the correct indexing at the boundaries. Finally, in the body of the function each element is indexed by  $e * (q_a * q_b * q_c)$  and each quadrature point is multiplied by the Jacobian. Notice that this is only possible because the elements in an `ElementBlock` have consecutive memory addresses separated by blocks of size  $q_a q_b q_c$ . We are not showing the view allocation here, but it suffices to say that  $u$  and  $wJ$  reside in global memory on the device and have the correct data.

For Listing 3, a basic matrix-matrix multiplication is implemented for each element. The structure is similar to that of the element-to-element multiplication code, but another layer of parallelism is included. The `ThreadVectorRange` object adds an extra dimension to the thread teams. For that reason, in the constructor of the `TeamPolicy` object, two arguments are given defining the block size in each dimension. The Kokkos library computes the number of blocks necessary to cover the ranges in `TeamThreadRange` and `ThreadVectorRange`,



**Algorithm 3.2** backward\_transform function description

**Input:** Number of quadrature points in each dimension,  $q_a$ ,  $q_b$ , and  $q_c$ . Number of modes in each dimension,  $L_a$ ,  $L_b$ , and  $L_c$ . Matrices to project onto the basis functions,  $\mathbf{B}_a$ ,  $\mathbf{B}_b$ , and  $\mathbf{B}_c$ . Projected field,  $\hat{\mathbf{u}}$ , for each element. Number of elements,  $E$ . All matrices and vectors are assumed to be laid out in memory in row-major ordering for reshaping purposes.

**Output:** Field value on the grid points,  $\mathbf{u}$ , for all the elements

```

1: procedure BACKWARD_TRANSFORM
2:   for  $e = 1$  to  $E$  do
3:      $\hat{\mathbf{U}} = \text{reshape } \hat{\mathbf{u}}^{(e)} \text{ from } L_a L_b L_c \text{ to } L_a \times L_b L_c$ 
4:      $\hat{\mathbf{H}}' = \mathbf{B}_a^\top \hat{\mathbf{U}}$ 
5:      $\hat{\mathbf{H}}'' = \text{zero matrix of size } q_a \times q_b L_c$ 
6:     for  $q = 1$  to  $q_a$  do
7:        $\hat{\mathbf{H}}^{(q)} = \text{reshape } \hat{\mathbf{H}}'^{(q)} \text{ from } L_b L_c \text{ to } L_b \times L_c$ 
8:        $\hat{\mathbf{R}} = \mathbf{B}_b^\top \hat{\mathbf{H}}^{(q)}$ 
9:        $\hat{\mathbf{H}}''^{(q)} = \text{reshape } \hat{\mathbf{R}} \text{ from } q_b \times L_c \text{ to } q_b L_c$ 
10:       $\hat{\mathbf{H}}'' = \text{reshape } \hat{\mathbf{H}}''^{(q)} \text{ from } q_a \times q_b L_c \text{ to } q_a q_b \times L_c$ 
11:       $\mathbf{U} = \hat{\mathbf{H}}'' \mathbf{B}_c$ 
12:       $\mathbf{u}^{(e)} = \text{reshape } \mathbf{U} \text{ from } q_a q_b \times q_c \text{ to } q_a q_b q_c$ 
13:   return  $\hat{\mathbf{u}}$ 

```

automatically. In the body of the function, a simple loop is used to compute each element of the output matrix, which is equal to the inner product of a row of the first matrix and the column of the second matrix. In this way, the order of operations does not affect the value of the output compared to the sequential implementation, and both values should be exactly the same. This was not by design, and some accuracy could be sacrificed in spite of more parallelism but that is left for future optimizations of the code. The remaining portions of the routine have similar implementations and thus are not shown here.

Listing 3: Matrix-matrix multiplication of the basis function matrix and the reshaped Jacobian-multiplied field

```
Kokkos::parallel_for(TeamPolicyExec(num_elem, block_y, block_x),
  KOKKOS_LAMBDA (const MemberTypeExec& team_member)
{
  int e = team_member.league_rank();

  Kokkos::parallel_for(Kokkos::TeamThreadRange(team_member, num_modes),
    [=] (const int i)
    {
      Kokkos::parallel_for(Kokkos::ThreadVectorRange(team_member, q_b * q_c),
        [=] (const int j)
        {
          double* A = Ba;
          double* B = u + e * (q_a * q_b * q_c);
          double* C = Hp + e * (L_a * q_b * q_c);

          double sum = 0.0;

          for (int k = 0; k < q_a; k++)
          {
            sum += A[i * q_a + k] * B[k * (q_b * q_c) + j];
          }

          C[i * (q_b * q_c) + j] = sum;
        });
    });
});
```

**4. Results.** Our tests were run on the Shannon test bed in interactive mode. The specifications per node are:

- Two 8-core Sandy Bridge Xeon E5-2670 @ 2.6GHz
- 128GB DDR3-1600MHz
- 2x NVIDIA K20 GPUs
  - 2688 CUDA Cores
  - 732MHz Clock rate
  - 5760MB Global memory
- RHEL 6

We performed 5 runs of each test and averaged the results. In analyzing the effectiveness of our algorithm, the number of elements is varied as well as the polynomial degree of the basis functions in each element. DGM provides an executable to generate a cubic mesh with hex elements of the same structure. Listing 4 shows the command to generate a computational mesh. Notice that the code generates a linear grid of multiple elements of size  $1 \times 1 \times 1$  in the  $x$ -direction. Other tests with a cubic grid of elements did not influence the speedups and the results were equivalent.

Listing 4: Mesh generation with DGM (Python string formatting)

```
"dgm_mesh.exe -nsd 3 -nx %d -ny 1 -nz 1 -x0 0.0 -y0 0.0 -z0 0.0 -Lx %f -Ly 1.0 -Lz 1.0 -r
3d" % (E, E)
```

Figures 4.1-4.4 show the measured speed improvements for a HexBlock of size 8, 16, 32, 64, 128, 256, and 512, with and without the cost of communication for both functions of interest. In this case, communication means the data transfer between the CPU and the GPU. As we can see, with more grid points to work with, the speed improvement increases since the communication cost has less effect on the overall computation. Furthermore, a higher polynomial degree also means better performance, as a higher polynomial means

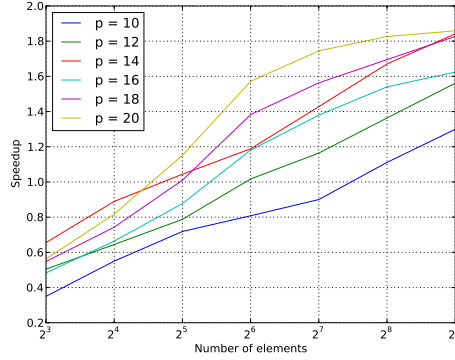


Fig. 4.1: Overall speedup of inner\_product function

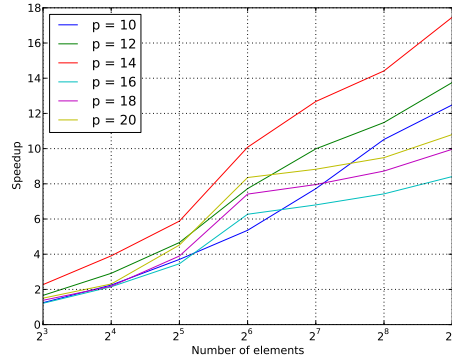


Fig. 4.2: Speedup of inner\_product function without communication

more grid points per element. However, this trend is not seen when the communication cost is not taken into account. As we can see from Figure 4.2 or 4.4 the configuration with the best performance is a hex element with  $p = 14$ . This is true regardless of the number of elements. We believe that, because  $p = 14$  means  $p + 2 = 16$  grid points in each direction, an element with  $16^3 = 4096$  grid points fits exactly in one memory page and so it is very efficient, but further investigation is needed.

When looking at the overall execution and not just each individual function, an overall speedup gain is also achieved under the right conditions. Figure 4.5 shows these results. As we can see, a total of 5% improvement is seen for a run with 1000 time-steps. We expect that with more portions of the code ported to the GPU larger improvements will be seen.

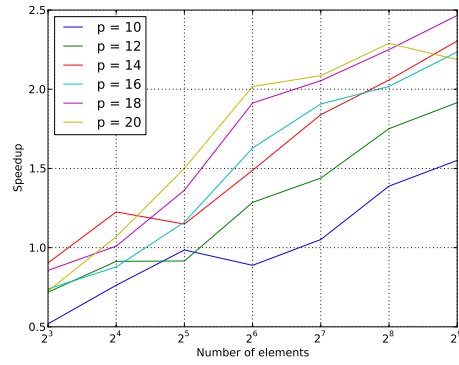


Fig. 4.3: Overall speedup of backward\_transform function

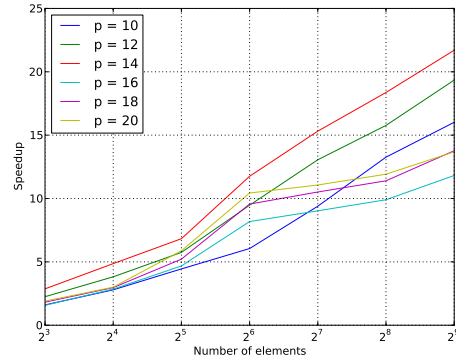


Fig. 4.4: Speedup of backward\_transform function without communication

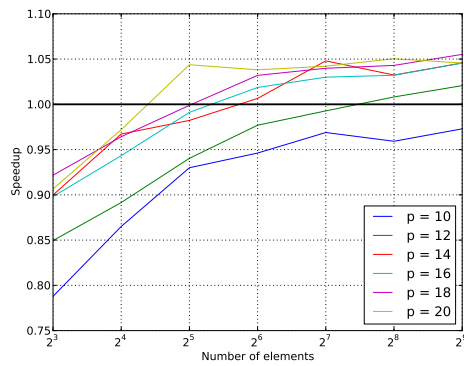


Fig. 4.5: Total performance gain for a test run with 1000 time-steps. Time measured from the beginning of the application to the end including all aspects of the execution (e.g. reading input files, setup, etc)

**5. Conclusions.** Numerical applications benefit greatly from the computing power of GPUs since computation can be broken into small, lightweight pieces that can be mapped efficiently onto the many core architecture of the GPU. However, in order to maximize performance, a careful mapping of the tasks is needed, and this can't be done without significant understanding of the architecture. Tools like Kokkos facilitate the parallelization process by abstracting the hardware and generalizing parallel work so multiple platforms can be covered. We showed that improvements of twice the speed are possible even when the communication is present and a relatively low number of elements are available. The results show that increasing the total amount of work manifests into higher speedups. Furthermore, if communication is ignored, huge improvements are seen since transferring data from the CPU to the GPU is the bottleneck of GPU computing in general. Future generations of GPUs will share the memory space with the CPU so we expect to see even faster GPU code in the near future. On the other hand, this initial approach to GPU parallelization of DGM aims to demonstrate how GPUs can be used with Kokkos in accelerating the application. Based on the results obtained, we can move forward and target more portions of the code that fit well on the GPU, and we expect to see a reduced computing time of DGM as a whole.

## REFERENCES

- [1] ANDREAS KLÖCKNER, Loo.py: Transformation-Based Code-Generation for GPUs and CPUs, in Proceedings of ARRAY '14: ACM SIGPLAN Workshop on Libraries, Languages, and Compilers for Array Programming, Edinburgh, Scotland., 2014, Association for Computing Machinery.
- [2] CONSTANTINE POZRIKIDIS, *Introduction to Finite and Spectral Element Methods Using MATLAB, Second Edition*, Chapman and Hall/CRC, Boca Raton, FL, USA, 2014.
- [3] DAVID V. HUTTON, *Fundamentals of Finite Element Analysis*, Tata McGraw Hill India, 2003.
- [4] M. P. FORUM, MPI: A Message-Passing Interface Standard, tech. rep., Knoxville, TN, USA, 1994.
- [5] H. CARTER EDWARDS AND CHRISTIAN R. TROTT, Kokkos: Enabling Performance Portability Across Manycore Architectures, in 2013 Extreme Scaling Workshop (xsw 2013), Aug 2013, pp. 18–24.
- [6] JAN S. HEASTHAVEN AND TIM WARBURTON, *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications*, Springer-Verlag New York, 2008.
- [7] JOHN NICKOLLS AND IAN BUCK AND MICHAEL GARLAND AND KEVIN SKADRON, Scalable Parallel Programming with CUDA, Queue, 6 (2008), pp. 40–53.
- [8] K. SUBBARAJ AND M.A. DOKAINISH, A Survey of Direct Time-Integration Methods in Computational Structural Dynamics-II: Implicit Methods, Computers & Structures, 32 (1989), pp. 1387–1401.
- [9] KENETH MORELAND AND CHRISTOPHER SEWELL AND WILLIAM USHER AND LI-TA LO AND JEREMY MEREDITH AND DAVID PUGMIRE AND JAMES KRESS AND HENDRIK SCHROOTS AND KWAN-LIU MA AND HANK CHILDS AND MATTHEW LARSEN AND CHUN-MING CHEN AND ROBERT MAYNARD AND BERK GEVECI, VTK-m: Accelerating the Visualization Toolkit for Massively Threaded Architectures, IEEE Computer Graphics and Applications, 36 (2016), pp. 48–58.
- [10] M.A. DOKAINISH AND K. SUBBARAJ, A Survey of Direct Time-Integration Methods in Computational Structural Dynamics-I: Explicit Methods, Computers & Structures, 32 (1989), pp. 1371–1386.
- [11] MICHAEL A. HEROUX AND ROSCOE A. BARTLETT AND VICKI E. HOWLE AND ROBERT J. HOEKSTRA AND JONATHAN J. HU AND TAMARA G. KOLDA AND RICHARD B. LEHOUCQ AND KEVIN R. LONG AND ROGER P. PAWLOWSKI AND ERIC T. PHIPPS AND ANDREW G. SALINGER AND HEIDI K. THORNQUIST AND RAY S. TUMINARO AND JAMES M. WILLENBRING AND ALAN WILLIAMS AND KENDALL S. STANLEY, An Overview of the Trilinos Project, ACM Trans. Math. Softw., 31 (2005), pp. 397–423.
- [12] OPENMP ARCHITECTURE REVIEW BOARD, OpenMP Application Program Interface Version 4.5, November 2015.
- [13] PHILIPPE G. CIARLET, *Finite Element Method for Elliptic Problems*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002.
- [14] SANDRA WIENKE AND PAUL SPRINGER AND CHRISTIAN TERBOVEN AND AN MEY, DIETER, OpenACC: First Experiences with Real-World Applications, in Proceedings of the 18th International Conference on Parallel Processing, Euro-Par'12, Berlin, Heidelberg, 2012, Springer-Verlag, pp. 859–870.
- [15] SCOTT COLLIS, DGM. For more information contact sscoll@sandia.gov.

## KOKKOS-BASED STRUCTURED GRID MULTIGRID SOLVER

ZACHARY A. BOOKEY\*, JOHN K. HOLMEN†, AND JONATHAN J. HU‡

**Abstract.** Multigrid is a provably scalable solution method for certain linear systems that arise from discretizing elliptic partial differential equations. This project sought to create a multigrid solver that is designed for problems discretized on structured meshes. The resulting solver utilizes the Kokkos library [15] to allow for performance portability across many architectures. Kokkos was used in creating a sparse matrix storage object that uses knowledge of the problem structure in an attempt to avoid some memory accesses needed for the compressed row storage format. A so-called “black box” multigrid solver is created using Kokkos, a structured grid matrix format, and coarsening based on ideas in [20]. Various smoothing algorithms are compared using this proof-of-concept.

**1. Introduction.** Multigrid methods are a class of provably scalable solvers for certain linear systems arising from elliptic partial differential equations (PDEs). Efficiently solving PDEs is a key computational task in many computer simulations, and the multigrid methods in the Sandia/Trilinos [10] project are often employed. These methods are algebraic in nature, meaning they do not leverage simulation-specific information. In fact, the existing methods presume that the underlying meshes are unstructured.

In this paper, we present a multigrid method for systems arising from fully structured hexahedral meshes. The structured nature of these meshes allows matrix storage and operations to be handled more efficiently than is possible for general unstructured sparse matrices. This in mind, the proposed algorithm is algebraic, rather than geometric. This is advantageous because it frees the application from having to generate additional data for the multigrid method, apart from the linear system itself. The coarse matrices that are produced by the algorithm discussed here are fully structured, and, thus, amenable to the same improvements in storage and operation efficiency. We refer to this algorithm as structured grid algebraic multigrid, or SGAMG for short. SGAMG leverages Sandia’s Kokkos library [15] to manage low-level data storage and facilitate node-level parallelism. By using Kokkos, we hope to gain performance portability across various architectures.

This paper provides a brief introduction to multigrid and the Kokkos software package in sections 1.1 and 1.2. We discuss the Kokkos class that was created to take advantage of the assumed structure of these matrices in section 2. We then describe the SGAMG algorithm used to both create these matrices and solve the problems they represent in section 3. We cover basic smoothing algorithms and pseudocode to demonstrate how they were adapted to use the proposed data format in section 4. Finally, we compare SGAMG to a similar multigrid driver within the MueLu software package [16] in section 5.

**1.1. Introduction to multigrid.** Multigrid is a provably scalable method for solving certain classes of linear systems  $Ax = b$ .<sup>1</sup> In a multigrid solver, a *hierarchy* of coarser resolution systems is generated and used to accelerate the solution of the problem of interest. This problem of interest, typically supplied by the application, is referred to as the *fine grid system*. Each linear system within the hierarchy has a solver associated with it, called a *smoother*. The smoother is typically an inexpensive iterative method such as Jacobi or Gauss-Seidel, but may also be a more expensive solve, such as an incomplete factorization. Information is transferred between the various systems in the hierarchy by matrices called *restrictors* and *prolongators*. Collectively, these matrices are called *grid transfers*. The

\*Saint John’s University/University of Minnesota, zabookey@csbsju.edu/booke038@umn.edu

†Scientific Computing and Imaging Institute, University of Utah, jholmen@sci.utah.edu

‡Sandia National Laboratories, jhu@sandia.gov

<sup>1</sup>Multigrid can be applied to nonlinear systems, as well, but that is not the focus of this paper.

---

**Algorithm 1.1** Pseudocode for an  $N$ -level multigrid V-cycle schedule. Here,  $k$  is the grid level,  $P$  is the prolongator,  $R$  is the restrictor, and  $S$  is the smoother.

---

```

 $A_0 = A$ 
function VCycle( $A_k, b, x, k$ )
  // Solve  $A_k x = b$  ( $k$  is current grid level)
   $x = S_k(A_k, b, x)$ 
  if ( $k \neq N - 1$ ) then
     $r_{k+1} = R_k(b - A_k x)$ 
     $A_{k+1} = R_k A_k P_k$  // Normally performed during the startup phase
     $z = 0$ 
    VCycle( $A_{k+1}, r_{k+1}, z, k + 1$ )
     $x = x + P_k z$ 
     $x = S_k(A_k, b, x)$ 
  else
     $x = S_{c_{N-1}}(A_{N-1}, b, x)$ 

```

---

pseudocode for a multigrid *V-cycle*, a typical schedule for visiting the different multigrid levels, is given in Algorithm 1.1. Note that, subscripts featured within Algorithm 1.1 are excluded throughout the remaining discussion as this work emphasizes two-level methods.

The main idea of multigrid is that the smoother for the fine grid system efficiently reduces certain high energy solution error components. All other error components are reduced on some coarser level of the hierarchy by that level's smoother.

There are two main classes of multigrid methods. The first class, geometric multigrid, requires rediscretizing the PDE on a coarser resolution mesh. Grid transfers are created by appropriately interpolating between two levels. In the case of finite elements, this boils down to interpolating between finite element basis functions on two consecutive levels. In any case, the grid transfers are problem-specific and require close interaction with the application. The second class of multigrid methods is algebraic multigrid. In this class, the application supplies only the fine grid system, and the multigrid algorithm generates the grid transfers and coarse grid systems. For more information on multigrid, see [13, 19].

**1.2. Introduction to Kokkos.** With the increasing diversity among current and emerging architectures, it is growing more difficult to write a single piece of code that performs optimally across multiple architectures. To address this challenge, the Kokkos library was developed to facilitate performance portability via an added layer of abstraction between a developer and the target architecture. This abstraction allows a developer to compile code optimized for a target architecture, which is specified at compile time. Kokkos accomplishes this through the use of `memory_spaces` and `execution_spaces`. These constructs are used to manage where data resides and where a function executes. For example, `memory_spaces` allow Kokkos to layout data in a manner amenable to the target architecture. Similarly, `execution_spaces` allow Kokkos to dispatch parallel code leveraging backends for a variety of parallel programming models. Currently, Kokkos supports the following `execution_spaces`:

- Serial
- PThreads [12]
- OpenMP [7]
- Cuda. [1]

A Kokkos View is a multidimensional array that stores data in the user specified `memory_spaces`. On top of allowing the user to specify which `memory_spaces` to use,

Views allow the user to specify different data layouts and memory access patterns, or use the defaults for the given `memory_spaces`. Views also manage their own allocations and will deallocate themselves when they are no longer referenced.

Kokkos parallel dispatch consists of three types of kernels: `for`, `reduce`, and `scan`. `parallel_for` executes a generic `for` loop by dividing the work among the user specified number of threads. `parallel_reduce` acts like the former, but asynchronously updates a variable with either the default or user defined `init` and `join` methods to handle how each thread initializes its update variable and combines its results with the rest of the threads. `parallel_scan` keeps a running sum of the values in a View and places the current sum in the corresponding location.

**2. Matrix Data Storage for Structured Meshes.** Since the underlying mesh is structured, the resulting matrix features a regular stencil pattern. Therefore, the first objective was to create a Kokkos-based matrix that will exploit this inherent structure. These structured matrices are often sparse which encourages the use of a sparse matrix storage format. The resulting structured grid matrix (SG) format is motivated by the compressed row sparse (CRS) matrix format but utilizes some knowledge of the structure of the matrix to avoid unnecessary memory accesses.

The traditional CRS format stores three arrays, `rows`, `inds`, and `vals`. `inds` has length equal to the number of matrix nonzeros and contains the corresponding column index for each nonzero matrix entry. `vals` has length equal to the number of matrix nonzeros and contains all the nonzero values in the matrix. `rows` has length  $n + 1$ , where  $n$  is the number of rows in the matrix. `rows` $\{i\} \dots \text{rows}\{i + 1\} - 1$  are the indices into `inds` and `vals` for column and value information for row  $i$ .

The proposed SG format utilizes knowledge about both the grid and stencil to store two arrays. The first array stores values corresponding to each nonzero entry within the sparse matrix. The second array stores column indices corresponding to each nonzero entry within the sparse matrix. Figure 2.1 demonstrates the differing storage patterns utilized by the CRS and SG formats.

Note that in Figure 2.1, it is possible for the second array used by the SG format to feature unused entries within a given row (as indicated by the  $-1$ ). In the event that a row features unused entries, a  $-1$  is stored in these locations to help identify the termination of the row. This has been done to simplify the process of obtaining a given row's nonzero entries. When using the CRS format, a user must compute the start and end indices of such entries using the row map and then use these values to obtain the corresponding indices and values for each nonzero entry. When instead using the SG format, a user need only multiply the row index by the row width. This, however, is achieved at the expense of additional storage for unused entries and conditionals used to indicate the termination of a row (e.g. when performing matrix-vector multiplication).

The SG format can be found in the `SGMatrix_def.hpp` and `SGMatrix_decl.hpp` files. This templated structure allows the user to specify what type of data the values are, the type of ordinals used to access data, which Kokkos Device to use, and specific memory access traits. `SGMatrix` consists of a pair of two-dimensional Views of size  $n \times s$  where  $n$  is the number of rows and  $s$  is the size of the structures stencil. These views are stored as 1-dimensional arrays on which ever `memory_spaces` is denoted within the Kokkos Device. The SG format can be sorted so that for each rows chunk, the diagonal value is the first value available and the rest of the values are in descending order so that the  $-1$  invalid flags are all at the end of the chunk.



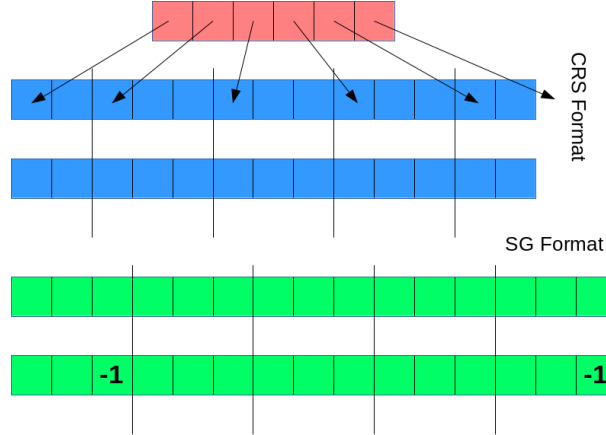


Fig. 2.1: The blue and salmon arrays correspond to the storage utilized by the CRS format. Specifically, the salmon array stores the map of row pointers, which are used to index the blue arrays when storing column indices and values for nonzero entries within a sparse matrix. The green arrays correspond to the storage utilized by the SG format. Specifically, the green arrays store the column indices and values for nonzero entries within a sparse matrix. Note that, locations featuring a  $-1$  are unused and indicate that a given row features fewer nonzero entries than the size of the stencil. This example depicts the storage of a  $5 \times 5$  tridiagonal matrix.

**3. The SGAMG Algorithm.** Prior to starting a `V-cycle` (or similar scheme), two structured matrices must be available in order to setup the SGAMG hierarchy. Specifically, one must have the top-level (finest) matrix describing the target problem,  $A_0$ . This matrix would normally be supplied by the calling application, but for this paper, it is generated in the driver program. In order to generate  $A_0$ , we first generated a structured mesh that establishes the adjacency list for points within the top-level matrix. Using this mesh, the top-level matrix is constructed by identifying adjacent neighbors and applying stencils to describe the target problem. The second required matrix is the prolongation matrix,  $P$ . Once this and  $A_0$  are available, we can then compute the Galerkin product,  $P^T A_0 P$ , which generates the corresponding coarse matrix,  $A_1$ . For purposes of this paper, we consider only the two-level method, but in theory, this idea can be applied recursively.

The key idea of the SGAMG algorithm is to apply *semi-coarsening* in each mesh direction, and then combine the resulting 1D interpolation basis functions into a final 2D or 3D interpolation basis function. The semicoarsening used in SGAMG is the same as the semi-coarsening algorithm described within [20]. The approach in that paper is to collapse the PDE underlying  $A_0$  to a unidirectional operator along a single dimension. This is accomplished by summing contributions from adjacent points within each row/column of the stencil for a given point within  $A_0$  as depicted in Figure 3.1.

Whereas [20] only considers semicoarsening in one or two dimensions, our approach semicoarsens in all mesh directions. The result of this semi-coarsened process is a set of unidirectional basis vectors for each coarse point within the matrix: two for 2D and three for 3D. These basis vectors are then combined via a tensor product, yielding a 2D or 3D basis vector for each coarse point. Once all coarse points have been processed, the resulting prolongation column vectors are combined into the final prolongation matrix  $P$ , at which

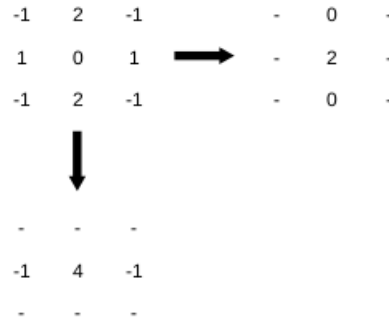


Fig. 3.1: Horizontal and vertical collapsing of a 9-point stencil. When collapsing horizontally, elements within a given column are summed and the results are stored in the middle row. When collapsing vertically, elements within a given row are summed and the results are stored in the middle column.

point the coarse grid can be generated via the Galerkin product.

**3.1. Software implementation.** To implement this functionality within our proof-of-concept, `SGMatrix_matGen.hpp` was created with three supporting functions: `buildTopLevel()`, `buildCollapsed()`, and `buildP()`. When implementing this, an effort was made to generalize indexing via the `Tpetra::Map` [9] object. This was done to enable use of the `Tpetra::Import` and `Tpetra::Export` classes to ease the integration of MPI [6] in the event that distributed-memory parallelism is introduced as future work. Additionally, an effort was made to generalize `buildCollapsed()` and `buildP()` to accept an arbitrary structured matrix in the SG format. This was done to facilitate the generation of  $P$  for subsequent levels within the multigrid hierarchy. These functions are described in detail within Sections 3.1.1–3.1.3.

**3.1.1. Function `buildTopLevel()`.** This function generates the top-level matrix,  $A_0$ , for a target multigrid hierarchy and stores it within the SG format. In its current state, this function supports the generation of 2-dimensional constant-coefficient problems featuring either a 5-point or a 9-point stencil. Note however that the semi-coarsening process produces a coarse matrix,  $A_1$ , which features a 9-point stencil regardless of the size of the stencil used for  $A_0$ .

The core functionality of `buildTopLevel()` is based upon supporting functions<sup>2</sup> used within Galeri’s `BuildProblem()` [2]. A custom implementation of `BuildProblem()` was created to minimize intermediate storage requirements when storing  $A_0$  in the SG format. This decision also allows for easier integration of Kokkos in the event that this function is parallelized as future work.

When building the top-level matrix, problem construction is based upon a map that establishes the adjacency list used to generate the sparse matrix corresponding to  $A_0$ . Following suit with Galeri, this map features 0-based indexing, which begins in the bottom-left corner and proceeds from left-to-right then bottom-to-top as depicted in Figure 3.2. With this in mind, the algorithm loops over each point within the map and populates the corresponding row of  $A_0$  with both the value and column index for itself and adjacent points. In

<sup>2</sup>Supporting functions that `buildTopLevel()` are based upon can be found in `\Trilinos\packages\galeri\src-xpetra\GaleriXpetraMatrixTypes.hpp` [11]

its current state, this serial algorithm leverages a single `for` loop featuring a number of conditionals to check existence of each possible adjacent neighbor when populating  $A_0$ .

20	21	22	23	24
15	16	17	18	19
10	11	12	13	14
5	6	7	8	9
0	1	2	3	4

Fig. 3.2: Map used to generate the sparse matrix corresponding to  $A_0$  for a problem featuring 5 points in each dimension.

**3.1.2. Function `buildCollapsed()`.** This function semi-coarsens  $A_0$ , generating intermediate tridiagonal matrices that are needed to construct the grid of prolongation column vectors,  $P$ . Specifically, this function populates arrays that correspond to semi-coarsened matrices of  $A_0$  for each dimension. These arrays are specially formatted to be passed as input parameters to Lapack’s `DGBSV()` function [4]. This function is used to solve systems of banded linear equations within `buildP()`.

The core functionality of `buildCollapsed()` is based upon the semi-coarsening algorithm described within [20]. In its current state, this function supports the collapsing of arbitrary structured grids corresponding to 2-dimensional problems featuring either a 5-point or a 9-point stencil. To accomplish this, the algorithm loops over each point within the map depicted in Figure 3.2 and collapses the PDE along both the x- and y-dimensions. As with `buildTopLevel()`, the current serial algorithm leverages a single `for` loop featuring a number of conditionals to check existence of adjacent neighbors when performing the collapsing procedure.

When collapsing a given point, it is important to note that there is special handling for points on the fine grid that correspond to points on the coarse grid. Such points are injected into the coarse grid rather than being semi-coarsened via the collapsing procedure. This injection is facilitated via parameters that define the distance between coarse points for each dimension. These distances enable the identification of coarse grid points on the fine grid. Note that, the bottom-left corner of the coarse grid is arbitrarily defined as being offset from the bottom-left corner of the fine grid by 1 in each dimension. An example of the relationship between fine grid points and coarse grid points can be seen in Figure 3.3.

**3.1.3. Function `buildP()`.** This function generates the matrix of prolongation column vectors,  $P$ , that is used to generate a coarse grid via the Galerkin product. To accomplish this, a variety of intermediate steps are needed to move from the tridiagonal matrices generated by `buildCollapsed()` to the final form of  $P$ . At a high-level, this involves solving a system of banded linear equations for each coarse point, extracting nonzero entries from these solutions, computing their tensor products, and assembling  $P$ . An example of this process can be seen in Figure 3.4.

In its current state, this function supports the generation of  $P$  for 2-dimensional problems that generate basis column vectors conforming to an assumed structure. To accomplish this, the current serial algorithm loops over each coarse point within the map depicted in Figure 3.3 and generates the corresponding prolongation column vectors.

-	-	-	-	-
-	16	-	-	19
-	-	-	-	-
-	6	-	-	9
-	-	-	-	-

Fig. 3.3: Map indicating fine grid points from  $A_0$  in Figure 3.2 that are injected into the coarse grid,  $A_1$ , for a problem featuring 5 points in each dimension with coarse points spaced 3 and 2 points apart in the x- and y-dimensions, respectively. Points labeled with a " - " are not injected into the coarse grid and are, instead, semi-coarsened via the collapsing operation.

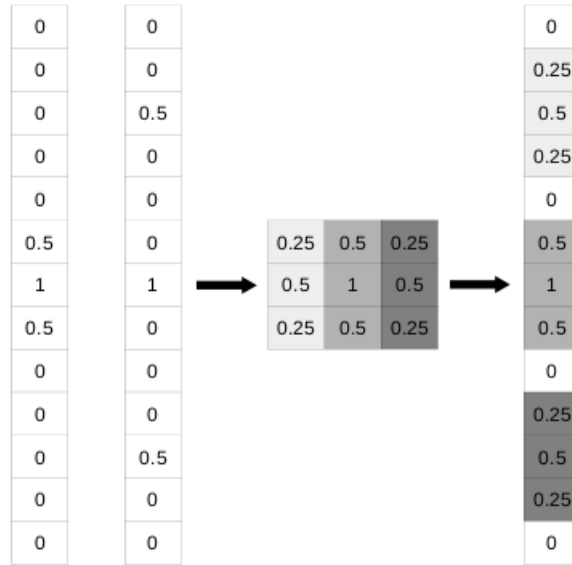


Fig. 3.4: For each coarse point in  $A_1$ , a basis column vector is generated from the horizontally and vertically collapsed matrices (on the left). Nonzero entries within the basis column vectors are extracted and their outer product is computed (in the middle). Columns from the result of the outer product are then extracted and used to assemble a column of  $P$  (on the right).

**4. Smoothers.** The key to multigrid lies in the complementary nature of the smoothing and correction of error between the current solution and actual solution. The smoother's role is to quickly remove high energy error. However once the error is damped, the convergence of the smoother stagnates and correction comes into play by solving  $A_c e_c = R * r$  (where the fine level residual  $r = b - Ax$  is restricted to the coarse level designated by  $A_c$ ) and updating the fine approximate solution  $x$  by  $x = x + P * e_c$  (where the coarse error approximation  $e_c$  is interpolated to the fine level). Updating  $x$  may reintroduce high energy

error components, so smoothing is applied again.

Three iterative methods commonly used as smoothers in multigrid are as follows. For more information see [17].

- Jacobi
- Chebyshev
- Gauss Seidel.

In our work we implemented a Kokkos version of each smoother that is designed for matrices in the SG format. Pseudocode for each can be found in the appendix in Algorithms 7.1, 7.2, and 7.3, respectively. The implementations of these algorithms all utilize Kokkos parallel dispatch with the exception of a purposely serial Gauss Seidel. Jacobi utilizes `parallel_for` over the outermost loop to run over the rows of the matrix in parallel. Chebyshev utilizes various parallel kernels to accomplish the vector additions, matrix-vector multiplications, and the scale by diagonal. Gauss Seidel requires some modifications to run in parallel.

To utilize parallelization in the Gauss Seidel smoother, a staged Gauss Seidel was created that allows for specific chunks of rows to be run at the same time. The user can specify the chunks of rows via two arrays that act as a graph and map in a way similar to the CRS format's graph and map. There are two provided ways of splitting rows into these chunks, which can be found in `Color_Helper.hpp`<sup>3</sup> and `Level_Helper.hpp`. These methods respectively assign colors and schedule levels to the matrix. Because Staged Gauss Seidel aims to preserve symmetry, the forward and back sweep run the stages in reverse order of one another. Thus if chunks are not chosen so that no row in a chunk depends on another row in the chunk, symmetry may be lost.

**5. Results.** We implemented a driver program that mimics a simplified version of the program `MueLu_Driver` found in the Trilinos `MueLu` AMG library [5]. Our driver accepts the grid dimensions for a two dimensional grid and generates the corresponding 2D-Laplacian matrix. It then utilizes the coarse grid creation functions discussed in Section 3 to create a single coarse level for the system of equations. Finally it uses a V-cycle multigrid implementation as a solver. The user can define which smoothers are to be used within the V-cycle by passing compile flags found in the CMake files. The program also passes in hard coded values to specify how many V-cycle iterations should be performed along with the number of pre-smoothing and post-smoothing iterations, as well as how many iterations of the smoother should be used at the coarsest level to consider the coarse system solved. In the future these may be able to be passed in at runtime via command line arguments.

We compared runtimes of our driver with those produced by the `MueLu_Driver`. The tests were run and results gathered using the `pbatch` queue on a local cluster Shannon [8]. We used a Shannon compute node with a dual-socket 8-core Sandy Bridge Xeon E5-2670 [3]. All results were collected using 1, 2, 4, and 8 threads.

In the first set of experiments, we compare strong scalability of SGAMG to that of `MueLu`. In strong scaling, the global problem size is fixed, and the number of threads is increased. Figure 5.1 shows results for a two-level V-cycle. One pre-smoothing and post-smoothing Chebyshev sweep is used on the fine level, and 30 Chebyshev sweeps are used as a solver on the coarse level in practice we would use a direct solve on the coarsest level. Figure 5.2 shows results for a two-level V-cycle. One pre-smoothing and post-smoothing Jacobi sweep is used on the fine level, and 30 Jacobi sweeps are used as a solver on the coarse level in practice we would use a direct solve on the coarsest level. As can be seen in Figures 5.1 and 5.2 our driver scales similarly to `MueLu` but is often outperformed. The SGAMG driver outperforms the `MueLu` driver when the problem consists of only 10,000 rows. For

<sup>3</sup>Using `Color_Helper.hpp` requires enabling Tpetra Experimental Kernels

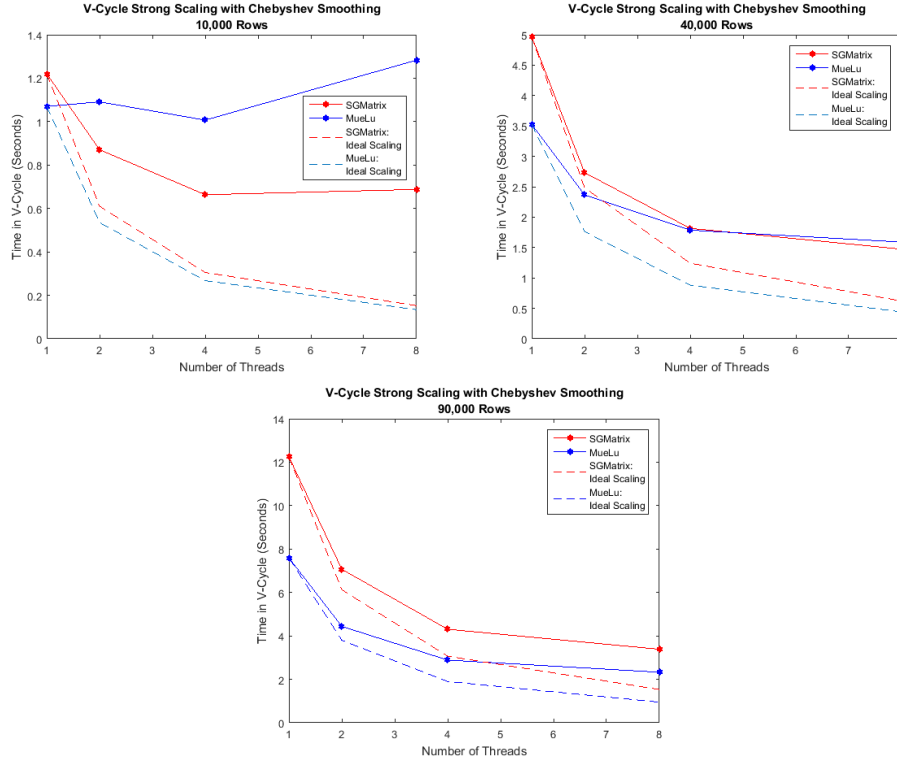


Fig. 5.1: SGAMG vs. MueLu V-Cycle timings across problems with 10000, 40000, and 90000 rows using Chebyshev smoothing. The V-Cycle timing for the SGAMG driver is captured in red while MueLu is captured in blue. The dashed lines represent the ideal scaling for each color respectively.

the 40,000 rows per thread case, however, the MueLu driver is faster than SGAMG for 1 and 2 threads. For 4 and 8 threads, the two codes have identical performance. For 90,000 rows per thread, the MueLu driver is faster for all thread counts. To improve its performance, SGAMG could utilize hierarchical parallelism<sup>4</sup> within the Kokkos parallel kernels instead of the top level only parallelism that is currently used. Specifically with Jacobi smoothing, we note that when the work is divided to 10,000 rows per thread, SGAMG tends to outperform MueLu, however, the opposite occurs when there are more than 10,000 rows per thread. With Chebyshev smoothing, we note that though SGAMG never outperforms MueLu when the work is divided to 10,000 rows per thread, it scales similarly.

In Figure 5.3 we compare weak scalability of SGAMG of that of MueLu. Weak scaling adjusts the problem size based on the number of threads used so that the amount of work per thread stays the same. For this experiment we adjust the problem size in each run so that there are approximately 10,000 rows of work per thread. For both smoothing options the SGAMG driver performs competitively with a similar growth pattern as MueLu until eight threads are used. Ideally the time required by either driver should be flat across the different number of threads. Instead we see the each driver's runtime increasing with thread

<sup>4</sup>For more information please see the Kokkos Programming Guide [18]

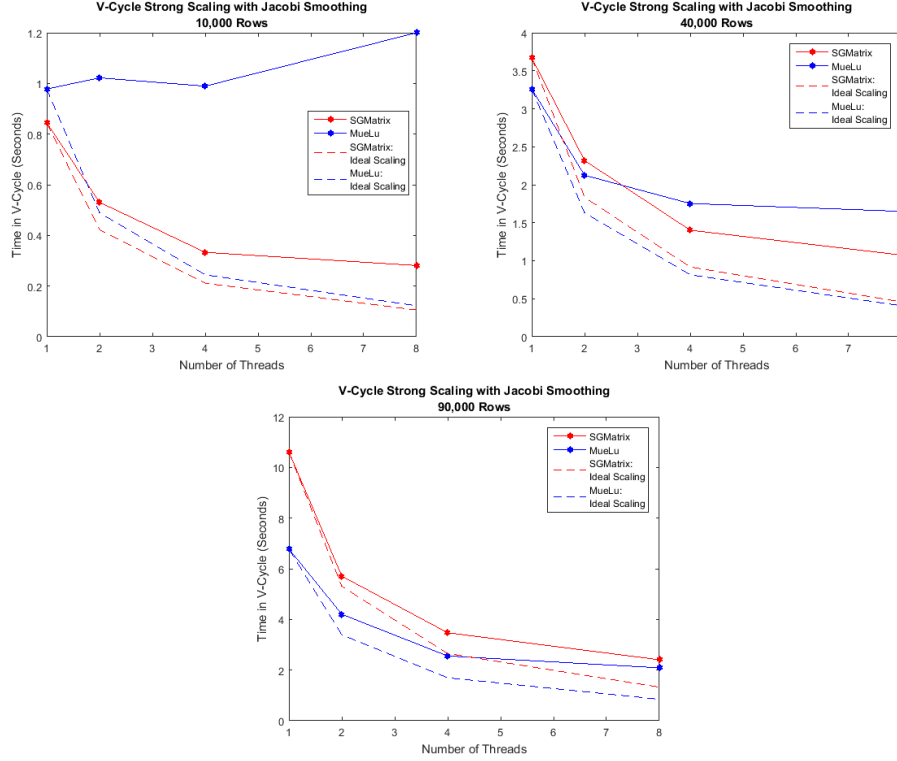


Fig. 5.2: SGAMG vs. MueLu V-Cycle timings across problems with 10000, 40000, and 90000 rows using Jacobi smoothing. The V-Cycle timing for the SGAMG driver is captured in red while MueLu is captured in blue. The dashed lines represent the ideal scaling for each color respectively.

count; at 8 threads the SGAMG driver's runtime is larger than MueLu's. Noting that SGAMG behaves similarly to MueLu, we conjecture that the spike in V-cycle time could be caused by thread initialization.

With SGAMG we had hoped to see improved performance compared to MueLu by taking advantage of problem structure, e.g., by removing extra memory accesses. Instead we are seeing that avoiding those memory accesses doesn't have as much of an impact on performance as anticipated. Further profiling of the code is necessary to pinpoint bottlenecks in performance, but we currently suspect that conditionals to handle boundary conditions in the matrix-vector product kernel may be partly to blame.

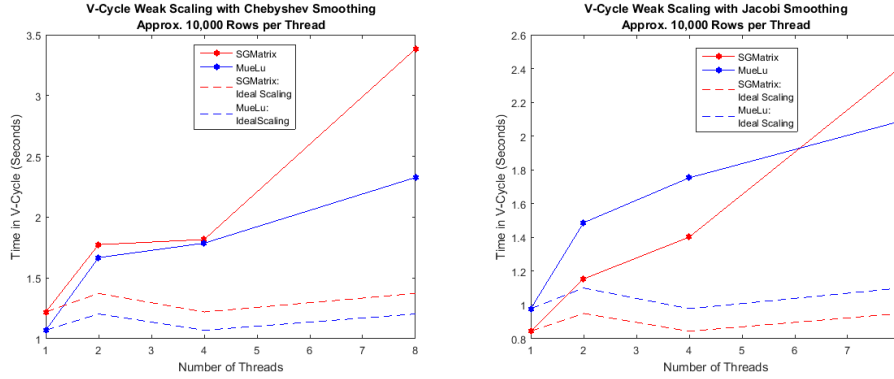


Fig. 5.3: SGAMG vs. MueLu V-Cycle timings utilizing 1, 2, 4, and 8 threads with approximately 10000 rows per threadcount. The V-Cycle timing for the SGAMG driver is captured in red while MueLu is captured in blue. The dashed lines represent the ideal scaling for each color respectively.

**6. Conclusions.** The multigrid approach is an efficient way to solve many of the PDE's that occur in scientific computing. Since these grids have a structure, we devised a data structure utilizing Kokkos that takes advantage of this structure to try and improve performance in the multigrid cycle. We implemented a black box approach that takes only the top level problem and generates a coarse representation of the problem as well. Finally we wrote a driver program that generates the problem and its coarser counterpart and uses a V-cycle to attempt to solve the problem.

We compared our driver to the one found in MueLu and noted that with smaller loads per thread our V-cycle performs as desired but increasing the amount of work per thread gives the MueLu driver the edge. Further analysis would show which parts of our V-cycle are bottlenecks, whether it be the sparse matrix vector multiplication used for grid transfer or the actual smoother implementations themselves. Once bottlenecks have been identified some possible optimizations are implementing Kokkos hierarchical parallelism instead of top level parallelism and removing the need for flag checking while iterating across rows of the matrix stored in the SGMatrix format.

**7. Future Work.** There is still plenty of work to be done on this project on top of identifying and fixing performance bottlenecks. Current builds only support single node runs but future support for MPI [6] is ideal. Similarly, we can currently only create problems over a two dimensional mesh with a fixed coefficient stencil. Extending support for both a three dimensional mesh and a variable coefficient stencil are necessary to increase the scope of problems we can solve. SGAMG is also currently restricted to one coarse level. Being able to create and use multiple coarse levels will help improve convergence rates and allow for larger problems to be solved quickly.

**Appendix.** This appendix contains pseudocode for various algorithms mentioned within the paper. Algorithm 7.1 shows pseudocode for the Jacobi algorithm over an SGMatrix:

Algorithm 7.2 shows pseudocode for the Chebyshev algorithm over an SGMatrix:

Algorithm 7.3 shows pseudocode for a Staged Gauss Seidel algorithm over an SGMatrix:



**Algorithm 7.1** Jacobi over SGMATRIX

---

```

for  $i = 0$  to # of Matrix Rows do
   $sum = 0$ 
  for  $j = 0$  to size of structure stencil do
     $col = A.graph_{ij}$ 
    if  $col \geq 0$  and  $col \neq i$  then
       $sum += x_{col}^n * A.values_{ij}$ 
   $x_i^{n+1} = (b_i - sum) / A.diagonal_i$ 

```

---

**Algorithm 7.2** Chebyshev over SGMATRIX

---

```

 $\lambda_{min} = \lambda_{max} / Ratio$ 
 $d = (\lambda_{max} + \lambda_{min}) / 2$ 
 $c = (\lambda_{max} - \lambda_{min}) / 2$ 
 $r = b - Ax$ 
for  $i = 0$  to Iterations do
  for  $j = 0$  to # of Matrix Rows do
     $z_j = r_j / diagonal_j$ 
  if  $i = 0$  then
     $p = z$ 
     $\alpha = 2 / d$ 
  else
     $\beta = \alpha * (c/2)^2$ 
     $\alpha = 1 / (d - \beta)$ 
     $p = z + \beta p$ 
   $x = x + \alpha p$ 
   $r = b - Ax$ 

```

---

**Algorithm 7.3** Staged Gauss Seidel over SGMATRIX

---

```

for  $s = 0$  to numStages do
   $begin = stageMap_s$ 
   $end = stageMap_{s+1}$ 
  for  $i = begin$  to  $end$  do
     $row = stageInd_i$ 
     $sum = b_{row}$ 
    for  $j = 0$  to size of structure stencil do
       $col = A.graph_{ij}$ 
      if  $col \geq 0$  and  $col \neq i$  then
         $sum -= A.values_{ij} * x_{col}$ 
     $x_{row} = sum / diagonal_{row}$ 

```

---

## REFERENCES

- [1] CUDA Toolkit Documentation. <http://docs.nvidia.com/cuda/#axzz4K3M4eZ0d>.
- [2] Galeri - The Trilinos Project. <https://trilinos.org/packages/galeri/>.
- [3] Intel Xeon Processor E5-2670 Specifications. [http://ark.intel.com/products/64595/Intel-Xeon-Processor-E5-2670-20M-Cache-2\\_60-GHz-8\\_00-GTs-Intel-QPI](http://ark.intel.com/products/64595/Intel-Xeon-Processor-E5-2670-20M-Cache-2_60-GHz-8_00-GTs-Intel-QPI).
- [4] Lapack DGBSV(). <http://www.math.utah.edu/software/lapack/lapack-d/dgbsv.html>.
- [5] MueLu - The Trilinos Project. <https://trilinos.org/packages/muelu/>.

- [6] Open MPI: Open Source High Performance Computing. <https://www.open-mpi.org/>.
- [7] The OpenMP API Specification for Parallel Programming. <http://openmp.org/wp/>.
- [8] Sandia National Laboratories: Advanced Simulation and Computing: Advanced Systems Technology Test Beds. [http://www.sandia.gov/asc/computational\\_systems/HAAPS.html](http://www.sandia.gov/asc/computational_systems/HAAPS.html).
- [9] Tpetra - The Trilinos Project. <https://trilinos.org/packages/tpetra/>.
- [10] The Trilinos Project. <https://trilinos.org/>.
- [11] Xpetra - The Trilinos Project. <https://trilinos.org/packages/xpetra/>.
- [12] B. BARNEY, POSIX Threads Programming. <https://computing.llnl.gov/tutorials/pthreads/>.
- [13] W. L. BRIGGS, V. E. HENSON, AND S. F. MCCORMICK, *A Multigrid Tutorial*, SIAM, 2nd ed., 2000.
- [14] J. E. DENDY, JR., Black Box Multigrid, *Journal of Computational Physics*, 48 (1982), pp. 366 – 386.
- [15] H. C. EDWARDS AND C. TROTT, Kokkos. Available online: <https://github.com/kokkos/kokkos>, 2016.
- [16] J. J. HU, A. PROKOPENKO, C. M. SIEFERT, R. S. TUMINARO, AND T. A. WIESNER, MueLu Multigrid Framework. <http://trilinos.org/packages/muelu>, 2014.
- [17] Y. SAAD, *Iterative Methods for Sparse Linear Systems*, Society for Industrial and Applied Mathematics, 2nd ed., 2003.
- [18] C. R. TROTT, M. HOEMMEN, S. D. HAMMOND, AND H. C. EDWARDS, Kokkos: The Programming Guide. <https://github.com/kokkos/kokkos>, 2015.
- [19] U. TROTTEBERG, C. W. OOSTERLEE, AND A. SCHÜLLER, *Multigrid*, Academic Press, 2000.
- [20] R. TUMINARO, M. PEREGO, I. TEZAUER, A. SALINGER, AND S. PRICE, A Matrix Dependent/Algebraic Multigrid Approach for Extruded Meshes with Applications to Ice Sheet Modeling. Submitted.

## DISTRIBUTED AND TASK-PARALLEL APPROACH TO LINEAR ALGEBRA BASED GRAPH ANALYTICS USING HPX

DANIEL C. BOURGEOIS<sup>¶</sup> AND MICHAEL M. WOLF<sup>||</sup>

**Abstract.** It is challenging to obtain scalable HPC performance on real applications, especially for data science applications with irregular memory access and computation patterns. To drive co-design efforts in architecture, system and application design, we are developing miniapps representative of data science workloads. This paper discusses an implementation of one such miniapp, miniTri, which makes use of both distributed workloads and task parallelism using the parallel runtime system, HPX. One specific challenge addressed in this paper is minimizing communication across nodes while still maintaining high levels of asynchrony.

### 1. Introduction.

**1.1. miniTri Data Analytics Miniapp.** Application proxies or miniapps are important driving forces in the architecture-system-application co-design efforts that attempt to ensure good performance for real applications on modern processors and supercomputing systems. miniTri is a data analytics miniapp that is part of the Mantevo [4] suite of miniapps.

As per the purpose of miniapps, miniTri is a compromise between the simplicity of kernel benchmarks and the complexity of real applications. HPC data analytics benchmarking applications have been represented over the years by SSICA-2 [2] and Graph 500 [6]. These similar benchmarks emphasize graph generation and search. The miniTri miniapp is not based on neighbor set expansion, and offers different challenges than traditional graph search [9].

Instead, miniTri is a triangle enumeration or listing based algorithm (not triangle counting). miniTri applies several operations to find an upper bound on the largest clique in the input graph. The upper bound on the largest clique is a useful property to have as it helps describe what dense subgraphs might look like. In the case of a distributed implementation of miniTri, the input graph would generally be a large network. Even if the input graph does not fit into main memory on one compute node, the calculation should still be feasible. miniTri uses the triangle enumeration kernel within the miniapp to tabulate k-counts over all triangles in a given graph. A k-count is defined for a triangle in a graph. Given triangle  $t$ , let  $t_v$  ( $t_e$ ) be the number of triangles incident on vertex  $v$  (resp. edge  $e$ ).  $t_v$  and  $t_e$  are also referred to as triangle degrees, computed by enumerating all triangles in the graph. A k-count for a given triangle  $t$  is defined by

$$\operatorname{argmax}_k \{ (\min_{v \in t} t_v \geq \binom{k-1}{2}) \wedge (\min_{e \in t} t_e \geq k-2) \}. \quad (1.1)$$

miniTri can be formulated in terms of linear algebra, where multiplication of two matrices is overloaded to enumerate triangles. This formulation is shown in Figure 1.1, where  $A$  is the adjacency matrix of the graph,  $B$  is the incidence matrix of the graph and  $C$  is a matrix where each non-empty element is the three vertices of a triangle. Specifically, the multiplication is overloaded such that  $C(i, j) = (i, x, y)$  iff  $A(i, x) = A(i, y) = 1$  and  $B(x, j) = B(y, j) = 1$  (by construction,  $B(*, j) = 0$  for other elements in this column). See [9] for a worked out example.

<sup>¶</sup>Louisiana State University, dbour27@lsu.edu

<sup>||</sup>Sandia National Laboratories, mmwolf@sandia.gov

```

1: procedure MINITRI(A, B)
2:   kCnts <- 0
3:   C = A*B                                |> Enumerate triangles
4:   triVertDegrees = C*1                    |> Calculate triangle vertex degrees
5:   triEdgeDegrees = C'*1                   |> Calculate triangle edge degrees
6:   For all {v1, v2, v3} in C(i, j) where C(i, j) != Null
7:     if (v1 > v2 and v1 > v3)
8:       k1 = argmaxk{min v in t st. triVertDegrees(v) >= (k-1, 2) }
9:       k2 = argmaxk{min e in t st. triEdgeDegrees(e) >= k-1}
10:      k = min(k1, k2)
11:      kCnts(k) = kCnts(k) + 1
12:     endif
13:   endfor
14: end procedure

```

Fig. 1.1: General miniTri algorithm

The linear algebra variant of miniTri contains implementations in serial, OpenMP, MPI, Kokkos with Qthreads and HPX with shared-memory. This paper is concerned with an additional HPX with distributed-memory implementation.

**1.2. Task Parallelism and HPX.** Task parallelism is a form of parallelization that focuses on assigning tasks across available resources, where a task can be defined as a unit of execution that is done sequentially. Task parallelism can be contrasted with data parallelism, which focuses on distributing data across parallel resources. By decomposing a program into logical tasks rather than the number of processors, a task parallel program can scale transparently based on input size and available processors without reimplementing. Instead of placing the burden of scheduling parallel execution on the programmer, the runtime system handles scheduling, making one implementation viable for a wider range of computer architectures. With the recent changes in system architecture with many more lighter-weight compute cores and extended multi-level memory hierarchies, task-parallel programs provide a means to be portable across multiple architectures while still maintaining a high utilization of resources.

To make the best use of task parallelism, over decomposition of parallel work must occur. The purpose of over decomposition is to give the runtime system enough tasks to prevent starvation of resources. In miniTri, each linear algebra kernel is over decomposed into many light-weight tasks. For sufficiently large graphs the number of tasks is much greater than the number of computational cores that are being targeted. Another method that aides over decomposition in miniTri is the removal of all global barriers not intrinsic to the computation itself. Removing global barriers can potentially expose more parallelism than programs with lock-step synchronization.

Numerous task-based languages and runtime systems have been developed (e.g., Cilk [3], OpenMP [7], Qthreads [8] and High Performance ParalleX (HPX) [5]. This paper focuses on HPX, a “general purpose C++ runtime system for parallel and distributed applications of any scale” [5]. Specifically, HPX supports fine-grained parallelism through the use of lightweight threads as well as a global system-wide address space. One of the benefits of using HPX as the runtime system is that task parallelism can be expressed on both shared-memory resources and distributed-memory resources. Often, the distinction between shared-memory resources and distributed memory resources does not need to be made. Using task

parallelism, if a light-weight thread is waiting for non-local data, the runtime system can suspend that thread until the data becomes available and schedule available work. As inter-node communication can be a large bottleneck in distributed-memory applications where high latencies from communication are unavoidable, hiding that latency with available work is ever more important.

**1.3. HPX Task-Parallel Functionality.** In this miniTri implementation, task-parallel operations in the HPX interface are controlled through futures. Futures are “proxies for results that are not yet known” [5]. They usually represent the result of a task that will not be available until the task is finished. The canonical function for launching a task in HPX is `hpx::async`. `hpx::async` accepts a function and arguments to that function, launches the function on its own lightweight thread and returns the result asynchronously inside of a future. To launch a task without tracking the result the task, `hpx::apply` can be used.

Two important operations on futures are waiting for a future to be ready and getting the value of a future. Waiting for a future suspends the thread of execution until the values inside of the future are ready and getting a future first waits for the future to be ready and then returns the result contained inside of the future. Waiting for a non-ready future in HPX suspends the thread so that other computations can be done on the compute core. As an example in HPX, querying a distributed data structure for a value yields a future immediately. Whether or not that future is ready immediately depends on whether or not the data is stored locally.

The task parallel runtime system must be aware of the dependencies of the computation so that tasks can be scheduled correctly and parallelism can be exploited. In HPX, this is done by attaching continuations to futures. A continuation is the stopping and restarting of tasks across long-latency events. To say that a continuation is attached to a future means that once the future is ready, another computation will occur. One such way to attach a continuation is through `hpx::future`’s `then` function. `then` is passed a function that accepts the future that `then` is being called on. Once the future is ready, the passed function is called, the result of which is returned asynchronously (inside of a future) by `then`.

`then` is an example of a dataflow object. Dataflow objects are mechanisms that manage continuations among futures. Given a set of futures, the dataflow object “launches a predefined function passing on the values encapsulated by the input futures” [5]. Two other dataflow objects that this implementation takes advantage of are `hpx::when_each` and `hpx::when_any`.

**1.4. HPX Distributed Functionality.** HPX supports task parallelism across nodes on a cluster. In HPX terms, a node on a cluster is called a *locality*. Functions that can be called across localities are called *actions*. An action can be invoked using the `hpx::async` function which returns a future. An action can also be invoked using “fire-and-forget” semantics with `hpx::apply`.

Structures and classes that are acknowledged by HPX’s global address space are called *components*, and contain a global id. Components can have member functions that are actions. To invoke an action on a component, call `hpx::async` with the global id of the component. Components can also be migrated across localities or even be linked with other components on other localities.

One component defined in HPX and used in this miniTri implementation is the distributed data structure `hpx::partitioned_vector`. On each locality, `partitioned_vector` maintains a contiguous chunk of memory. The `partitioned_vector` distributes an even number of elements to each locality, in a block partitioning scheme. For multiple components to refer to the same underlying `partitioned_vector`, the `partitioned_vector` must be registered. When querying values from a `partitioned_vector`,

the component will determine where the data is stored and if the data is not local, invoke the corresponding action on that data structure and return a future. On the other hand, if the value is located locally, then the future that is returned is ready.

**2. Distributed Task-Based miniTri.** This section describes in detail the distributed task-based miniTri implementation using HPX. While the same linear-algebra based formulation of the algorithm described in Figure 1.1 is used, a number of complications are considered. First, global barriers between kernels should be removed. The implementation should also take into consideration graphs that are too big to fit into main memory on one locality. Lastly, how should decomposition of tasks occur in a distributed environment?

With regards to communicating and receiving data, the design of this distributed HPX implementation is based off of two assumptions. First, communication across localities is expensive. Second, requesting more data in a request is cheaper per unit of data than requesting less data per request. While this implementation could have decomposed tasks in a memory-hierarchy ignorant way and let HPX manage the added latencies, the amount of added work in the form of communication from doing so would have been too great. Instead, this implementation explicitly managed communication.

**2.1. AsyncView.** The miniTri implementation uses two data structures, a sparse matrix `CSRMat` and a distributed vector, `Vector`. Both data structures contain an underlying `partitioned_vector` and an `AsyncView`. To describe the data structures, two terms are used in this paper, *partition* and *block*. A *partition* is the section of a distributed data structure on a locality. And a *block* refers to a section of a partition. See Figure 2.1 for an example applied to a sparse matrix. The `hpx::partitioned_vector` manages communication of elements and the `AsyncView` manages on-node parallelism.

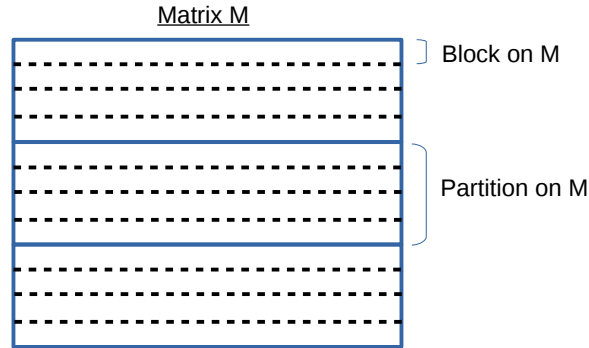


Fig. 2.1: Partitioning of Matrix M

To facilitate on-node parallelism inside of distributed data structures, and to reduce the number of nested for loops, `AsyncView` was created. An `AsyncView` attaches continuations to blocks on distributed data structures. Given a start index and an end index, an `AsyncView` creates  $n$  blocks, with an even number of elements on each block where  $n$  is the number of cores that HPX finds. A subsequent implementation relaxes this restriction and allows multiple blocks per core by specifying the number of rows belonging to each block. Each block inside of an `AsyncView` contains a future. The `AsyncView::for_loop` member function takes as an argument a function that is called once for each element. When `AsyncView::for_loop` is called, a continuation to each block is attached to itself, using

`hpx::future::then` member function. As an example, in Figure 2.2 the following guarantees are made: `f1` is guaranteed to be called before `f2`, each block is allowed to operate in parallel to the other blocks and any block is permitted to finish executing `f2` for each of its elements before other blocks have finished `f1`.

```

AsyncView av(0, 100);
std::vector<int> v(100);
// Given index of v, assign idx to v[idx]
auto f1 = [&](int const& idx){ v[idx] = idx; };
// Given index of v, double v[idx]
auto f2 = [&](int const& idx){ v[idx] *= 2; };
av.for_loop(f1);
av.for_loop(f2);
av.wait_all();
// v now contains 0, 2, 4, ..., 198

```

Fig. 2.2: AsyncView usage. `f1` is guaranteed to be executed before `f2` for each index from 0 through 99.

**2.2. Distributed Memory Based Triangle Enumeration.** After obtaining the adjacency matrix `A` and the incidence matrix `B`, the first kernel in `miniTri` creates `C` using the overloaded multiplication operator in the linear algebra formulation. In the distributed HPX implementation, `C`, `A` and `B` are all distributed into partitions of sparse rows. Each row contains the indexes of columns that are non-zero and the values in those row column pairs. The computation of one row of `C` requires the corresponding row of `A` and the rows of `B` corresponding to the non-zero columns of `A`, as shown in Figure 2.3. In this case, the dependency on `B` isn't known until the `A` matrix is set up.

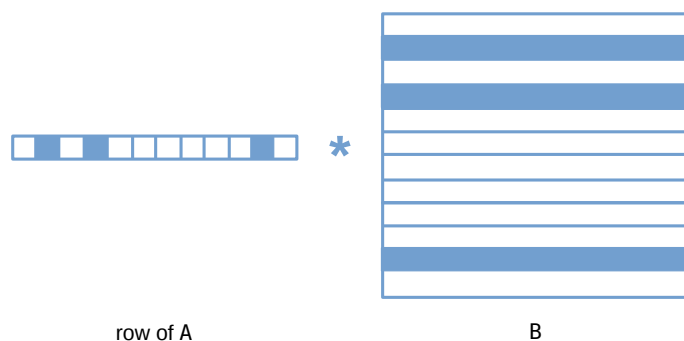


Fig. 2.3: The rows of `B` that are highlighted are required to calculate the row of `C` corresponding to the row of `A` shown. The highlighted elements of the row of `A` shown are the non-zero values of that row.

Following the assumptions on communication outlined in above, as few calls that requests data from `B` as possible should be made. For this reason, all of the non-zero columns

in the local partition of A are calculated. Then the data is requested from B by the application. All of the data from locality  $l$  is lumped together and communicated in one request. HPX then manages the request inside of a future. The rows of B requested are then stored in a container containing one future per locality requested. This way, calculations that don't depend on all localities can start computation before data from all localities are received. As each row of C is independent of the other rows of C, the calculation can proceed in parallel. The parallelization is done by grouping each block into a task.

**2.3. Triangle Vertex Degrees.** In the linear algebra formulation of miniTri, calculating the triangle degrees of each vertex is a matrix-vector multiplication, `triVertDegrees = C*1` in Figure 1.1. Each element in `triVertDegrees` should correspond to the number of non-zeros in the corresponding row of C. For the 1D distribution used, `triVertDegrees` is a `Vector`. A `Vector` wraps a `hpx::partitioned_vector` and a `AsyncView`. `triVertDegrees` is partitioned the same way as C, so no communication is required.

```
triVertDegrees.for_loop(matmatAview,
    [this, &triVertDegrees](std::size_t const& idx)
    {
        triVertDegrees.set_value(idx, get_row_sync(idx).nnz());
    });
```

Fig. 2.4: Calculate triangle vertex degrees, code from the HPX distributed-memory version of miniTri.

The actual code is shown in Figure 2.4. `matmatAview` is the `AsyncView` that is owned by the C matrix containing the futures representing tasks of triangle enumeration on each block of C. The `Vector::for_loop` member function attaches the continuations from the `AsyncView` passed as an argument to its own `AsyncView`. This is done so that each block in `triVertDegrees` will be calculated only after the corresponding block in C has finished its computation.

**2.4. Triangle Edge Degrees.** Similar to calculating the triangle vertex degrees, calculating the triangle edge degrees is a matrix-vector multiplication, except C is transposed. And unlike calculating triangle vertex degrees, calculating triangle edge degrees may require communication across localities. Consider the `Vector` containing triangle vertex edges, `triEdgeDegrees` from figure 1.1. Each element  $e$  in `triEdgeDegrees`, belonging to index `idx`, is the number of non-zero elements in C belonging to column `idx`.

Creating a  $1 \times n$  matrix is on the same order of magnitude as A, B or C, which could possibly not fit into the main memory of a single locality. The current implementation avoids creating a  $1 \times n$  matrix. Instead, each triangle edge degree task (one for each locality) is responsible for two subtasks. The first subtask is taking the local partition of C and reducing that into a map from the column index to the number of non-zero indices of C for that column and partition. Then, that map is communicated out to all localities, only sending to each locality the part of the map required by that locality. The second subtask is to receive the maps from all the localities and calculate its own partition of `triEdgeDegrees`. It should be stressed that communication could be done in HPX without explicitly recognizing the boundaries where the data is stored. Even though asynchrony helps to hide latencies, this implementation takes advantage of the knowledge of data location.



**2.5. K-Counts.** The last step in the algorithm is to tabulate k-counts for all triangles. For a given triangle  $t$ , the k-count, defined in 1.1, depends on the three triangle vertex degrees and the three edge degrees of  $t$ . Once the k-counts are tabulated on each block  $C$  for each of the triangles (provided that the first index of the triangle is the largest, to avoid duplicate triangles), a local reduce is performed to calculate the k-counts for that partition. Finally, to get the final output of miniTri, a global reduce operation is performed for the k-counts on each partition.

Figure 2.5 shows the dependencies for calculating k-counts on a block of  $C$ . Once triangle enumeration on a block of  $C$  is complete, the vertices of the triangles on that block are collected. Those vertices correspond to indices in `triVertDegrees`. To calculate the indices in `triEdgeDegrees` requires a mapping from a pair of vertices to an edge. This mapping comes from a by-product of creating the incidence matrix. Once the indices are collected, calls to the distributed data structures `triVertDegrees` and `triEdgeDegrees` are made so that the k-counts can be calculated. However, calls to `triVertDegrees` and `triEdgeDegrees` may require communication, which figure 2.5 ignores. If this were the implementation pursued, more communication would be performed than required.

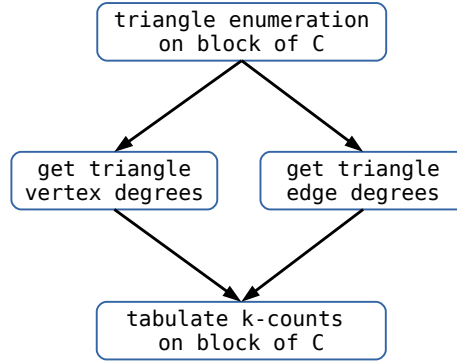


Fig. 2.5: Dependencies for calculating k-counts on a block of  $C$ .

To take into account communication with `triVertDegrees` and `triEdgeDegrees`, triangle degrees can be retrieved by locality, making only one call to each locality per partition (for each of the two `Vector` data structures). This is shown in figure 2.6, where one task per locality is created to retrieve the triangle vertex degrees on that locality. The same occurs for triangle edge degrees. Also, figure 2.6 relies on the entire partition of  $C$ . This is so that all of the vertices and edges can be collected before making calls to `triVertDegrees` and `triEdgeDegrees`. This was the implementation pursued.

For both figure 2.5 and figure 2.6, tabulating k-counts on each block of the partition is permitted to run in parallel. However, figure 2.6 adds extra synchrony by waiting for the entire partition to finish. This is done to reduce communication and stay in accordance with the assumptions mentioned above. Note that the way that the code is structured, calculating each block of k-counts for  $C$  does not necessarily depend on retrieving vertex degrees from all localities. If all the required vertices are on one locality for a block, then only the data for that locality will be waited on.

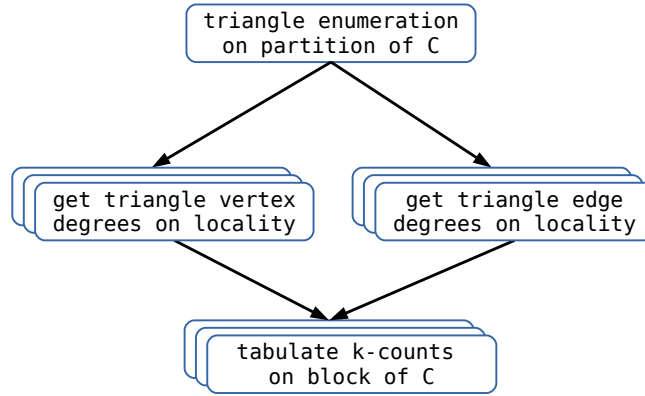


Fig. 2.6: Dependencies for calculating k-counts on a partition of C.

**3. Preliminary Results.** Preliminary results for triangle enumeration and calculating triangle degrees are shown in figure 3.1. The results are for the Oregon-1 (autonomous system dataset) graph (11.5k vertices, 23.4k edges) obtained from the SNAP collection [1]. The implementation was compiled using gcc version 4.9.2, and the experiments were run on NERSC’s Edison, where each node has two sockets populated with a 12-core Intel “Ivy Bridge” processor @2.40GHz, 24 cores per node. The Interconnect on Edison is Cray Aries with Dragonfly topology. The code scales to 16 nodes.

A possible reason that two nodes runs slower than one node is that the added fixed cost of communication is greater than the initial benefits of extra processing power.

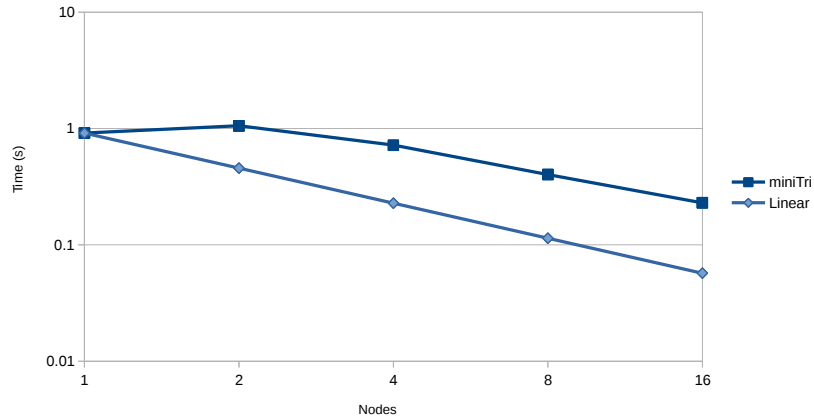


Fig. 3.1: Time for triangle enumeration and triangle degrees calculation on the Oregon-1 (autonomous system data) input graph [1] run on NERSC’s Edison.

**4. Future Work and Conclusions.** This implementation of miniTri focused on using task parallelism in a distributed setting. While implementing miniTri, two key assumptions

were made that shaped the design of the implementation—that communication across localities is expensive and requesting more data in a request is cheaper per unit of data than requesting less data per request. Using these assumptions, this paper focused on the interplay between task-parallelism and using the knowledge of how the partition of data was structured to explicitly control communication.

However, different assumptions could have been made that would have been more true to task parallelism. Asynchrony was sacrificed for bulk synchronous communication in an effort to minimize communication overhead. Namely, this implementation could have assumed that there would always be enough asynchrony to hide the latencies of communication. In the future, it would be interesting to implement a more asynchronous task task parallel version, especially as it would provide a point of reference for future computer architectures and task-parallel systems to strive for.

In addition, future implementations of miniTri should experiment with a 2-D partitioning distribution to overcome the difficulties of a 1-D partitioning distribution. If just a few rows in a sparse matrix were much denser than the other rows memory could be overwhelmed on the locality that must store those rows. A 2-D distribution could prevent this by blocking each row into multiple localities. An added benefit of using HPX, more partitions than localities can be created and the partitions could be sent to other localities to ease memory demands. Another problem with the 1-D distribution is that to calculate any triangle edge degree, the entire C matrix must finish its calculation. Besides creating a barrier that increases synchrony, this also requires storing the entire C matrix, which adds memory demands. A 2-D partitioning scheme could fix this as an edge could be calculated as long as its entire column of partitions are ready.

## REFERENCES

- [1] SNAP: the Stanford Network Analysis Project. <http://snap.stanford.edu>.
- [2] SSCA#2 v2.1 Specification. <http://www.graphanalysis.org/benchmark/>, 2007.
- [3] R. D. BLUMOFÉ, C. F. JOERG, B. C. KUSZMAUL, C. E. LEISERSON, K. H. RANDALL, AND Y. ZHOU, Cilk: An Efficient Multithreaded Runtime System, in Proceedings of the 5th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, New York, NY, USA, 1995, PPOPP '95, ACM Press, pp. 207–216.
- [4] M. A. HEROUX, D. W. DOERFLER, P. S. CROZIER, J. M. WILLENBRING, H. C. EDWARDS, A. WILLIAMS, M. RAJAN, E. R. KEITER, H. K. THORNUST, AND R. W. NUMRICH, Improving Performance via Mini-Applications, Sandia National Laboratories, Tech. Rep. SAND2009-5574, 3 (2009).
- [5] H. KAISER, T. HELLER, B. ADELSTEIN-LELBACH, A. SERIO, AND D. FEY, HPX: A Task Based Programming Model in a Global Address Space, in Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models, PGAS '14, New York, NY, USA, 2014, ACM, pp. 6:1–6:11.
- [6] R. C. MURPHY, K. B. WHEELER, B. W. BARRETT, AND J. A. ANG, Introducing the Graph 500, Cray Users Group (CUG), (2010).
- [7] OPENMP ARCHITECTURE REVIEW BOARD, *OpenMP Application Program Interface*, 2.5 ed., May 2008.
- [8] K. B. WHEELER, R. C. MURPHY, AND D. THAIN, Qthreads: An API for Programming With Millions of Lightweight Threads, in Proceedings of the 22nd International Symposium on Parallel and Distributed Processing, IPDPS/MTAAP, April 2008, pp. 1–8.
- [9] M. M. WOLF, J. W. BERRY, AND D. T. STARK, A Task-Based Linear Algebra Building Blocks Approach for Scalable Graph Analytics, in High Performance Extreme Computing Conference (HPEC), 2015 IEEE, IEEE, 2015, pp. 1–6.

## AUTOMATIC TUNING OF PERFORMANCE PORTABLE SPMV KERNELS IN KOKKOS

SIMON P. GARCIA DE GONZALO\*, SIMON D. HAMMOND†, AND CHRISTIAN R. TROTT‡

**Abstract.** Kokkos [5, 6] is a C++ programming model that offers the ability to write portable code that targets a wide degree of hierarchical parallelism found in current HPC systems. It works by providing abstractions for parallel execution and data layouts that are mapped to different hardware resources. Some parameters, such as the size of thread teams and vector width, are available for the application scientist to tune their code to a particular hardware platform. For many applications, choosing the right parameters is highly dependent on the data input more than the device characteristics. SPMV is a highly irregular computational kernel that can be found in many different applications. Achieving good performance depends heavily on the associated matrix sparsity pattern. In this paper, we propose to extend the current set of Kokkos tools with an autotuner that would iterate over possible candidate teams of threads and vector width, taking advantage of runtime information to choose the optimal parameters for a particular input. This approach would allow an iterative application that calls the same kernel multiple times to continue to progress towards a solution while lifting the burden from the application programmer of knowing the underlying hardware and accounting for variable inputs. We compare an approach using the autotuner against a Fixed approach and show that the optimal choice made by the autotuner is significantly different among three distinct architectures. After 100 iterations a subset of the matrices benefits from a substantial performance improvement, while others are near the break-even point, where the overhead of the tool has been completely hidden.

**1. Introduction.** The current high performance computing (HPC) landscape has become increasingly diverse. HPC nodes now feature many distinct characteristics that can significantly affect performance if not well utilized. These features range from type and number processing units to widely different memory hierarchies and memory types. Within a processing unit there exist a significant variation across vendors concerning support for various numbers threads, numbers of vector units, vector widths, and cache hierarchies. The introduction of accelerator-type devices such as NVIDIA’s GPUs and Intel’s Xeon-Phi [9] has only exacerbated the differences. All of these architectural differences pose challenges for code portability. A significant tuning effort on the part of the application programmers is required to run on each different type of architecture, resulting in the unsustainable solution of maintaining multiple source codes optimized for each type of hardware device.

For these reasons mechanisms to improve code portability and take advantage of the existing highly-parallel resources and their complex shared-memory hierarchies have become numerous. The most well-known solutions for targeting a single source code to both current multi-core processors and accelerator devices such as GPUs are OpenMP [3], OpenACC [18], and OpenCL [15], each with advantages and disadvantages over the others. The problem has now become deciding which solution to use, and determining which solution will provide an application with the greatest performance across all possible hardware devices. Furthermore, it is not sustainable to apply all solutions. This variety regarding possible solutions has made the rise of programming models such as C++AMP [7] and Kokkos [5, 6] possible. These models seek to provide abstractions as part of the language to successfully hide the complexity and variety in the hardware from the application programmer and at the same time provide performance portability across the HPC landscape.

In this paper, we use Kokkos programming model to target different hardware systems. Kokkos allows us to specify some parameters that are usually meant for tuning by the application programmer. We develop a dynamic autotuner tool that would attempt to find the

---

\*University of Illinois at UrbanaChampaign, grcdgnz2@illinois.edu,

†Sandia National Laboratories, sdhammo@sandia.gov,

‡Sandia National Laboratories, crtrott@sandia.gov

most optimal set of these parameters for a particular hardware architecture during runtime. We test our tool using a very common and highly irregular sparse matrix vector (SPMV) computational kernel found in many scientific areas, and we show that the autotuner picks very distinct parameters based on the architecture and the properties of the matrix, highlighting the necessity for this type of runtime tool. Finally, we show that the autotuner incurs a significant overhead and requires a variable number of iterations to improve the performance of the computational kernel.

## 2. Background.

**2.1. Kokkos.** Kokkos programming model provides the application programmer the means to abstract themselves out of the complexities of hardware details, and at the same time maintain portability across hardware devices. Kokkos accomplishes these goals by providing abstractions for the memory space, how is data allocated, and execution space, where parallel patterns are executed. Parallel patterns in Kokkos form part of the language and are defined as a simple set of patterns such as parallel for, reduce, and scan that can be nested within each other to form more complex algorithms. An example of nested patterns is illustrated in lines 19 to 32 of listing 1. Kokkos works on top of an abstract machine model (Figure 2.1), which assumes multiple types and numbers of computing units within a node. Each compute unit also comes with one or more memory spaces optimized for that unit. The execution space represents a group of homogeneous units within the machine model. These units are used to execute a parallel pattern. The programmer can then create multiple instances of execution spaces that target different units. The process of compiling and running computational kernels on these different spaces is abstracted completely by Kokkos. This ability completely removes the need for hardware-specific optimizations to be part of the main code. Kokkos can then provide further tuning through parameters passed to the Kokkos API. We use these parameters for the autotuner proposed in this paper. The memory spaces follow the same logic as the execution spaces. Different instances can represent different types of memory within a node, such as DRAMs, on-package, and non-volatile memory. For accelerators, a memory instance can be used to describe various types of memory such as shared, texture, and global memory.

**2.2. SPMV.** Sparse matrix-vector multiplication, defined as  $y = Ax$  where  $A$  is a sparse input matrix,  $x$  is a dense input vector, and  $y$  is the vector product, is a widely used computational kernel found in many scientific applications. The performance of this operation is of great importance for iterative linear solvers. Within these solvers, SPMV can potentially be called up to many thousands of times until solution convergence. Due to its importance there exist a large variety of implementations and extensive literature dedicated to it. Unlike dense matrix-vector operations, SPMV has to deal with a broad range of irregularity, in particular regarding the sparsity of the matrix. This asymmetry makes it difficult to develop general-purpose, high-performance solutions. Implementations often struggle to make successful use of caches, memory hierarchies, and available computational resources. An extensive summary of approaches by Vuduc [17] is a good overview of this area. Using Kokkos to write a portable high-performing SPMV kernel, we have to deal with the same obstacles. Tuning parameters have to take into account the hardware as well as the matrix characteristics. By making use of Kokkos runtime performance hooks, we develop an autotuning framework that can use runtime timing information to search for the best performing set of parameters for a particular matrix on a particular hardware device. A skeleton implementation of SPMV using Kokkos can be seen in listing 1. The algorithm is defined within a functor from line 19 to 32. It uses a series of nested parallel patterns in lines 19 and 22 to represent the behavior of the SPMV abstractly. This functor is instantiated in

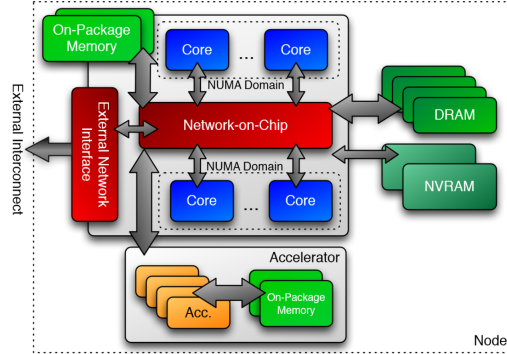


Fig. 2.1: Conceptual model of a high performance computing node [16]

line 5 and used as the last parameter for the parallel pattern in line 11. This pattern breaks the problem size into groups of team threads and determines vector widths.

Listing 1: Kokkos SPMV skeleton code

```

1  int team_size;
2  int vector_length;
3
4  //instantiate SPMV functor
5  SPMV_Functor<...> func (...);
6
7  //registration of the team-size and vector-length parameters
8  Kokkos::Profiling::autoTune(&team_size,&vector_length);
9
10 //These parameters are use by the Kokkos TeamPolicy to map to hardware
11 Kokkos::parallel_for("SPMV",
12                      Kokkos::TeamPolicy<Kokkos::Schedule<Kokkos::Dynamic>>
13                      (league_size, team_size, vector_length), func);
14
15 struct SPMV_Functor {
16 // ...
17 operator() (const team_member& dev) const
18 {
19     Kokkos::parallel_for(Kokkos::TeamThreadRange(dev,0, rows_per_team), [=](...) {
20         // ...
21         //perform vector reduction
22         Kokkos::parallel_reduce(Kokkos::ThreadVectorRange(dev, row_length), [=](...) {
23             // ...
24             lsum += ...;
25         }, sum);
26
27         // Add the results once per thread
28         Kokkos::single(Kokkos::PerThread(dev), [&] () {
29             // ...
30             m.y(iRow) = sum ;
31         });
32     });
33 }
34 //...
35 }

```

**3. Autotuner.** The autotuner is implemented as part of the Kokkos performance tools. These tools work by taking advantage of the existing hooks embedded into the Kokkos runtime. These hooks provide access to runtime information that is later reported back to the user by the tools. Currently, they can provide parallel kernel dispatch timings and memory usage and can generate data that 3rd party tools such as Intel's Vtunes [12] and Nvidia's NSight [10] can consume to produce performance reports. All of the tools are dynamically loaded and can be enabled by defining an environment variable. The autotuner was developed within the same framework but with the difference that it would run as a dynamic tool that would use the runtime information provided by the hooks as feedback to self-improve the parameters that are usually left for the application scientists to determine. The autotuner consists of two main components, the parameter registration interface, and

the search space iterator.

**3.1. Registration Interface.** For the autotuner to be able to modify the parameters that are fed into the existing Kokkos API, the user must first register the parameters using the registration interface designed for the autotuner. An example of the registration interface can be seen in line 8 of code listing 1. Registration works by keeping track of the pointers to those variables. These variables are then internally associated with a parallel pattern label—in this case, the `parallel_for` label, “SPMV” in line 11 of the same code listing. The association happens the moment the parallel pattern is executed. What this means is that variable registration should follow a particular order. Registration must be followed by a parallel pattern. If, for example, multiple pairs of parameters are registered in a row, only the last registration will be associated when the next parallel pattern is executed. Because the registration works by the aliasing of pointers and not any data copying, it has a negligible overhead. The registration step could be placed within an iterative algorithm that would register the same set of parameters to the same parallel pattern every iteration without incurring an overhead penalty.

**3.2. Search Space Iterator.** Once a set of parameters is registered and associated with a particular parallel pattern label, the search space iterator (SPI) is responsible for keeping track of the current team size and vector width. The SPI is also responsible for iterating through all possible combinations of parameters for a specific hardware. Each hardware platform has a different search spaces, which is directly related to the architecture features of a device. The search space for the architectures used in this paper are illustrated in Table 3.1. The limits for each device were determined by running a large selection of SPMV problems with different characteristics. We then picked the highest and lowest optimal parameters. Results confirm the close relationship between performance and hardware features. For the K40 [11] team sizes don’t go below 32 due to the warp size execution being an integral part of GPU architecture. No optimal combination of team size and vector width will go over 1,024 due to hardware limits. Knights landing (KNL) [13] vector size won’t go over 16 since no more vector lanes are available to use. It has two 512-bit vectors that can handle up to eight double-precision elements each. Its team size is limited by hardware resources. If we max out the KNL by using all 72 cores and the max four hyper threads per core and fully use the vector lanes to execute 16 elements we get 2,304, which is slightly above 2,048. Similarly, with the Haswell [8] architecture, the optimal choices match up with the available hardware.

The SPI is triggered whenever the same parallel pattern label is encountered. When executing a parallel pattern, the autotuner uses the existing Kokkos tools framework to search through a map of all pre-existing labels. The information about each parallel pattern instance is stored as a C++ class. This class records the kernel name, number of times called, and total execution time. We build the SPI as part of this class. We extend the information tracked to include data about the current and best team size and vector width. Thus, upon finding a matching label, the autotuner will trigger the SPI by updating the class.

We implemented the SPI using a naive approach of iterating over all possible options by increasing the vector size by two until the maximum vector length is reached, at this point we increase the team size by two and reset the vector length to the minimum size. The algorithm increases the vector size and the team size by two due to hardware features such as vector widths and number of hyper-thread being designed to be powers of two. When the SPI is activated it will record the current set of parameters, team size and vector width used, and the corresponding execution time. It will then check if the current execution time is faster than the current fastest set of parameters. If so, it will update the `best_team_size` and

Architecture	Max Team Size	Min Team Size	Max Vector Width	Min Vector Width
K40m	1024	32	32	1
Knights Landing	2048	1	16	1
Haswell	8	1	8	1

Table 3.1: Search-Space bounds per architecture: The limits for each device were determined by running a large number of SPMV problems. The resulting bounds nicely match hardware characteristics of each device.

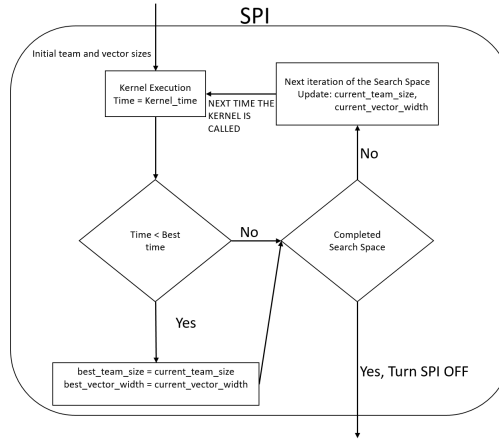


Fig. 3.1: Search Space Iterator (SPI) logic: Activated during the first invocation of the kernel, it will iterate through the possible team size and vector width parameters until the kernel is no longer called or it has finished iterating through the available options

best\_vector\_width accordingly. Finally, it will update the current set of parameters to reflect the next set in the search space to be used when encountering this parallel pattern again. Figure 3.1 shows the logic behind the SPI framework. When all the possibilities in the search space have been attempted, the SPI will default to the best set of parameters and turn itself off. This approach hinges on iterative applications that call the same computational kernels multiple times during execution of the program, where each time the SPI will attempt a new set of parameters. Kernels that are only executed a few times during the lifetime of an application will not be able to take advantage of this system unless the application runs multiple times.

**3.3. NNZ Hint.** In many cases, not all iterations of a search space are worth exploring due to the characteristics of a particular matrix. If a matrix has a low number of non-zeros per row, we could skip some of the search space iterations that we suspect won't results in good performance by taking that information into account. Some of the vector width choices in the search space can be discarded if we know there are not enough non-zero elements in a row to take advantage of all the vector lanes. Thus we can shrink the search space by adding a check that will limit the SPI to only iterate through parameters that can meet the above criteria. We accomplish this by adding an extra optional parameter to the autotuner that will register the number of non-zero elements per row that will later be used by the SPI for shrinking the iteration space. We call this extra optional parameter that represents



Architecture	Cores	Logical Cores	Frequency	GFLOPs (double)	Max. Memory	Max. Memory B/W
NVIDIA Tesla K40m GPU (Kepler)	15(SMX)	2880	745 MHz	1,430	12 GB	288 GB/s
Intel Xeon Phi (Knight's Landing)	72(Silvermont)	288	1.5 GHz	3,543	384 GB (DDR4) 16 GB (HBM)	90 GB/s (DDR4) 400 GB/s (HBM)
Haswell	32 (Xeon)	64	2.3 GHz	583	512 GB	68 GB/s

Table 4.1: Table of architectures details: No two architectures have the same value for any of the characteristics in the table. [2][14]

the number of non-zero elements per row hint NNZ.

**4. Results.** In this section, we present an evaluation of the effectiveness of the auto-tuner tool with and without the number of non-zeros per row hint. We compare it with two other schemes: the Fixed approach, and the Oracle. The Fixed approach stands for the conservative choice made by an application programmer that is likely going to use most of the available resources of the hardware without taking into account the matrix characteristics. We define the fixed parameters as {team size, vector width} of {1, 1} for CPU type devices and {32, 32} for GPUs. Finally, the Oracle will always use the best setup from the start. We evaluate these schemes using a large number of sparse matrix-vector problems taken from the University of Florida Sparse Matrix Collection [4], which contains a set of real-world matrices of different sizes and sparsity patterns from a wide variety of domains. We show that for only 100 iterations the autotuner is effective in finding the most optimal setup, incurring some overhead compared with the Fixed approach and in some cases substantially outperforming it.

**4.1. Experimental Setup.** All experiments were run on a wide variety hardware. These include a K40m GPU part of the Nvidia Kepler architecture, Intel's new member to the MIC series the Xeon-Phi Knights Landing, and an Intel Haswell CPU. We initially generated results for three GPUs, a K20x, a K40m, and a K80, but we chose the results for the K40m due to space constraints as well as the behavior of all GPUs were the same for all matrices. The K80 did not benefit much from its dual GPU setup. Table 4.1 summarizes these architectures in more detail. For the Intel KNL we also explore some different configurations that are possible with this architecture. We chose from a number of different clustering styles and memory type partitions. We ran on seven different setups that are possible from the many combinations and show results for two of the most used configurations: Alpha and Delta. All configurations are described in Table 4.2. Finally, for KNL and Haswell we ran with different environment variables. `OMP_NUM_THREADS = 256`, and `KMP_AFFINITY = compact`, to control the number of threads and placement. We chose different names/modes for how we ran the autotuner. In vanilla mode, no environment variables were used. For the KNL we named the optimized modes that used the environment variables based on the memory partition used, DDR4 or HBM. For Haswell, we use "Opt" as the name for runs that used the variables. All results are from executing the SPMV kernel for 100 iterations.

## 4.2. Experimental Results.

**4.2.1. K40m.** The normalized performance of the autotuner on the NVIDIA K40m can be observed in Figure 4.1. These results are sorted based on the sparsity pattern of each test matrix, from very sparse on the left to more dense matrices on the right. The number of non-zero elements per row is shown in parentheses next to the each matrix name. We can immediately notice a clear distinction based on the above description. Matrices that have a low number of non-zero elements per row performed very poorly when using a fixed size

KNL Configuration Name	Clustering Mode	Memory Mode
KNL-Alpha	Quadrant	Flat HBM
KNL-Bravo	SNC=4	NUMA/SNC partitioned (no cache)
KNL-Charlie	Quadrant	50/50 cache/HBM
KNL-Delta	Quadrant	100% cache
KNL-Echo	SNC=4	50/50 cache/HBM
KNL-Foxtrot	SNC=4	100% cache
KNL-Golf	SNC=2	Flat HBM
KNL-Hotel	SNC=2	NUMA/SNC partitioned (no cache)

Table 4.2: Table of KNL configurations: Cluster mode describes the communication pattern for a cache miss. Different clusters maintain different locality for the messages among the KNL computing tiles. In Quadrant mode the tiles are divided into four parts. Memory addresses served by a memory controller in the same quadrant is guaranteed to be mapped within the quadrant. SNC or sub-NUMA cluster does the same as the Quadrant mode but also treats the clusters as different NUMA domains. HBM stands for the high bandwidth memory, and the cache mode stands for DDR4 memory.

for the thread teams and vector width. This performance can be explained due to there not being enough uniform work per row take full advantage of a large degree of threads per team. It is important to note that matrices that have this behavior like *italy\_osm*, *road\_central*, *relat9*, *mc2depi*, and *memship* are vast matrices containing well over a million elements. It is well known that large irregular data sets perform poorly on the GPU due to divergence as shown in [1]. By forcing a large team of threads, it enforces a significant divergence penalty. Other highly sparse but far smaller matrices like *olm2000*, *lnsp131*, *lnsp511*, *dc1*, and *fullship* that only have a few thousand elements, by choosing a larger size team of threads and vector size will not incur significant amounts of divergence penalty. Thus the fixed choice performs almost optimally when compared to the Oracle. When we analyze the effectiveness of the autotuner for the large matrices with a low number of non-zero elements per row, 100 iterations are enough to make the tool worth using, as it performs many times better than the fixed choice. The improvement from the Regular approach to the NNZ approach varies between 73% in the best case to only about 15% in the worst case. For small matrices with a low number of non-zeros per row, there is essentially no benefit from the NNZ hint, and in many cases it hinders performance. The reason for this drop in performance is due to the optimal choice for vector width being larger than the number of non-zeros per row. Thus the NNZ approach never reaches the most optimal setup. As an example, we look more deeply into the different configurations chosen for *olm2000* and *lnsp511*. For *olm200* the number of non-zero per row is three. The NNZ approach maxed the vector width to two due to a vector width of four being larger than the number of elements per row. When we ran the Regular approach the, which searches the entire search space, the most optimal vector width was determined to be four. A similar case is found with *lnsp511* wich has five elements per row. The most optimal vector width is eight, but the NNZ picks a vector width of four.

When the number of elements per row increases to the teens, we see a slightly different result. In all cases, the autotuner perform worse than the fixed choice. What this means is that 100 iterations are not enough to amortize the overhead of the tool to at least break even regarding performance compared to the fixed choice. The difference in performance

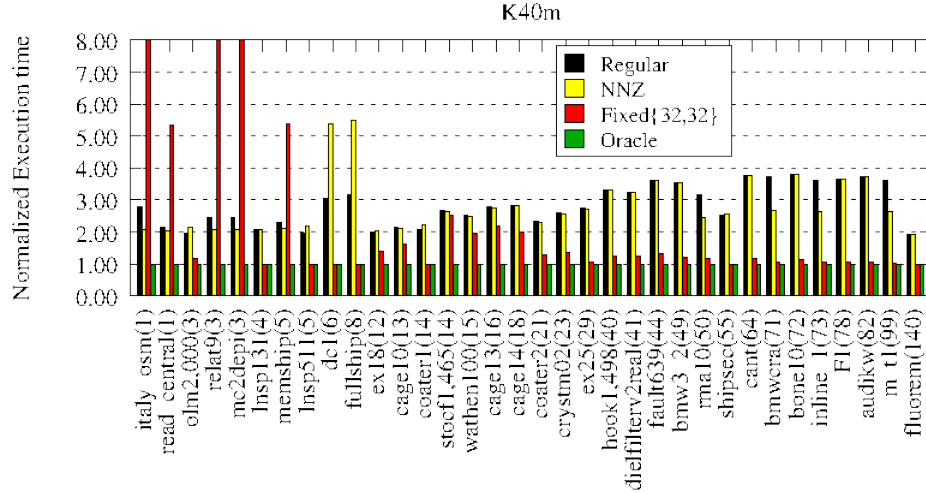


Fig. 4.1: Normalized performance of the autotuner on the NVIDIA K40m

between the autotuner and the Fixed approach is about 15% to 80%, and the difference between the fixed and the Oracle is 40% to 150%. In other words, because the difference in performance between the tool and the Fixed approach is not too large, we expect that it would not require a substantial increase in iterations for the autotuner to break even. Once beyond the break-even point, because the difference between the fixed and Oracle is significant, it would only take a few more iterations to make a noticeable improvement in performance using the autotuner. Finally, when the matrix density increases by increasing the number of elements per row to over 20, we see that the difference in performance between the autotuner and the Fixed approach significantly increases, and the difference between the Fixed approach and the Oracle shrinks substantially. What this means is that it would require a substantial increase in iterations to break even, and after breaking even it would take a further increase for the autotuner to make a significant difference. We plan to use all of these observations to improve the performance of the autotuner tool further by taking into account more properties of the matrix such as total size and the total number of iterations in the simulation when possible.

**4.2.2. KNL ALPHA.** The results for the Knights Landing architecture using the Alpha configuration are shown in Figure 4.2. These results are presented in the same fashion as previous results. Matrices are sorted from left to right based on the number of non-zero elements per row. Unlike the GPU results, this figure shows results for three different execution modes. The first mode is called Vanilla which stands for no optimizations, and no specific memory were used. The following mode is called DDR4 and stands for the specific memory configuration specified to the KNL to be used. This mode also uses environment variables to specify the maximum number of threads to be used and the clustering of the threads. Finally, the last mode, HBM, is similar to the DDR4 mode with the only difference being the type of memory specified to the KNL changed to High Bandwidth Memory. The Fixed approach was defined to be one thread per team and a vector width of one. Lastly, the Oracle was not calculated for this architecture.

Large matrices with a low number of non-zero elements per row behave very similarly to each other. The Fixed approach performs only marginally better than the autotuner. For the Vanilla mode, the fixed setup is about 5% to 40% better than the Regular approach and 2% to 30% better than the NNZ. For both the RDD4 and HBM this difference drops even further. For smaller, similarly sparse matrices we see a different behavior. For all modes, regardless of memory model or environment variables, the Fixed approach is always the worst-performing setup. The difference in performance is substantial: it is almost 5X slower for the Vanilla mode, 2X for DDR4 and 1.5X for HBM. The HBM mode is always the better configuration for these types of matrices, but only slightly when comparing against DDR4 memory mode. What this means regarding the autotuner is that for large matrices with very few elements per row, 100 iterations is not enough to hide the overhead of the tool, but because the difference in performance is subtle, it would not take much more to break even when compared to the fixed option. For small matrices, the benefit of the tool is evident. Among matrices with a slightly larger number of non-zero elements, between 12 to 29, we see a similar trend. The autotuner performs very well with small matrices and close to even among large matrices, with the HBM memory mode performing slightly better than DDR4 among all scenarios. For matrices with a substantially larger number of elements per row, 44 and above, we see a different pattern. For all modes, the autotuner performs very close to the Fixed approach, sometimes performing slightly better and sometimes slightly worse. What this means for the autotuner is that it takes about 100 iterations to hide its overhead for these type of matrices. Whenever NNZ performs slower than Regular, it is due to the picking a suboptimal vector length that was just outside the search space. The clear difference concerning the performance between matrices with a large number of non-zero elements compared to the rest is the substantial speedup the HBM memory mode has over the DDR4 mode. This performance difference ranges from about 1.5X to 3.5X. Due to not having the Oracle results, we could not make a prediction of how many more iterations, after breaking even, it would take for the autotuner to make a meaningful difference in performance. We are working towards getting these results.

**4.2.3. KNL Delta.** The results for the Knights Landing using the Delta configuration are shown in Figure 4.3 and are presented in a similar fashion to the previous architecture results. Matrices are sorted from left to right based on the number of non-zero elements. The difference in results between the Delta and the Alpha is that the HBM memory mode cannot be enabled for this configuration. Instead, the HBM memory is used as an extra level in the cache hierarchy. Thus HBM results are not shown on Figure 4.3. Similarly to the Alpha configuration, we see two kinds of behavior for matrices with a low number of non-zero elements per row. The performance results for these matrices differ depending on the size of the matrix. For large matrices, the autotuner tool performs very similarly to the fixed option for both Vanilla, and DDR4 modes. For the non-optimized Vanilla mode, the Regular approach performs only about 3% to 48% worse compared to the Fixed approach. Using the NNZ hint lowers that difference to zero or in the worst case to about 36%. When using the DDR4 the autotuner performs better, matching or slightly surpassing the performance of the Fixed approach for both Regular and NNZ or in the worst case performing only 27% to 18% lower than the fixed. What this means for the autotuner is that the break-even point, where the overhead from the tool is completely hidden, is at about 100 iterations.

From this point, any further iterations would get a benefit in performance compared to the Fixed. Similarly sparse but smaller matrices perform better than the Fixed approach by a significant margin. For some matrices, the difference can be from 22% to up to 3.4X for the Vanilla mode, and 60% to 1.5X for DDR4. For these types of matrices, the benefit of the tool is evident for 100 iterations. A similar behavior for both large and small matrices

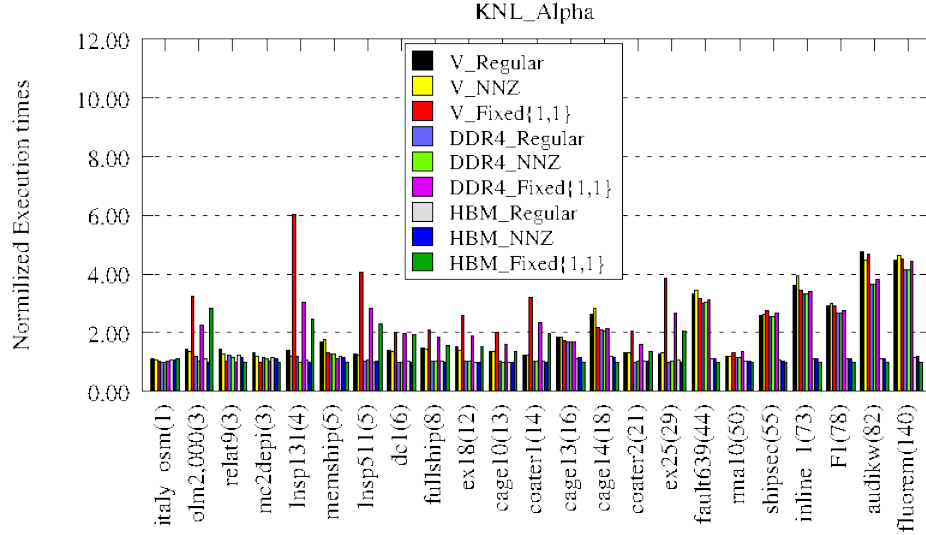


Fig. 4.2: Normalized results for the Knights Landing architecture using the Alpha configuration.

can be observed for matrices with a number of non-zero elements per row between twelve to fourteen. For matrices with a larger number of elements per row the autotuner performs very close to the performance of the Fixed approach. Both the Regular and NNZ approach are about 10% to 48% slower than Fixed for the Vanilla mode. For the DDR4 mode, the difference drops to nothing or in the worst case up to 30%. Similarly to the matrices with a low number of non-zero elements per row, we can conclude that about 100 iterations marks the break even point of the tool. Only two matrices with a large number of non-zero elements behave differently than the rest. Coater2 and ex25 show a substantial benefit from the autotuner.

**4.2.4. Haswell.** The result for Intel’s Haswell architecture are shown in figure 4.4. The results are laid out in the same fashion as the previous results. Matrices are ordered based on the number of the non-zero elements per row. Unlike KNL results, Haswell does not have multiple memory configurations, but we did run using two different modes. We ran using the standard Vanilla mode which is not using any environment variables, and we ran what we call the “Opt” mode which includes optimization such as the number of threads and thread placement. Similarly to other hardware, the behavior for matrices with a low number of non-zero elements per row can be separated between large and small matrices sizes. Large matrices, on order of millions of elements, show compelling results for the autotuner. The Regular Vanilla approach performed only in the worst case 28% slower than the Fixed and in the best case about 7% better than the fixed. These results do not vary too much when using the NNZ as a hint. The optimized results shows that the difference in performance decreases even further for the autotuner. The Regular optimized approach in the worst case performs 15% slower and in the best case 20% faster. Again, there is not much difference when using the NNZ hint. These results tell us that for 100 iterations the autotuner begins to overcome its overhead and starts to be a beneficial choice. Because we lack the results for the Oracle, we cannot make a reasonable prediction in how fast this

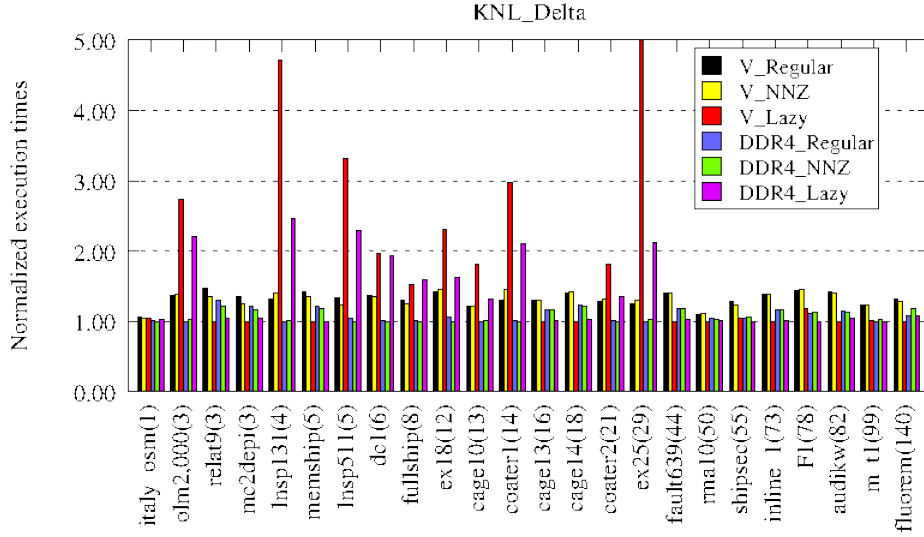


Fig. 4.3: Normilized results for the Knights Landing using the Delta configuration.

benefit would improve. On the other hand for small, sparse matrices, the autotuner always performs slower than the Fixed approach. The NNZ hint does provide an improvement but not substantial enough to compete with the Fixed approach. The autotuner can sometimes be up to 100% slower than the Fixed approach and 94% slower when using the NNZ hint. This difference also lower when running on the Optimized mode. Under the Optimized mode, we can narrow the difference to 26% slower in the best case and at 58% at the worst. NNZ improves this to 20% and 30% respectively. What we see from this behavior is that for the Vanilla approach the different between the autotuner and the Fixed method is too large. It would require a substantial increase in the numbers of iterations to pay the overhead of the autotuner. Using the Optimized environment we see a much more manageable difference in performance. As the number of non-zero elements increases, we don't see much difference between matrix sizes.

For matrices with non-zero elements per row in the range from twelve to 29, the autotuner perform slightly slower than the Fixed approach for both Vanilla and Optimized mode and both Regular and NNZ approaches. For most matrices, the difference can be only a few percent, but for cage10 and cage13 the difference can be up to 45% and 30%, respectively. In other words, for these type of matrices, the autotuner still needs a few more iterations to become beneficial for the overall performance. Finally, for a vast number of non-zero elements per row (over 55) we see almost no difference between the autotuner and the Fixed approach. The difference is single digits percentage points wich indicates that the autotuner had broken even and had managed to hide its overhead. Running for more iterations should provide an overall benefit compared to the Fixed approach.

**4.3. Autotuner selection.** To emphasize the importance of using an autotuner to find the most optimal set of parameters we present Table 4.3. In this table, we have picked a subset of the matrices used for out experiments, and for each matrix we show the most optimal team size and vector width for each architecture. What we can clearly see from the table is that each architecture had a substantially different set of optimal parameters.

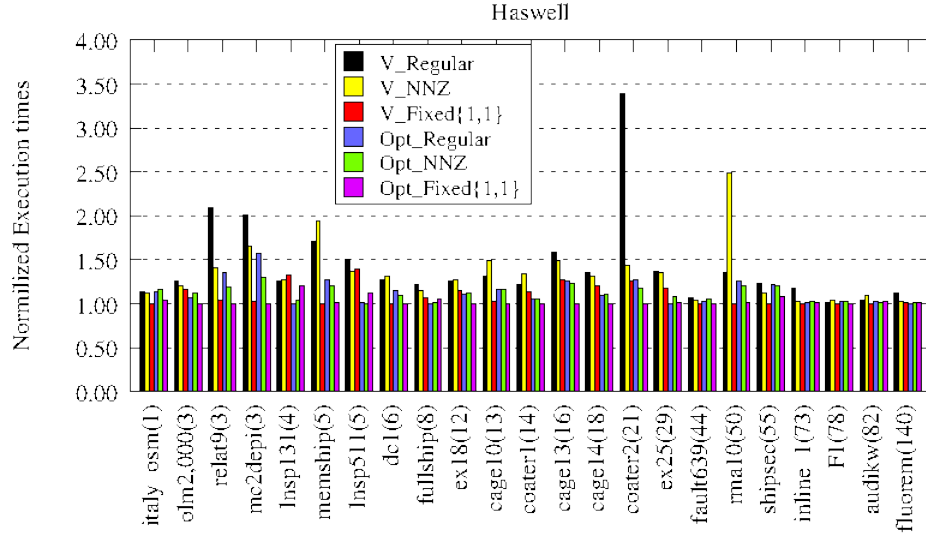


Fig. 4.4: Normalized results for Intel's Haswell architecture.

Matrix	# Of Non Zeros Per-row	K40m		KNL Alpha		KNL Delta		Haswell	
		Team Size	Vector Width	TeamSize	Vector Width	Team Size	Vector Width	Team Size	Vector Width
italy_osm	1	128	1	4	1	4	1	1	2
relat9	3	64	2	1	2	1	1	1	1
linsp131	4	128	4	16	2	16	1	4	1
fullship	8	32	32	4	4	4	4	2	1
cage10	13	64	8	4	1	4	4	1	1
coater2	21	32	8	4	2	2	4	1	2
fault639	44	32	16	1	4	1	2	1	4
shipsec	55	32	16	1	2	1	2	2	2
fl	78	32	16	1	4	1	4	1	1
fluorem	140	32	32	1	1	1	4	1	1

Table 4.3: Best-performing selection by the autotuner

Even the two KNL configurations, which expectedly have the most similar set of parameters, have the same parameters less than half the time. This difference between Alpha and Delta configurations for the KNL is mainly due to the memory used. What this would mean is that for the most optimal application performance, the programmer would have to re-tune his or her application when porting it between configurations of the same architecture. When comparing between KNL and Haswell, we see that the parameters usually decrease in size to match the available hardware. In only three instances does Haswell uses the same set of parameters as one of the KNL configurations. Lastly, the K40m's optimal parameters show in GPU's preference for a large team size when compared to CPU type devices. We also see that the vector width of the K40m and KNL Delta increases as the as the number of non-zeros per row increases, showing how the autotuner adapts to matrix features. The trend is not particularly true for KNL Alpha or Haswell as it may depend more heavily on other matrix features. What this table means, in general, is that portability across and within architectures has to be done by taking into account small hardware details and data characteristics. This puts too much of a burden on application programmers and should be automated by adaptive runtime tools.

**5. Conclusions.** SPMV is a highly-irregular computational kernel that can be found in many different scientific applications. To get good performance it is not only critical to take into consideration details of the architecture you are running on, but it is also highly dependent on the sparsity pattern of a particular matrix. In order to improve SPMV performance portability across many different kinds of hardware, we proposed to extend the current set of Kokkos tools with an autotuner that iterates over possible candidate parameters that are fed to Kokkos parallel patterns such as teams of threads size and vector width. The autotuner works by taking advantage of runtime information to choose the optimal parameters for a particular input matrix. We showed that this approach allows an iterative application to continue to progress towards a solution while lifting the burden of the application programmer from knowing the underlying hardware as well as to account for matrix characteristics. We compared the autotuner against a Fixed approach on three distinct architectures and showed that after about 100 iterations the optimal choice made by the autotuner was in many cases beneficial to the overall performance, especially when the number of non-zero elements per row was small. In other situations, the autotuner performed slower or very close to a Fixed approach, which signifies that the number of iterations was not sufficient to overcome the overhead of the autotuner. We also illustrated how the most optimal set of parameters for a particular matrix changes substantially between architectures, and that among the same architecture different sparsity patterns required different parameters. This highlights the need for an automated tool to lift the burden of performance portability from the application programmers.

## REFERENCES

- [1] N. BELL AND M. GARLAND, Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors, in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, ACM, 2009, p. 18.
- [2] C. CUDA, Programming Guide, 5 March 2015. NVIDIA Developer Zone: Website, 2015.
- [3] L. DAGUM AND R. MENON, OpenMP: an Industry Standard Api for Shared-Memory Programming, *IEEE computational science and engineering*, 5 (1998), pp. 46–55.
- [4] T. A. DAVIS AND Y. HU, The University of Florida Sparse Matrix Collection, *ACM Transactions on Mathematical Software (TOMS)*, 38 (2011), p. 1.
- [5] H. C. EDWARDS, D. SUNDERLAND, V. PORTER, C. AMSLER, AND S. MISH, Manycore Performance-Portability: Kokkos Multidimensional Array Library, *Scientific Programming*, 20 (2012), pp. 89–114.
- [6] H. C. EDWARDS, C. R. TROTT, AND D. SUNDERLAND, Kokkos: Enabling Manycore Performance Portability through Polymorphic Memory Access Patterns, *Journal of Parallel and Distributed Computing*, 74 (2014), pp. 3202–3216.
- [7] K. GREGORY AND A. MILLER, C++ AMP: Accelerated Massive Parallelism with Microsoft Visual C++, (2014).
- [8] P. HAMMARLUND, R. KUMAR, R. B. OSBORNE, R. RAJWAR, R. SINGHAL, R. D’Sa, R. CHAPPELL, S. KAUSHIK, S. CHENNUPATY, S. JOURDAN, ET AL., Haswell: The Fourth-Generation Intel Core Processor, *IEEE Micro*, (2014), pp. 6–20.
- [9] J. JEFFERS AND J. REINDERS, *Intel Xeon Phi Coprocessor High-Performance Programming*, Newnes, 2013.
- [10] N. NSIGHT AND V. S. EDITION, 3.0 User Guide, NVIDIA Corporation, (2013).
- [11] T. NVIDIA, K40 GPU Active Accelerator, Board specification, (2013).
- [12] J. REINDERS, *VTune Performance Analyzer Essentials*, Intel Press, 2005.
- [13] A. SODANI, Knights landing (KNL): 2nd Generation Intel® Xeon Phi Processor, in *Hot Chips 27 Symposium (HCS)*, 2015 IEEE, IEEE, 2015, pp. 1–24.
- [14] A. SODANI, R. GRAMUNT, J. CORBAL, H.-S. KIM, K. VINOD, S. CHINTHAMANI, S. HUTSELL, R. AGARWAL, AND Y.-C. LIU, Knights Landing: Second-Generation Intel Xeon Phi Product, *IEEE Micro*, 36 (2016), pp. 34–46.
- [15] J. E. STONE, D. GOHARA, AND G. SHI, OpenCL: a Parallel Programming Standard for Heterogeneous Computing Systems, *Computing in science & engineering*, 12 (2010), pp. 66–73.
- [16] C. R. TROTT, M. HOEMMEN, S. D. HAMMOND, AND H. C. EDWARDS, Kokkos the Programming Guide,



- (2015).
- [17] R. W. VUDUC, *Automatic Performance Tuning of Sparse Matrix Kernels*, PhD thesis, University of California, Berkeley, 2003.
  - [18] S. WIENKE, P. SPRINGER, C. TERBOVEN, AND D. AN MEY, OpenACC First Experiences with Real-World Applications, in *European Conference on Parallel Processing*, Springer, 2012, pp. 859–870.

## TRANSITIONING GREEN-GAUSS GRADIENTS TO THE KOKKOS FRAMEWORK

WILLIAM B. HELD\* AND ANDREW M. BRADLEY†

**Abstract.** Sandia Parallel Aerodynamics and Reentry Code (SPARC) is a research code for compressible CFD model and algorithm development. SPARC is currently being extended to use Kokkos [2], a C++ programming model for writing performance portable applications targeting all major HPC platforms. This paper outlines the software narrative of updating the Green-Gauss gradient kernel in SPARC to use Kokkos.

**1. Introduction.** The Green-Gauss formulation is a discretized gradient calculation that integrates around the boundary of closed control volumes. It is used in the equation assembly phase for many Computational Fluid Dynamics (CFD) problems inside of the Sandia Parallel Aerodynamics and Reentry Code (SPARC) [5]. Green-Gauss gradients had previously been thread serial inside of SPARC, which, when run on the GPU, required a series of costly data synchronizations between the CPU host and GPU device. This project converted Green Gauss gradients into the Kokkos framework. Kokkos provides abstractions for the parallel execution of code and for data management that enable the same code to achieve performance on a variety of modern HPC platforms. For our purposes, Central processing unit (CPU) and Graphics processing unit (GPU) were the primary platforms tested upon with GPU posing most of the software development challenges.

The average directional derivative over a control volume  $\Omega$  having volume Vol for a vector  $u$  with direction vector  $s$  is

$$\langle u \rangle_d \equiv \frac{1}{\text{Vol}} \int_{\Omega} s \cdot \nabla u \, d\Omega = \frac{1}{\text{Vol}} \int_{\partial\Omega} u \, s \cdot d\partial\Omega, \quad (1.1)$$

where the equality follows from integration by parts. In a cell-centered method, the second integral can be discretized as

$$(\langle u \rangle_d)_i = \frac{1}{\text{Vol}} \sum_{k \in \mathcal{F}_i} \frac{u_i + u_k}{2} (s \cdot a_{ik}) \quad (1.2)$$

for cell  $i$ ,  $\mathcal{F}_i$  the set of cells sharing a face with cell  $i$ , and  $a_{ik}$  the face area vector pointing from cell  $i$  to cell  $k$ . If we set  $s$  to each of the Cartesian basis vectors  $e_i$ ,  $i = 1, 2, 3$ , we obtain  $\langle u_x \rangle_i$ ,  $\langle u_y \rangle_i$ , and  $\langle u_z \rangle_i$ , which is the cell-centered Green-Gauss gradient [5].

In the code, the calculation is performed by iterating over structured faces of a mesh and calculating the gradients of each face. These face gradients are then summed to ultimately form the cell gradients. The path of traversal and the gradient calculation differ slightly depending on the various face types which reside within the mesh. Due to this, Green-Gauss gradients can be broken down to loops for interior faces, block interfaces, and boundary faces. Interior faces exist inside a block and affect two adjacent cells in the block. Interfaces operate between blocks, affecting one cell in the block and one cell outside the block. Boundary faces are affected by the boundary conditions of the mesh, but only affect one cell inside the mesh.

## 2. Code Structure.

---

\*New York University Abu Dhabi, wbh230@nyu.edu

†Sandia National Laboratories, ambradl@sandia.gov

**2.1. Kernel Style.** SPARC is an object-oriented code. Therefore, Kokkos functions used are designed as kernels that take only the data they need and modularly contain code complexity. A kernel is created as a class. Inside that class are the two key functions `Compute()` and `Run()`. `Compute()` defines the calculations which you are performing in parallel and is defined as a Kokkos inline function. `Run()` handles the calls to the compute function by initializing the iterator values and iterating over the faces of the mesh, which will be hereafter referred to as a mesh traversal. A toy kernel is declared below exhibiting a typical kernel's structure. This toy kernel will be used throughout this paper to illustrate patterns used in the Green-Gauss kernel.

```
class ToyKernel :
    // This kernel is part of a family of ToyKernels.
    public ToyKernelBase<ExeSpace>,
    // This kernel uses the mesh traverser service class.
    public MeshTraverserKernel<ToyKernel<ExeSpace>, ExeSpace>
{
private:
    typedef MeshTraverserKernel<ToyKernel<ExeSpace>> MT;

    KOKKOS_FORCEINLINE_FUNCTION
    void compute(const int& k, const int& j, const int& i) const;

    virtual void Run();
};
```

The Green-Gauss gradient calculation is made up of two kernels that encapsulate the three loops described in the introduction. Interior faces and interfaces are computed in the non-boundary condition kernel, and boundary faces are computed in the boundary condition kernel. Interior faces and interfaces are able to be computed in the same kernel because they use an identical `Compute()` function, meaning that their differences can be handled in the `Run()` function. The boundary faces, on the other hand, have a different `Compute()` function and therefore are implemented in a separate kernel.

**2.2. Compile-time polymorphism.** Inside of the `Compute()` of both of the Green-Gauss Gradient kernels, there is a call to another function, which calculates the face contribution. The face contribution calculation changes based on face type, so the function is external to the kernel. The face contribution calculation had been implemented through the use of runtime polymorphism, as is standard for object-oriented code. In C++, runtime polymorphism is implemented using a virtual table (vtable), a table of function pointers mapping an object's virtual function to the function implementation. However, when a vtable is copied to a GPU, it is copied byte-wise. The copy then resides in a different address space than the one in which it was constructed. Consequently, the vtable contains invalid pointers. Although they present a challenge on GPU, object-oriented design principles, including polymorphism, are widely used and found helpful in the design and maintenance of large software applications. These conflicting problems are resolved by using compile-time polymorphism, which is implemented using templates. Templates are resolved at compile time; hence any function that is dispatched through a template is resolved without a function pointer, avoiding the vtable issue. The disadvantage of this approach is it increases object file size and build time. We mitigate build times by using explicit template instantiations. One or a few instantiations are performed in each of a large number of automatically generated translation units. Then, we can run the build with 32 cores, for example, and build the kernels efficiently in parallel.

### 3. Non-Boundary Kernel.

**3.1. Abstracting the Compute function.** Although visually similar, the face gradient calculation is computationally different for I, J, and K faces. In the Green-Gauss kernel, these differences were abstracted away by using a struct called Indexer. Indexer modifies variables depending on the index slot given as a template parameter. In the definition below of the Compute() function from the toy kernel, a simple example is shown of how the Indexer function Add() is used.

```
KOKKOS_FORCEINLINE_FUNCTION
void compute(const int& k,
             const int& j,
             const int& i) const
{
    int kp = k, jp = j, ip = i;
    Indexer<>::Add(kp, jp, ip, slot, -1);
    data(kp,kj,ki) = 2;
}
```

Depending on the value of slot, Add() subtracts one from the variable in the corresponding index slot. This allows the compute function to set a different location inside the variable data equal to 2. Using this method in Green-Gauss allows for the Compute() function to be abstracted from the face index type. Due to this, the different index values can be handled in the Run() function. This enables all of the interior face, and later interface, calculations to take place inside a single kernel. If indexing dominates a kernel's runtime, one can use a version of Indexer templated on the slot.

**3.2. Interior Faces.** The interior face calculation was the first to be implemented, as it is the majority of the calculation and the initial phase of the calculation. It is contained within a loop that iterates over every block from the input mesh and sets up a sequence of non-boundary condition kernels. In a serial implementation, each block is iterated over three separate times for the I, J, and K faces in the block. By Eq. 1.2, each face gradient is calculated and then summed into the gradient for the cell. However, since each face affects both of the cells it borders, this summation causes a race condition when performed in parallel. Atomic writes can be used to avoid race conditions. For our structured mesh, graph coloring is very simple and is an alternative to atomic writes. Experiments in other parts of the code showed that graph coloring is faster by 5–10% on a conventional CPU than the fastest atomics-based implementation. Therefore, we have opted to use graph coloring in this kernel, as well. A 2-color graph coloring is done in the direction of the iterating faces for the I, J, and K loops. In the Run() definition of the toy kernel below, an implementation of the 2-color graph coloring is shown for a simple  $10 \times 10$  mesh.

```
public:
    IndexSlot slot;
    // Runs through the entire mesh with 2-coloring
    virtual void Run()
    {
        // Set the slot value for Compute()
        slot = I_SLOT;
        // Set the start, stride, and end for traversal
        this->InitializeTraversal (0, 1, 9,
                                0, 1, 9,
                                1, 2, 8);

        this->MT::Run();
    }
```

```

this->InitializeTraversalI(2, 2, 8);
this->MT::Run();
slot = J_SLOT;
this->InitializeTraversal (0, 1, 9,
                          1, 2, 8,
                          0, 1, 8);

this->MT::Run();
this->InitializeTraversalJ(2, 2, 8);
this->MT::Run();
slot = K_SLOT;
this->InitializeTraversal (1, 2, 8,
                          0, 1, 9,
                          0, 1, 9);

this->MT::Run();
this->InitializeTraversalK(2, 2, 8);
this->MT::Run();
}
};

```

Each traversal first goes across the mesh skipping every other face, later following with a second traversal which calculates and sums gradients for the faces previously skipped. In Green-Gauss since faces only update the same cell when they are direct neighbors, this alternating graph coloring avoids the race condition without using atomics.

**3.3. Interfaces.** The interface calculation was the second step of implementation due to its high commonality with the interior face calculation. It is contained within a loop inside `Run()`, which iterates over the interface boundary conditions to provide it access to interface type data used to create the traversal for the interfaces.

The face gradient calculation and the summation that takes place in the `Compute()` phase are identical for both interior faces and interfaces. However, the way that interfaces are iterated through is different and therefore requires a unique mesh traverser call within the `Run()` function of the non-boundary condition kernel. Two values are pulled from the interface type data: the index slot and a boolean which indicates whether the interface is a minimum or maximum face. These values are then used to initialize a mesh traversal down only the faces that border another block. In the interface loop, each face only affects one cell within its own block; therefore, there is no race condition on these cells. It also updates the cell gradient value for another cell in the block that it shares a border with, known as a ghost cell. Ghost cell gradients also do not have a race condition. Thus, the traversal can have a stride of 1.

**3.4. Organization.** The design decision was made to implement all of the loops of both interior faces and interfaces within the `Run()` function of this kernel. This makes the code simple at surface level as only a single call of `Run()` is written and the code complexity is contained within the kernel itself. The code snippet below illustrates the simplicity of this organization as it appears in the source file.

```

// Toy Function as it appears in .C file
void ToyFunctionForIllustration()
{
    for(const auto& data : data_chunks)
    {
        auto toy_kernel = ToyKernelBase<>::Create(data);
        toy_kernel->Run();
    }
}

```

```

}

```

**4. Boundary Kernel.** Faces affected by the boundary conditions have the most varied method of calculation, since they only affect the gradient value of a single cell and the calculation is changed by what type of boundary the face resides upon. The boundary condition calculation is written as an entirely separate kernel, which takes an extra argument of boundary condition data.

The traversal for boundary conditions follows the same structure as that of the interfaces, initiating the iterators based on the index slot and whether or not the face is a minimum face. However, for boundary conditions, which of the two affected cells is the ghost cell makes a computational difference. Therefore, the boolean that defines whether the face is a minimum or maximum face is used to define two separate calculations within the `Compute()` function.

**5. Calculation Wrap-Up.** To finalize the calculation, each cell gradient is divided by the volume of the cell. This calculation was placed into a simple kernel that iterates over the entire mesh by cell and divides every cell's gradient by its volume.

**6. Performance Results.** With Green-Gauss natively running on both GPU and CPU, it is important to understand how performance is comparable on each of these architectures. For the experiments we used Sandia's Shiller [4] Testbed, comparing 32 core Haswell node to a NVIDIA K80 [6] package, using 2 devices. The comparisons show the minimum time, maximum time, and mean time utilized by the Green-Gauss calculation as well as the mean time for each call of the function. The timings were measured using the Teuchos [1] package of Trilinos [3]. The Thermal Design Power (TDP) of a 32 core Haswell node is 330 Watts. Comparatively, the TDP of a 2 device K80 package is 300 watts.

Architecture	32 Core Intel Xeon Haswell	NVIDIA K80m Package
MinOverProcs	9.849s (100 calls)	19.02s (100 calls)
MeanOverProcs	10.08s (100 calls)	19.03s (100 calls)
MaxOverProcs	10.41s (100 calls)	19.05s (100 calls)
MeanOverCallCounts	.1008s	.1903s

While the K80 package performs worse than a 32 core Haswell node for Green-Gauss, it is important to note that GPU performance is superior in the solve phase of SPARC, meaning that for solve-dominated problems it is advantageous to use the GPU. Therefore, it is important to kernelize the assembly phase to lose as little performance as possible in the assembly phase of GPU runs.

**7. Conclusions.** As one of two second-order accurate discretizations of gradients used in SPARC, Green-Gauss is key to the equation assembly phase of many CFD problems. By transitioning it to the Kokkos programming model, expensive data synchronizations were removed and SPARC's scalability to modern HPC architectures was improved. The 3 face types were implemented within 2 kernels to create a source code that is simple to understand from an application programmer's perspective. Each of these kernels were implemented while maintaining the object-oriented code style, which had previously existed in SPARC. The completion of this kernel constitutes a second important group of code paths in SPARC developed entirely for GPU, contributing to SPARC's performance portability on modern HPC architectures.

## REFERENCES

- [1] R. A. BARTLETT, Teuchos:: RCP Beginners Guide, Sandia Report SAND2004-3268, (2010).
- [2] H. C. EDWARDS, C. R. TROTT, AND D. SUNDERLAND, Kokkos: Enabling Manycore Performance Portability through Polymorphic Memory Access Patterns, *Journal of Parallel and Distributed Computing*, 74 (2014), pp. 3202–3216.
- [3] M. A. HEROUX AND J. M. WILLENBRING, A New Overview of the Trilinos project, *Scientific Programming*, 20 (2012), pp. 83–88.
- [4] T. JAIN AND T. AGRAWAL, The Haswell Microarchitecture—4th Generation Processor, *International Journal of Computer Science and Information Technologies*, 4 (2013), pp. 477–480.
- [5] D. J. MAVRIPLIS, Revisiting the Least-Squares Procedure for Gradient Reconstruction on Unstructured Meshes, *AIAA paper*, 3986 (2003), p. 2003.
- [6] NVIDIA, NVIDIA Kepler GK110 Architecture Whitepaper, (2015).

## A GENERALIZED VECTORIZATION MECHANISM FOR VTK-M

ALOK HOTA\* AND KENNETH MORELAND†

**Abstract.** We propose a general method for achieving efficient vector processing with the Visualization Toolkit for Multi- and Manycore systems (VTK-m) framework [14]. In addition to threads, effective vector unit utilization is imperative for high performance code running on x86-based [6] multicore and manycore processors. This method can be fitted for the templated worklet mechanism existing in VTK-m to provide automatic vectorization of visualization algorithms. We show that within certain conditions, this method can result in speedups of over 2x in both arithmetic- and memory-intensive functions.

**1. Introduction.** Data visualization can be described as the process of efficiently transforming raw data into an intuitive, informative, human-readable format, often in real time. The data visualization pipeline is generally composed of three steps: data loading, data processing (also called filtering), and rendering. Achieving high performance in loading and rendering is primarily a hardware task, as parallel file systems with solid state cache and cutting edge graphics chips take the brunt of the work. However efficient data filtering is squarely a computational problem.

In response to continuously growing datasets, data filtering algorithms must be scalable to achieve the real time performance data scientists expect. This is especially a concern for in-situ visualization. Libraries like the Visualization Toolkit (VTK) [19], which form the backbone for visualization packages like ParaView [1] and VisIt [3], are invaluable as they allow for simpler development of visualization algorithms.

With the current push for exascale computing, computation on accelerators outside of the confines of a CPU and host memory are required. For example, the Titan [16] supercomputer at Oak Ridge National Laboratories achieves 99% of its peak theoretical performance via its GPUs. Other examples are the upcoming machines Trinity [11] and Cori [15]—at Los Alamos National Laboratories and the National Energy Research Scientific Computing Center, respectively—which will contain Intel’s Haswell generation CPUs and Knights Landing (KNL) generation Xeon Phi processors [8]. Utilization of the 72 cores on the KNL and the thousands of streaming cores on GPUs is essential. Visualization algorithms will require parallel code execution across different parallel programming models in order to run on current and future machines.

The accelerator platform diversity challenge is being met with VTK-m [14], which was created out of the group of packages used to augment and complement VTK for cross-platform parallelism, such as PISTON [10], Dax [13], and EAVL [12]. VTK-m aims to provide an abstracted development platform for visualization algorithms that can be run across a breadth of multicore and manycore hardware. VTK-m’s heavily templated and object oriented code present a challenge for automatic vectorization. In this work, we approach a solution to a generalized mechanism to aid automatic vectorization of VTK-m code compiled for x86-based platforms [6].

In Section 2 we describe the necessary contextual information regarding VTK-m, vectorization, and memory structuring. Section 3 outlines the various experiments performed to determine a potential vectorization solution. Finally, Section 4 describes how measurements were taken regarding performance gains, and details on results of the experiments are given.

## 2. Background.

---

\*University of Tennessee Knoxville, ahota@vols.utk.edu

†Sandia National Laboratories, kmorel@sandia.gov



**2.1. VTK-m.** VTK-m was designed to meet the challenge of simplifying visualization code development for cross-platform parallelism for current and upcoming machines. For example, VTK-m currently supports NVidia GPUs using the Thrust library [2] and threads for multicore and manycore nodes using Intel’s TBB library [17]. VTK-m additionally supports vectorization by way of compiler-specific preprocessor macros, which can give an additional performance boost to serial or threaded code.

The unified, general interface for different architectures is realized with worklets. A worklet is a serial functor operating on a small segment of the data being processed. They can be thought of as a generalized CUDA kernel. The developer is expected to write the functor (or use ones provided). Based on the architecture being targeted, the worklet is transformed into the appropriate representation at compile time. For example, worklets would be converted to kernels using Thrust if the CUDA platform were targeted.

VTK-m contains two types of worklets: field and topology. Field worklets map a function onto arrays; this is logically the same as iterating over an array and applying a function to each value. In this case, vectorization is an obvious performance booster, as this is exactly the type of computation at which vector operations excel. Topology worklets, on the other hand, have a less straightforward computational signature, as they work on the topological connections in a dataset. This could be the points, cells, or both in a grid. Due to the diverse nature of datasets, grids can be irregular, leading to haphazard memory access patterns in topology worklets. In-situ visualization exacerbates this, as memory layout is now completely out of the hands of the developer. The aim of the vectorization experiments is to ultimately find a generalized mechanism that can vectorize both field and topology worklets.

**2.2. Vectorization.** Vector instructions on CPUs are a type of Single Instruction Multiple Data (SIMD) processing. Traditional serial computation requires the data to be loaded into a register, a call to some operation, and the result to be offloaded from a register to memory. Figure 2.1a shows an example of iterating over an array to increment each value by 1. Note that there are eight loads into registers and eight additions.

This is contrasted with vectorized code, which operates using a vector unit and vector registers. This can greatly reduce the number of arithmetic function calls at the expense of loading the vector register. Continuing the previous example, a segment of the array would be loaded into the register simultaneously as opposed to individually. Once loaded, the vector unit then calls an add operation against the entire vector register. The number of values loaded depends on the vector width and the data type of the array values. On modern AVX2 [9] processors, the vector register width is 256 bits, which is capable of holding 8 single precision floating point values, or 4 double precision values. Assuming single precision values (or integers), the number of additions has reduced from eight in the serial version to just one in the vector version for each segment of the array. Figure 2.1b shows the vectorized version in comparison to serial.

Vectorized code can be automatically generated by the compiler given the correct conditions. Function calls break automatic vectorization unless the function is inlined. Without inlining functions, the compiler cannot read the contents of the function when determining vectorized sections. In some cases, there may be several layers of nested inline code, or execution paths that would stall the compiler from generating automatic vectorized code sections. Developers can then include explicit preprocessor instructions in loops that should be vectorized. Note that this does not guarantee vectorization, however.

Vectorization performance can be measured using a few metrics. In the previous example, the maximum theoretical gain is 8. This is determined by the maximum number of elements that populate the vector register, or the number of cycles that the vectorized

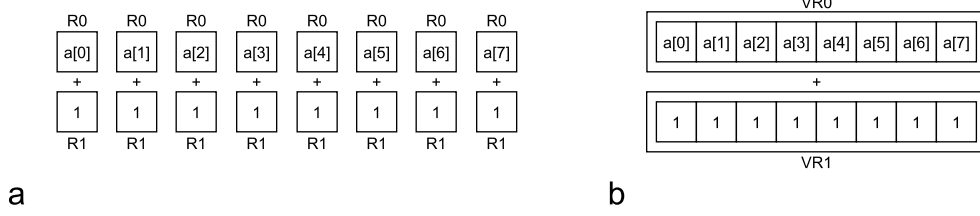


Fig. 2.1: Comparison of the conceptual difference between serial (a) and vectorized operations(b). Note that for the same functionality, the serial version has eight separate load and add operations in contrast to the single vector load and vector add operation.

version saves from the serial version. The actual gain observed may be less than the maximum gain; lowered efficiency can be caused by remainder loops if the array length is not a multiple of the vector width or by data dependencies forcing fewer values to be operable at once. Vector efficiency is calculated as actual gain divided by maximum theoretical gain. Naturally, efficiency at or near 100% is desired.

The last main measure of vectorization performance is speedup, which can either be measured as the ratio between processor cycles, wall time, or CPU time of the vectorized version versus the serial version. Wall time speedup is arguably the most immediately noticeable metric, and is strongly affected by memory access times, making efficient management of memory a critical component of vectorization. If the values to be loaded into an array are scattered across the heap (common for point coordinates or data values), access time will be much slower than if the values were consecutively arranged in an array on the stack. Array accesses can invoke burst reads, in which access time will be much lower than random access. If array segments can fit into cache—and especially if data is reused in the operation—cache misses will be greatly reduced, further speeding up execution.

**2.3. Loop Tiling.** Loop tiling, or strip mining, is a common solution to memory management to reduce access times and efficiently utilize cache memory. Explicit loop tiling can be used to aid the compiler in vectorizing segments of code and intelligently filling cache. At its core, loop tiling involves splitting a loop into one or more nested loops. The inner loop then iterates over a smaller segment of the original loop bounds. Given an array of length  $L$  elements, a simple loop to increment all elements of the array is given in Algorithm 2.1. The tiled version of this loop using a tile size of  $t$  is shown in Algorithm 2.2.

---

**Algorithm 2.1** Serial loop to increment values of an array  $A$  of length  $L$

---

```

for  $i \leftarrow 0, L$  do
   $A[i] \leftarrow A[i] + 1$ 

```

---



---

**Algorithm 2.2** Tiled loop with a tile size  $t$  to increment values of an array  $A$  of length  $L$

---

```

for  $i \leftarrow 0, L/t$  do
  for  $j \leftarrow 0, t$  do
     $k \leftarrow i \times t + j$ 
     $A[k] \leftarrow A[k] + 1$ 

```

---

If the inner loop is given preprocessor macros for vectorization, the compiler may find it easier to vectorize. The shorter inner loop length allows for values to be stored in cache for faster access during the loop execution. In this specific example, introducing tiling would most likely not affect performance strongly, or could even decrease performance. This is because loop tiling is highly sensitive to the amount of work done inside of the inner loop, data reuse, and stride. This example is trivial in terms of work done, which compilers may be able to optimize automatically without explicit instruction or tiling, depending on optimization level selected at compile time.

Loops that iterate over multiple large arrays using values from each for calculation are good candidates for vectorization. This is especially the case if values from each array must be accessed multiple times during the loop. A common example of this is a vector cross product function, which is described and tested in Section 3.

The stride of the array, or the distance between successive elements, also requires careful consideration. Ideally, elements should be packed such that values that will be accessed in succession are at unit stride. That is, the distances between the values' addresses are equal to the size of the value type. Modern vector architectures (such as AVX2) can use vector load operations to access elements that are at a non-unity but consistent stride. However, it is still beneficial to have unity stride for code that can be vectorized easily across platforms. Stride is especially an issue for algorithms that deal with grid coordinates, such as point-to-cell algorithms like cell averaging, which is also described and tested in Section 3.

**2.4. Array of Structures vs. Structure of Arrays.** Stride is also affected by the way values are stored. One dimensional C-style arrays of primitives afford simple access as only an index is required. VTK-m provides cross-platform compatible types that often fall back to primitives, such as `IdComponent` (int), `Float32` (float), and `Float64` (double). Additionally, VTK-m provides the `ArrayHandle` class, which behaves like a smart pointer. For the simple types described above, the `ArrayHandle` access methods resolve to a simple pointer, which is a direct memory access.

However, storing objects becomes problematic for access and vectorization. A very common example is a vector of 2, 3, or 4 elements, used often for point coordinates and vector data variables. VTK-m provides a templated `Vec` class that allows users to create such vector objects. When dealing with many `Vec` objects, it is conceptually simple to store the objects in an array and iterate over the array for calculations. This pattern is called Array of Structures (AoS). Though it is the most straightforward representation, this pattern inhibits vectorization due to indirect memory access patterns to read and write members of the objects, which leads to irregular strides. For arrays of point coordinates or data values, the vector objects may also be non-sequentially distributed in memory, as shown in Figure 2a. As mentioned before, this is exacerbated with in-situ visualization, where there is no control over memory layout.

The opposing design pattern is Structure of Arrays (SoA), which represents the same information using one object that contains multiple primitive-element arrays (though the container object is not necessary). For example, with 3D vectors, an SoA representation would dictate having three arrays which contain the x, y, and z components, respectively. In this representation, the original vector objects could be reformed using values at the same index from each array. Figure 2.2a shows an example of AoS representation of some 3D vector data using VTK-m data structures. The same data is shown in the corresponding SoA representation in Figure 2.2b. SoA representation makes vectorization much simpler due to the consistent unity stride and direct memory access to primitives instead of objects.

VTK-m's worklet structure utilizes `ArrayHandles` to store objects, meaning it uses AoS representation by default. Thus, the elements must be copied out of the array into

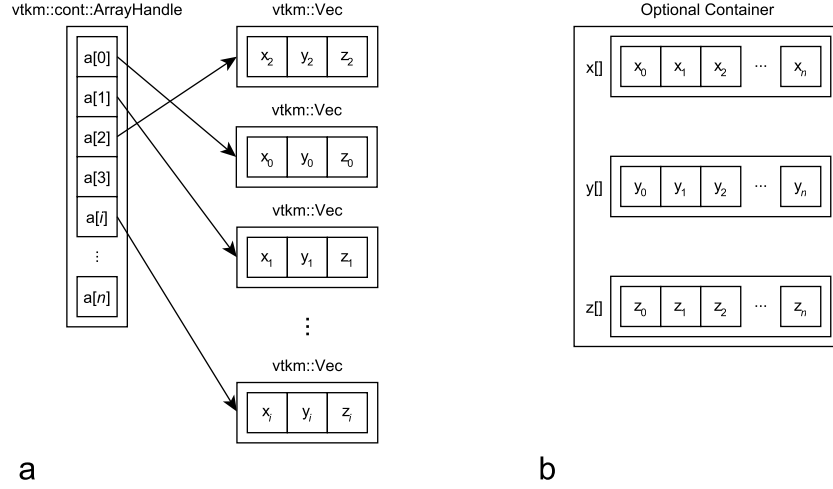


Fig. 2.2: The difference in memory layout between AoS (a) and SoA (b) representations. While intuitive, AoS inhibits vector loading due to indirect accesses. SoA enables much more efficient vectorization due to unity stride and direct memory access.

SoA representation at runtime. If done properly, the overhead of copying data is greatly outweighed by the improved vector performance. Because the data is in AoS form when a function is called, and only a partial SoA transformation is applied in the inner tiled loop, this pattern is sometimes referred to as a hybrid of the two, called AoSoA.

**3. Tests & Implementations.** Since field worklets operate over arrays, the internal scheduler that calls the worklet functor simply uses a for loop of the appropriate length when running on a CPU. Because of this, the initial approach to vectorization in VTK-m was to unilaterally force vectorization on all field type worklets. Forcing vectorization in this loop, however, did not always work; the compiler would still be unable to vectorize if the computation was too complex to automatically convert into vector operations. Additionally, though most function calls are inlined in VTK-m, the nesting level may just be too deep for the compiler. When the compiler was successful, performance was often worse than without vectorization. This was generally due to vector registers not being filled for each operation. Without dense vector usage, the overhead of copying into the vector unit outweighs the decreased number of operations.

**3.1. Initial Tests.** In order to formulate a generalized mechanism for automatic vectorization, the overall existing performance needed to be understood. The question being asked was, given the existing VTK-m structure with unilateral vectorization, what types of functions resulted in increased performance? Knowing this information gives insight on how certain function characteristics affect vector performance.

**3.1.1. Increment Worklet.** The first worklet created was the original increment example given in Section 2.2. The increment worklet simply took an array as input and iterated over it, adding 1 to each value.

This operation was initially expected to vectorize very well. However, while vector registers were engaged, overall performance was degraded. This is due to the low amount

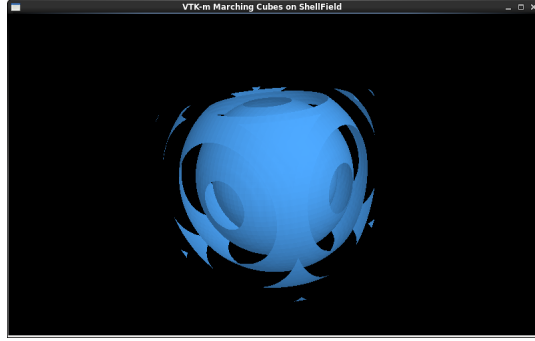


Fig. 3.1: A rendering of the dataset generated by the shells worklet. An isosurface at values of 0 is applied to make the structure visually apparent.

of work being done, as well as the lack of data reuse. Additionally, the extensive layers of inlined code and dependencies hinder vector register saturation. As described earlier, these problems combine to greatly limit any possible performance gains.

**3.1.2. Shell Worklet.** The shell worklet created a field, or a set of values for each point coordinate. The computation is conceptually simple, but involved more arithmetic and lookup compared to the incrementation worklet. For each point  $i$ , the distance from  $i$  to the center of the 3D grid was calculated. The  $\sin$  of the distance was taken, and the result was used as the final value. This results in spherical “shells” of values radiating from the center of the grid. Figure 3.1 shows the result of applying an isosurface filter (Marching Cubes) to the output of the worklet. The functor code is shown in Listing 1.

```
void operator()(const vtkm::Id &inputVertexId,
               vtkm::Float32 &outputVertexValue) const
{
    //calculate grid coordinates using vertex ID
    const vtkm::Id i = inputVertexId % width;
    const vtkm::Id j = (inputVertexId / width) % width;
    const vtkm::Id k = inputVertexId / layer;
    const vtkm::Float32 dist2 = (i-center)*(i-center)+
                                (j-center)*(j-center)+
                                (k-center)*(k-center);
    outputVertexValue = vtkm::Sin(vtkm::Sqrt(dist2)); //output
}
```

Listing 1: The shell worklet functor. Here, width is a member variable of the worklet. Note that there is low data reuse, resulting in high register pressure.

The shell field worklet did vectorize, with the vector registers being saturated at 8 values per operation. However, due to the low amount of data reuse, there is a high demand placed on the vector register. That is, the overhead of copying into the vector register is not adequately amortized over the number of operations being performed before the next load operation is called. This is also known high register pressure. Armed with the knowledge of data reuse and arithmetic density requirements gained from this and the previous test, two new experimental worklets were created.

**3.2. Vector Cross Product Worklet.** Computing vector cross products is a very common task, such as when computing surface normals and lighting. Storing the vector

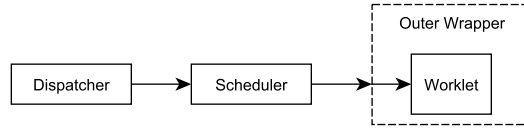


Fig. 3.2: A conceptual model of the tiled, two-layer worklet scheme. Developers can still write a field worklet and worklet functor as expected, and can remain oblivious to the wrapper around it that tiles the input for vectorization.

objects in an array for calculation is inherently an AoS solution. Thus this was a prime candidate for studying and understanding how a general optimization mechanism could be designed.

Several alternative versions were created to compare performance and guide the creation of a vectorized worklet. The functionality of each was the same. The inputs were two arrays of 3D vectors, either in `Vec` object form, or as three primitives. The output was a single array of 3D vectors containing the cross products, such that element  $i$  of the output array contained the cross product of the  $i$ th vectors in the input arrays.

First, a baseline worklet version was created to test as is and with the forced vectorization enabled. This is what developers would be expected to write. Next, a non-worklet AoS version was created. This was functionally identical to how a worklet would work. It allocated two arrays of `Vec` objects. A for loop then iterated over the arrays to compute the cross products. Then the equivalent SoA version was created. This used six allocated arrays for the x, y, z vector components and a for loop containing the cross product calculation.

Next, a hybrid AoSoA version was created, which introduced loop tiling. Initially, the vectors were stored in two arrays of `Vec` objects (AoS). For each iteration of the outer loop, six component arrays were allocated on the stack for a segment of the original array (SoA). These were filled with values, and a subsequent loop performed the cross product calculation. The results were then copied into the output array of `Vec` objects.

A slightly modified AoSoA version was created that introduced an extra step before computing the cross product; instead of directly using six primitive elements to compute the cross product, two `Vec` objects were created and passed to a function. There are two notable characteristics of this version. First, there is a copy from AoS to SoA and another copy back into AoS form, which was intended to allow the objects to contain values from contiguous memory locations. Second, by doing this additional copy, the input vectors are generalized into objects again, which allows the actual cross product computation to be isolated into a function more easily.

This then led to a tiled, two-layer worklet schema, wherein calling the worklet actually executed the outer loop, which transparently tiled the input and called the real worklet function repeatedly on only a segment of the input data. This preserves the initial purpose of a user-created worklet: a serial functor that operates on a small segment of the data. In this case, the developer need only write a cross product functor that takes two `Vec` objects as input and outputs another `Vec` object. The outer loop can then be a hidden templated wrapper that compiles to the correct types automatically. Figure 3.2 shows a conceptual model of the tiled, two-layer worklet scheme.

**3.3. Cell Average Worklet.** Cell averaging is a topology worklet: specifically, a point-to-cell worklet. Suppose a dataset contains a data field with values located at the points of a grid. To obtain data for the cells, the average of each cell's point values can be

used. The potential behavior of this operation differs greatly from that of a field worklet like vector cross product. Point and cell data for a grid are not necessarily linearly arranged in memory. Additionally, there is no guarantee that each cell has the same shape, meaning that the average calculation operation may not be exactly the same between successive calls.

VTK-m comes with a cell average worklet. It takes an array of point values and a cell set as input, and returns an array of cell values. It is designed to work on cell sets of varying shapes (sometimes called zoo cells). The cell average value is calculated with a loop whose bounds change based on the cell shape being operated upon. This generic implementation is necessary to allow for cell average calculation on any incoming cell set type.

However, if we restrict the input to explicitly defined cell sets, some insight can be gained. Explicit cell sets contain at minimum three arrays that define the grid's structure: cell shape, number of points for the cell, and point indices (also called connectivity). Any cell set can be converted to an explicit cell set. By representing the cellset structure data in arrays, the worklet can be converted to a field worklet. Now linearity can be guaranteed for accessing the worklet's input domain. A further restriction is to confine the allowed cell shapes to hexahedrons, which forces each cell to have eight points. If the bound for the loop calculating the average is not known at compile time—which is true of zoo cells due to varying shape—then vectorization will be difficult or impossible. By restricting the cell shapes, the viability of tiled worklets can be tested on topological operations.

The field cell average worklet iterates over the list of cells. Each iteration creates a set of eight arrays which correspond to the eight corners of the hexahedral cell. The array lengths are equal to the tile size. The inner loop then iterates over the *tileSize* point values. Each iteration *i* retrieves the *i*th value from the arrays, which are then passed to an inner worklet. The inner worklet simply iterates over the eight values to calculate the average.

Note that there are some issues with this structure. While it generally follows the tiled worklet schema, it does not have the data reuse and cache bonuses of a real field worklet. For example, after pulling eight values out to be averaged, those values are used only once when averaging to create a sum.

**4. Results.** Two different methods were used to measure performance. The first was inspection of performance, ability to vectorize, and vectorization effectiveness with Intel Advisor [7]. Advisor allows the developer to check if the compiler was able to vectorize the code, and to look at the generated assembly. It additionally measures time spent in vectorized loops and estimates gain and efficiency over the serial version. Finally, it shows details of vector register usage, and based on known patterns suggests improvements if vector utilization and efficiency are low.

The second method was to measure the wall time of the experiments using calls to the GNU C library [5] function `gettimeofday` [4], which allows for microsecond resolution. Wall time is in some senses a much more valuable measurement compared to the loop CPU time measured by Advisor because it reflects the time a user would wait for execution. The wall time of a process is heavily influenced by other processes running on a machine and I/O bandwidth. However, such influence in the measurement is desired as those same factors will exist when a user is running an application as well. To maintain usefulness in measurements, tests were run when as few other processes were running as possible, and tests were run 10 times to obtain a larger sample size for accuracy.

Advisor was helpful in finding reasons vectorization was not occurring, and for providing a proof of concept benchmark for testing. However, it was difficult to decipher how it calculated values of gain and efficiency. For example, reports of >8x gain could have been calculated by including memory efficiency. It was not clear, however, and the values for gain, vector length, and efficiency were used more as a cursory performance indicator. Wall time










Test	Gain	Vector Length	Average Wall Time (sec)	
Worklet (not vectorized)	0.00	1	21.59	
Worklet (vectorized)	0.47	2	12.15	
AoS for loop	1.89	8	7.04	
SoA for loop	8.10	8	5.91	
AoSoA for loop	9.80	8	10.19	
AoSoA for loop isolated function	6.10	8	7.45	
Two layer tiled worklet	7.47	8	9.00	

Fig. 4.1: Performance comparison of the various vector cross product tests. Gain and Vector Length are as reported by Advisor. Wall Time is as reported by `gettimeofday`.

reports show performance increases in a physical manner, which was much more intuitive. These reports were trusted more than Advisor results.

**4.1. Vector Cross Product Worklet Results.** All tests used two input arrays of  $2^{20}$  3D vectors and output one array of vectors of the same length. The baseline worklet was tested without any vectorization attempts to obtain a baseline wall time. The same worklet was then tested with the unilateral vectorization described in Section 3. This relied on the compiler automatically vectorizing the scheduler loop. For the remaining tests, this loop was preceded by an explicit pragma to prevent attempts at vectorization. This was done to allow the tiled loops to the inner worklets to vectorize properly. Additionally, any tiled loops used a tile size of 1024. This value was chosen as it is small enough to allow values to fit into a single cache line on most or all modern CPUs, while still remaining large enough to not incur a penalty for loading cache too often. Lastly, all tests were run with a single process utilizing a single core. Full results are shown in Figure 4.1.

The baseline worklet did vectorize and resulted in an approximately 44% wall time reduction. However the vectorization gain reported by Advisor was very low. This was thought to be caused by the indirect memory access of AoS representation. An additional issue was that the compiler vectorized the scheduler loop. The scheduler loop can be thought of as the outermost loop of the worklet, which is the incorrect level to vectorize. Ideally the innermost loop is vectorized, as it contains the simplest and lowest level instructions that can be directly converted into vector instructions. Vectorizing the outermost loop led to unexpected data dependencies and poor vectorization due to the number of nested loops and functions.

The same functionality was rewritten without the VTK-m worklet infrastructure for the AoS for loop test. This resulted in full vector register saturation, and increased performance. This result shows that by isolating the loop (i.e. no automatic vectorization of an outer loop), significant wall time reductions can be achieved even with AoS representation. The SoA representation for loop was then tested. By allowing direct access to primitive elements



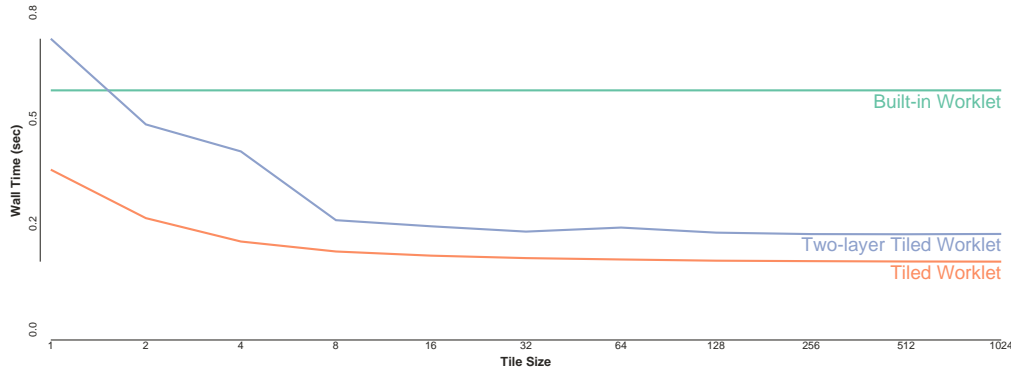


Fig. 4.2: Plot of wall time for the three cell average tests. Tile sizes greater than 8 resulted in a  $\leq 1\%$  additional speedup.

(which are reused in calculating the cross product), wall time is further decreased. An SoA representation is essentially the perfect design for vectorization, as it isolates the loop and allows for efficient loading from cache to vector units.

However, it is not appropriate to force all field worklets to be converted into for loops iterating over arrays. The hybrid AoSoA loop uses the speedup granted by SoA representation, while attempting to return to a worklet-style structure. For each iteration of the outer loop, six component arrays (three for each array of Vecs) of length 1024 were allocated, and the vector components were copied into them. For each iteration of the inner loop, the elements were passed to the cross product function. The memory allocation for each iteration introduced overhead, which reduced performance in comparison to the SoA for loop. However allocating and operating on these intermediate arrays is an easily templated structure. By introducing the tiled, two-layer structure to worklets, one can expect a wall time reduction over 50% like this test.

Since the cross product calculation was isolated into its own function, which mimics the original structure of the worklet, this two layer system was transformed into a wrapper around the worklet. Now, when the dispatcher and scheduler call the cross product worklet, they instead call an “Outer” worklet that takes in both AoS input arrays in their entirety. It then performs the intermediate array allocation, copying, and tiling, and calls the “Inner” worklet for each iteration of the inner loop. This results in dramatically improved gain, vector saturation, and efficiency over the automatically vectorized worklet. The two-layer tiled worklet is 240% faster than the baseline worklet and 135% faster than the vectorized worklet.

One caveat is that the vector cross product may be a special case. The mathematical operation of cross products involves multiplying vector components and summing/differencing the products. These sub-operations can be directly translated into Fused Multiply-Add (FMA) vector operations. Utilizing FMA reduces the number instructions even further than originally anticipated as it handles multiplication and addition at once [18]. Operations that do not translate to FMA instructions would still vectorize and should still see a significant performance increase, though possibly not to the extent of vector cross products.

**4.2. Cell Average Worklet Results.** The built-in cell average worklet was compared as a baseline against two worklets restricted to explicit cell sets. The two experimental worklets were functionally almost identical to the built in worklet. Both utilize tiling, however the first worklet did not have two layers; it simply calculated averages inside the loop. The second worklet did have two layers. Because the functionality of the new worklets is so close to the original—apart from being a field worklet—the tile size was varied from 1 to 1024. A tile size of 1 would make the both new worklets logically exactly the same as the original. Figure 4.2 shows results for the three tests.

In both cases, there is a performance increase that plateaus with tile sizes of 8 and higher. The single layer tiled worklet reduced wall time by approximately 33% at a tile size of just 1. This was due to it being a field worklet as opposed to a topology worklet. That is, the ability to directly access memory linearly over an array caused a significant speedup without using any tiling. At a tile size of 8, the single layer worklet reduced wall time by about 69%.

The two-layer tiled worklet actually performed 21% slower than the baseline with a tile size of 1. This was caused by the additional overhead of copying into a new array that is then passed to the inner worklet functor. Even though the functor is inlined by the compiler for vectorization, this additional step cost quite a bit in performance. Increasing the tile size alleviates this, however. At a tile size of 8, the two-layer tiled worklet performed approximately 55% faster than the baseline, albeit 44% slower than the single layer tiled worklet.

Increasing the tile size past 8 shows negligible performance improvements of  $\leq 1\%$ . This is thought to be caused by the averaging operation itself, which does not directly translate into a vector operation. Past a tile size of 8 there is a “wall” of arithmetic overhead that tiling cannot overcome.

Due to the difficulty involved, and number of restrictions required to achieve a small performance increase, it is unlikely that the same two-layer tiled worklet scheme will cleanly fit to topology worklets. If the restrictions placed were lifted, the high degree of variability in zoo cell sets would make it difficult to templatize the wrapper code.

**5. Conclusions.** In the push for exascale computing, vectorization plays a key role for the efficient utilization of x86 platforms and accelerators like the Xeon Phi [8]. Additionally, because it can be applied by the compiler, it is a performance boost that can be obtained “for free” given the correct computational structure.

Data filtering algorithms for visualization push some of the requirements for efficient vectorization. We were able to achieve a generic wrapper mechanism that can enable easier automatic vectorization for field worklets. The wrapper can remain invisible to the developer, who is still only required to write a serial functor operating on single elements. Additionally, the wrapper can be templatized, as it does not make assumptions about the input and output data types or the internal workings of the functor. Thus, VTK-m developers and applications based upon it can receive the benefit of higher performance x86-based code. For future work, we hope to extend this generic wrapper functionality to topology worklets, whose erratic memory accesses still present a challenge to vectorization.

## REFERENCES

- [1] J. AHRENS, B. GEVECI, C. LAW, C. HANSEN, AND C. JOHNSON, ParaView: An End-User Tool for Large-Data Visualization, 2005.
- [2] N. BELL AND J. HOBEROCK, Thrust: A Productivity-Oriented Library for CUDA, GPU Computing Gems Jade Edition, 2 (2011), pp. 359–371.

- [3] H. CHILDS, E. BRUGGER, B. WHITLOCK, J. MEREDITH, S. AHERN, D. PUGMIRE, K. BIAGAS, M. MILLER, C. HARRISON, G. H. WEBER, H. KRISHNAN, T. FOGAL, A. SANDERSON, C. GARTH, E. W. BETHEL, D. CAMP, O. RÜBEL, M. DURANT, J. M. FAVRE, AND P. NAVRÁTIL, VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data, in High Performance Visualization—Enabling Extreme-Scale Scientific Insight, Oct 2012, pp. 357–372.
- [4] FREE SOFTWARE FOUNDATION, High-Resolution Calendar. [[https://www.gnu.org/software/libc/manual/html\\_node/High\\_002dResolution-Calendar.html](https://www.gnu.org/software/libc/manual/html_node/High_002dResolution-Calendar.html); accessed 2016-09-06].
- [5] ———, The GNU C Library (glibc). [<https://www.gnu.org/software/libc/>; accessed 2016-09-06].
- [6] INTEL CORPORATION, Intel 64 and IA-32 Architectures Software Developer's Manuals. [<http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html>; accessed 2016-09-06].
- [7] ———, Intel Advisor 2017. [<https://software.intel.com/en-us/intel-advisor-xe>; accessed 2016-09-06].
- [8] ———, Introducing the Intel Xeon Phi Processor—Your Path to Deeper Insight. [<http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html>; accessed 2016-09-06].
- [9] ———, Overview: Intrinsics for Intel Advanced Vector Extensions 2 (Intel AVX2) Instructions. [<https://software.intel.com/en-us/node/582788>; accessed 2016-09-06].
- [10] LO, LI-TA AND SEWELL, CHRISTOPHER AND AHRENS, JAMES, PISTON: A Portable Cross-Platform Framework for Data-Parallel Visualization Operators, Eurographics Symposium on Parallel Graphics and Visualization, 2012.
- [11] LOS ALAMOS NATIONAL LABORATORY, Trinity: Advanced Technology System. [<http://www.lanl.gov/projects/trinity/>; accessed 2016-09-06].
- [12] J. S. MEREDITH, S. AHERN, D. PUGMIRE, AND R. SISNEROS, EAVL: The Extreme-Scale Analysis and Visualization Library, (2012).
- [13] K. MORELAND, U. AYACHIT, B. GEVECI, AND K.-L. MA, Dax Toolkit: A Proposed Framework for Data Analysis and Visualization at Extreme Scale, in Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on, IEEE, 2011, pp. 97–104.
- [14] K. MORELAND, C. SEWELL, W. USHER, L.-T. LO, J. MEREDITH, D. PUGMIRE, J. KRESS, H. SCHROOTS, K.-L. MA, H. CHILDS, ET AL., VTK-m: Accelerating the Visualization Toolkit for Massively Threaded Architectures, IEEE Computer Graphics and Applications, 36 (2016), pp. 48–58.
- [15] NATIONAL ENERGY RESEARCH SCIENTIFIC COMPUTING CENTER, Cori. [<http://www.nersc.gov/users/computational-systems/cori/>; accessed 2016-09-06].
- [16] OAK RIDGE NATIONAL LABORATORY, Introducing Titan, 2012. [<https://www.olcf.ornl.gov/titan/>; accessed 2016-09-06].
- [17] C. PHEATT, Intel Threading Building Blocks, Journal of Computing Sciences in Colleges, 23 (2008), pp. 298–298.
- [18] E. QUINNELL, E. E. SWARTZLANDER, AND C. LEMONDS, Floating-Point Fused Multiply-Add Architectures, in 2007 Conference Record of the Forty-First Asilomar Conference on Signals, Systems and Computers, IEEE, 2007, pp. 331–337.
- [19] W. SCHROEDER, K. MARTIN, AND B. LORENSEN, *The Visualization Toolkit: An Object-oriented Approach to 3D Graphics*, Kitware, 2006.

## EVALUATION OF AREA OF OPTIMIZATION IN HYDRODYNAMIC CODE WITH PERFMINER

TANNER JUEDEMAN\* AND JEANINE COOK†

**Abstract.** As the rates by which processing speeds and architectural innovations in next-generation computing systems begin to plateau, the need for alternative means of ensuring code speedup has become apparent. Utilities capable of describing low-level application behavior or identifying performance-defining trends within code bases allow developers a means by which to identify bottlenecks within their applications, optimize for these bottlenecks, and produce applications of elevated efficiency. In this work we verify findings established by the application performance modeling and profiling tool, MuMMI, via the statistical analysis of event counts collected using a tool of a similar nature, PerfMiner, on a common scientific benchmarking application, LULESH.

**1. Introduction.** The breakdown of Moore and Dennard scaling in recent history has shifted significant focus away from architecting increasingly complex and compact computer processors, effectively bringing code and architecture optimization into the spotlight of the computing community. Physical limitations inherent within processor designs, such as minimum attainable transistor size or maximum allowable energy dissipation, prevent the rapid advancement of computer processing speeds to continue consistently with the observable trend of the past few decades. While code optimization techniques in response to specific resource bottlenecks are already existent and prevalent, means by which to effectively and reliably identify these bottlenecks within complex code bases can still prove to be difficult, leaving a notable portion of optimization up to the intuition of the developer. As a product of this, a clear need for effective application profiling and optimization tools exists within our current age of computing.

**2. MuMMI.** The Multiple Metrics Modeling Infrastructure (MuMMI) [7] is an innovative utility currently undergoing development at Texas A&M University that allows users to measure, model, and predict the performance and power consumption changes made to an application's code based on utility-identified optimization areas. Identification of notable areas to be optimized is done by correlating application run times against hardware event counts through the use of a variety of statistical models. Highly-correlated events in this instance can generally be considered to be a component of run time, and substantial optimization can frequently be made by optimizing the processes responsible for generating such events [7].

MuMMI goes a step further by attempting to model application performance characteristics using previously identified events of interest. By fitting a model to application performance, adjustments to event counters can be simulated and expected speedup from optimization determined. This process can be used to identify which component of runtime have the potential to provide the most benefit to the user when optimized, or if optimization steps are even worth the effort if expected speedup is insignificant.

**3. PerfMiner.** PerfMiner [5] is a thread-level hardware performance counter collection utility capable of retrieving the counts of numerous hardware events over a single application execution by means of modeling counter behavior through sample-based multiplexing. The utility is scalable and able to evaluate hardware counts of runs conducted on a single core upwards to many thousands of cores with no notable run time dilation. Performance metrics are retrievable at a thread level using perf [4], a Linux performance counter profiler, or via

---

\*New Mexico State University, [juedeman@nmsu.edu](mailto:juedeman@nmsu.edu)

†Sandia National Laboratories, [jeacock@sandia.gov](mailto:jeacock@sandia.gov)

the Performance Application Programming Interface (PAPI) [1]. Choice of collection tool is dictated by user selection<sup>1</sup>.

PerfMiner’s multiplexing functionality allows the counts of numerous events to be collected within a single run. Typically, a given architecture will possess a relatively small number (4-8) of user-accessible performance counters, which can only record one event at a time. In the past, this has led users to rerun test applications while recording differing subsets of events in order to retrieve a representative picture of the behavior of an application. This process proves to be problematic when, for example, event lists of considerable length are desired, and single run execution times are of a significant duration. In many conventional cases, collection of such data could take upwards of weeks. However, PerfMiner remedies this problem by modeling the behavior of event counters as it multiplexes through an event list, effectively allowing for the collection of nearly all events present on an architecture with minimal error over the course of a single run. This is an appealing feature, as MuMMI utilizes an event list consisting of thirty-two unique PAPI events when profiling an application, and the time required to execute collection runs of an application at various configurations to validate just the 32 counters associated with MuMMI’s findings would have proven to be a time-intensive endeavor.

**4. LULESH.** The Livermore Unstructured Lagrange Explicit Shock Hydrodynamics mini-application (LULESH) is one of five challenge problems posed by Defense Advanced Research Projects Agency’s (DARPA) Ubiquitous High Performance Computing program (UHP), a multi-million dollar initiative aimed at stimulating the development of promising exa-scale architectures. Even though an execution of LULESH analytically solves a relatively basic Sedov blast problem, and can therefore be considered a simple application, LULESH is regarded as representative of the typical algorithms and data flow that exists within more expansive scientific applications. In this, LULESH proves to be useful for benchmarking purposes as well as for testing optimization toolkits [2].

Since the inception of DARPA’s initiative, LULESH has become a widely-studied proxy in the development of future architectures as well as seen use in the area of evaluating HPC application optimization techniques. Representative of larger hydrodynamic codes but packaged as a mini-app, LULESH is capable of being quickly integrated into differing architectures for use as an optimization test case [3]. This ease-of-integration allows for resource-intensive application runs to be quickly conducted on diverse architectures, which can then be evaluated by various optimization applications. Knowledgeable optimization of the original hydrodynamic code can be made from the findings of these evaluative applications. LULESH’s scalable run time footprint allows for unoptimized and optimized runs to be tested and compared at shorter execution times during preliminary testing, which can then be scaled to more significant test sizes once notable optimizable areas are identified. This expedited means by which optimization changes can be evaluated makes LULESH an appealing test case for the evaluation of performance monitoring and optimization software.

**5. Shepard.** As one of the multiple advanced architecture test beds housed at Sandia National Laboratories, Shepard exists as a thirty-six node architecture testbed featuring two Intel Xeon E5-2698 v3 processors per node and intended for use in the exploration of pre-exascale systems as a part of the National Nuclear Security Administrations (NNSA) Advanced Simulation and Computing (ASC) project. For the purpose of this evaluation, previous LULESH optimization findings gathered from Shepard using MuMMI will be verified with PerfMiner runs on the same architecture. This adherence to a single testbed is

<sup>1</sup>PAPI event descriptions can be found at [https://icl.cs.utk.edu/projects/papi/wiki/PAPIC:PAPI\\_presets.3](https://icl.cs.utk.edu/projects/papi/wiki/PAPIC:PAPI_presets.3), while perf event descriptions can be found at <http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-manual-325462.pdf>

established in order to enforce consistent application environment conditions between the differing experiment runs, and to minimize architecture-induced variations in experiment results.

**6. Methodology of Evaluation.** MuMMI in part identifies areas of optimization within a given code by correlating the run times of a set of executions with the hardware counters of predetermined events during those runs. Event counters that are highly correlated to an application’s execution times are hypothesized to be substantial components of, or potentially responsible for, the overall execution time of the application, and therefore rewriting the application’s code to minimize the counts of these highly-correlated counters could lead to improved run times. By evaluating a set of runs of single-node configurations in this way, one might use this method for identifying areas of optimization that will improve the run time of a configuration that is specified for single-node executions. By evaluating a set of runs from various node configurations, like those used in scaling studies, one might be able to identify areas of optimization that could potentially improve the scaling or distributed performance of the application.

Previous MuMMI experiments conducted on Shepard that evaluated thirty-two PAPI counters identified five events of particular interest: PAPI\_TLB\_IM, PAPI\_L3\_TCM, PAPI\_L1\_DCM, PAPI\_CA\_CLN, and PAPI\_BR\_TKN [6]. While PerfMiner possesses the capability of recording PAPI events, to record the same set of events using PerfMiner would essentially clone the previous experiment. Perf events were recorded instead as a means of approaching these findings from a different method.

Table 6.1: PAPI-to-Perf Conversion Table

PAPI Event	Perf Equivalent
PAPITOT_CYC	CPU_CLK_THREAD_UNHALTED:THREAD_P
PAPITOT_INS	INST_RETIRED:ANY_P
PAPIL1_TCM	L1D:REPLACEMENT
PAPIL2_TCM	LLC:REFERENCES
PAPIL3_TCM	LLC:MISSES
PAPICA_SHR	OFFCORE_REQUESTS:ALL_DATA_RD
PAPIL2_LDM	L2_RQSTS:DEMAND_DATA_RD_MISS
PAPIL2_STM	L2_RQSTS:DEMAND_RFO_MISS
PAPIBR_CN	BR_INST_RETIRED:CONDITIONAL
PAPIBR_TKN	BR_INST_RETIRED:CONDITIONAL, BR_INST_RETIRED:NOT_TAKEN
PAPICA_CLN	OFFCORE_REQUESTS:DEMAND_RFO
PAPICA_ITV	OFFCORE_RESPONSE_0:SNP_FWD
PAPIBR_NTK	BR_INST_RETIRED:NOT_TAKEN
PAPIBR_MSP	BR_MISP_RETIRED:CONDITIONAL
PAPIRES_STL	RESOURCE_STALLS:ANY
PAPIL2_TCA	L2_RQSTS:ALL_DEMAND_REFERENCES
PAPIL1_LDM	L2_TRANS:DEMAND_DATA_RD
PAPIL1_STM	L2_TRANS:L1D_WB
PAPIL2_TCW	L2_TRANS:RFO
PAPIL2_DCA	L2_RQSTS:ALL_DEMAND_REFERENCES
PAPIL2_DCR	L2_RQSTS:ALL_DEMAND_DATA_RD
PAPIL2_DCW	L2_TRANS:RFO
PAPIL1_ICM	L2_RQSTS:ALL_CODE_RD
PAPIBR_INS	BR_INST_RETIRED:ALL_BRANCHES
PAPIL1_DCM	L1D:REPLACEMENT
PAPIL2_ICA	L2_RQSTS:ALL_CODE_RD
PAPITLB_DM	DTLB_LOAD_MISSES:MISS_CAUSES_A_WALK, DTLB_STORE_MISSES:MISS_CAUSES_A_WALK
PAPITLB_IM	ITLB_MISSES:MISS_CAUSES_A_WALK
PAPIL2_DCM	LLC:REFERENCES, L2_RQSTS:CODE_RD_MISS
PAPIL2_ICM	L2_RQSTS:CODE_RD_MISS
PAPILLD_INS	MEM_UOPS_RETIRED:ALL_LOADS
PAPILSR_INS	MEM_UOPS_RETIRED:ALL_STORES

The use of perf-native events allows for the holistic evaluation of the code, similarly to MuMMI’s intent, does not mandate a need for direct source code manipulation, and only

requires the construction of a list of events to be recorded. The conversion of the thirty-two PAPI events used by MuMMI to perf natives acceptable by PerfMiner is outlined in Table 6.1. Over the course of this evaluation, a perf event that is directly equivalent to PAPI.L3.TCM could not be readily identified, and will therefore not be regarded in this study. In addition to these events, approximately 120 additional events were recorded by PerfMiner and processed in this evaluation as a means of potentially identifying additional highly correlated events.

As a means of identifying performance counters of interest, 152 events recorded across 21 uniquely-configured scaling runs were correlated against the execution times of the runs they were collected from. The Spearman's rank correlation was chosen for this evaluation as the coefficient output descriptive of a dataset's fit to a monotonic relationship, or how consistently a dependent variable increases or decreases as an independent variable increases. More specifically, this coefficient, referred to as rho, is a nonparametric measure of rank correlation, or the statistical dependence between two variables of interest. A correlation's p-value is the likelihood of the data existing and producing the observed rho value, assuming that the null hypothesis of the correlation is true, that there is no monotonic relationship. For this statistical model, observed p-values of less than 0.05 are generally regarded as statistically significant. This choice in correlation model is in opposition to the use of a Pearson correlation, which outputs a coefficient that is descriptive of a dataset's fit to a linear model. While linear resource-time relationships are desirable findings in scaling studies, in practice these relationships do not tend to exist in unoptimized distributed codes. Additionally, Spearman correlations are regarded as less susceptible to distortion from outliers than Pearson correlations, resulting in a relationship description that is more representative of an application's behavior. Examples of weak and strong Spearman correlations identified in this evaluation are given in Figures 6.1 and 6.2.

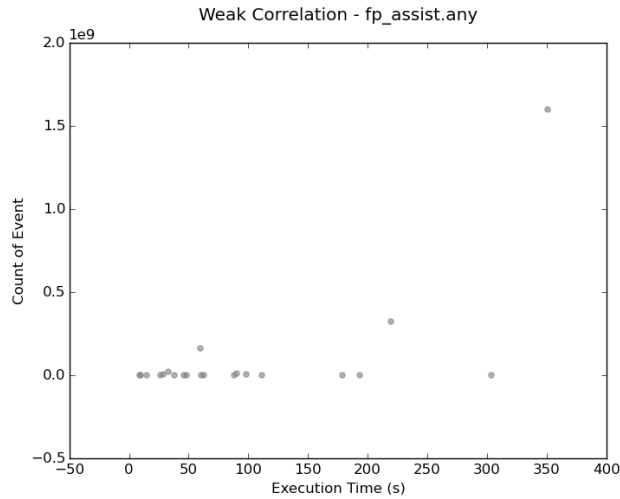


Fig. 6.1: Example of a weak Spearman correlation (no consistent monotonic relationship.)

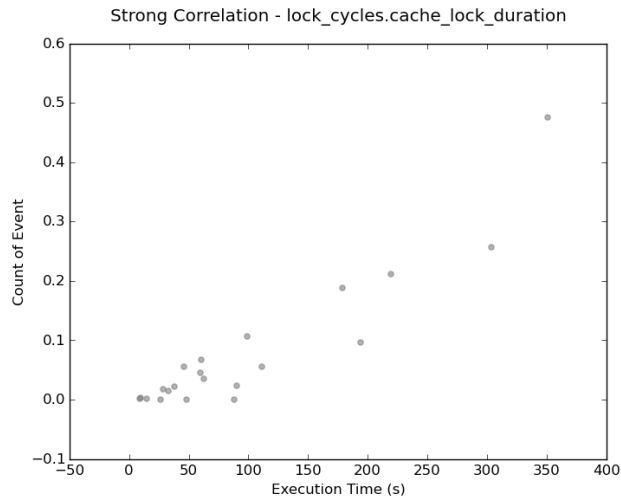


Fig. 6.2: Example of a stronger Spearman correlation (increasing trend, observable monotonic relationship.)

By representing each event counter output as a ratio of that count to the total number of execution cycles in the run from which it was collected, one can normalize the data set in a manner that reduces the impact of counter variation. While the counter values and total cycles may vary to a minor degree from run to run, the ratio of these counters to cycles tends to stay consistent, under the assumption that as cycles increase, the behavior of the program stays consistent. By evaluating the ratio of these events as opposed to their raw counter values, stronger correlations can be seen in the data and a more representative image of the scaling behavior of the application can be observed. Figures 6.3 and 6.4 illustrate a representative transformation that occurs when one of the counters in this evaluation's data set is normalized.



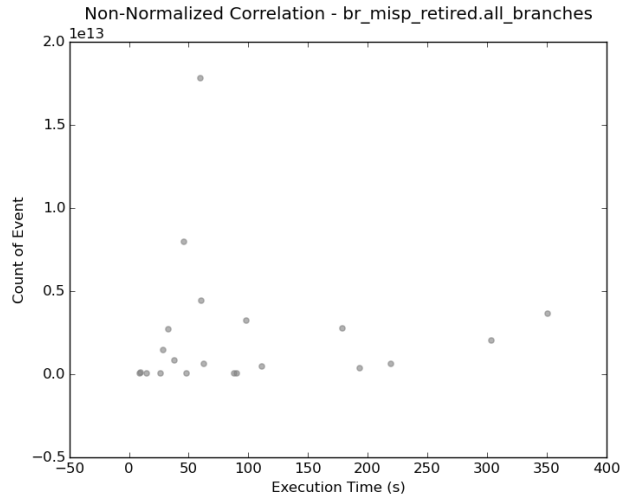


Fig. 6.3: A distribution of scaling values prior to normalization.

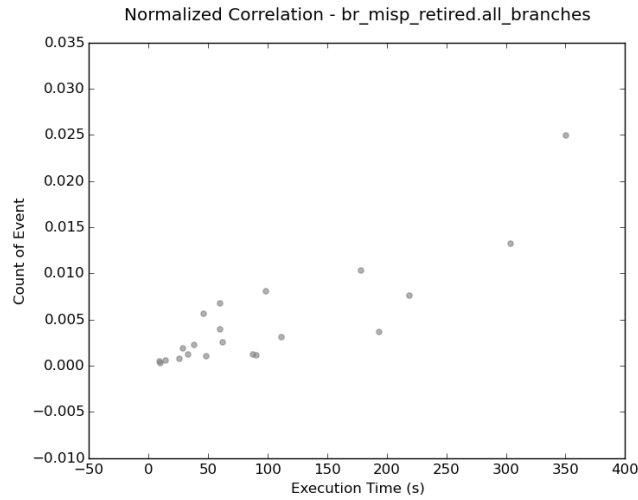


Fig. 6.4: A distribution of scaling values after normalization.

**7. Evaluation.** The top ten events highly correlated to run time in this evaluation and their corresponding rho and p-values are given in Table 7.1. From this data, one could conclude that the execution time of this application is primarily dominated by time spent in node/CPU synchronization, evident by highly correlated lock<sup>2</sup>, stall<sup>3</sup>, and recovery<sup>4</sup> cycle event counters, that could potentially have been caused by high instances of branch

<sup>2</sup> Lock cycles: (lock\_cycles.cache\_lock.duration), cycles stalled while L1D was locked.

<sup>3</sup> Stall cycles: (ild\_stall.lcp), cycles where the decoder is stalled on an instruction with a length changing prefix.

<sup>4</sup> Recovery cycles: The number of cycles spent waiting for a recovery after an event such as a memory nuke or misprediction.

misprediction. A situation such as this could be indicative of a poorly distributed task load within the application, or a task load that does not distribute well with scaling.

Table 7.1: Top Results of Spearman Correlation of 152 Events

Perf Event	Rho	P-Value
br_misp_retired.all_branches	0.8077922078	9.50261493034E-006
lock_cycles.cache_lock_duration	0.7948051948	1.67476895284E-005
offcore_requests.all_data_rd	0.7883116883	2.19089193045E-005
offcore_requests.demand_data_rd	0.7857142857	0.000024332
ild_stall.iq_full	0.7844155844	2.56285823777E-005
int_misc.recovery_cycles	0.7831168831	2.6984919006E-005
offcore_requests.demand_rfo	0.7831168831	2.6984919006E-005
offcore_response.pf_l2_data_rd.llc_hit.any_response	0.7779220779	3.3054555589E-005
idq.all_mite_cycles.any_uops	0.7753246753	3.6511080075E-005
icache.hit	0.774025974	3.83540889513E-005

Table 7.2 lists the correlation values of the events of interest previously identified by MuMMI based on event counts from their corresponding perf equivalent. A Spearman correlation of these events show a moderate correlation between the event and the run time of the application for events PAPI\_TLB\_IM, PAPI\_L1\_DCM, PAPI\_CA\_CLN, and PAPI\_BR\_TKN, with small associated p-values. This would imply that while these associations are not exceptionally strong, the associations may be non-incidental and can be regarded as legitimate areas to consider for optimization.

Table 7.2: PAPI Events of Interest, as Indicated by MuMMI

PAPI Event	Rho	P-Value
PAPI_TLB_IM	0.5636363636	0.0077925004
PAPI_L1_DCM	0.725974026	0.0001948689
PAPI_CA_CLN	0.7831168831	2.6984919006E-005
PAPI_BR_TKN	0.6584415584	0.0011733517

It is important to state that the highest correlating events found by collection with PerfMiner are not recorded by MuMMI in typical runs. By recording more events than the thirty-two PAPI presets used by MuMMI, events featuring greater correlation to run time were found in addition to those identified by MuMMI. In this, PerfMiner’s findings expand on these previous findings and demonstrate the benefit of more substantial event lists.

**8. Conclusions.** Analysis discussed within this evaluation identifies and verifies those areas of optimizable interest that were previously found by the MuMMI tool. In addition to these areas of interest, additional areas were identified from a more substantial collection list used by PerfMiner. As code optimization techniques in response to specific resource bottlenecks such as those identified in this evaluation are already existent and prevalent, optimization of these performance hampering areas is possible. Justification of the utility that evaluative applications such as MuMMI and PerfMiner possess come from the ease-of-identification possible with these applications.

## REFERENCES

- [1] J. DONGARRA, K. LONDON, S. MOORE, P. MUCCI, AND D. TERPSTRA, Using PAPI for Hardware Performance Monitoring on Linux Systems, Conference on LinuxClusters: The HPC Revolution, (2001).
- [2] R. HORNING, J. KEASLER, AND M. GOKHALE, Hydrodynamics Challenge Problem, Technical Report LLNL-TR-490254, Lawrence Livermore National Laboratory.
- [3] I. KARLIN, J. KEASLER, AND R. NEELY, LULESH 2.0 Updates and Changes, Technical Report LLNL-TR-641973, Lawrence Livermore National Laboratory.
- [4] A. MELO, The New Linux 'perf' Tools, (2010).
- [5] P. MUCCI, D. AHLIN, J. DANIELSSON, P. EKMAN, AND L. MALINOWSKI, PerfMiner: Cluster-Wide Collection, Storage and Presentation of Application Level Hardware Performance Data, Proceedings of the 11th international Euro-Par conference on Parallel Processing, (2005).
- [6] X. WU, Performance Modeling and Comparison of Lulesh Codes (Openmp-minimal and Openmp-optimized) on SNL Shepard and Cooper, Forthcoming, (2016).
- [7] X. WU, C. LIVELY, V. TAYLOR, H. CHANG, C. SU, K. CAMERON, S. MOORE, D. TERPSTRA, AND V. WEAVER, MuMMI: Multiple Metrics Modeling Infrastructure, The 14th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Network and Parallel/Distributed Computing, (2013).

## TRILINOS IMPROVEMENTS: IFPACK2 BLOCK RELAXATION PERFORMANCE AND YAML PARAMETER LISTS

BRIAN M. KELLEY<sup>¶</sup>, MARK F. HOEMMEN<sup>||</sup>, AND CHRIS M. SIEFERT<sup>\*\*</sup>

**Abstract.** Block relaxation is a preconditioning algorithm that is effective for finite element analysis problems with natural clusters of vertices [11]. Block relaxation excels in ice sheet simulation problems, because ice sheet meshes tend to be more dense vertically than horizontally [15]. The PISCEES project [14] performs large scale ice sheet simulations using Albany codes [13], which rely on Trilinos [1] for linear solvers and preconditioners.

This paper describes the process of refactoring and optimizing the block relaxation implementation in the Ifpack2 package [11] of Trilinos. Ifpack2 block relaxation is now up to 6.94 times faster than before. This improvement will benefit PISCEES.

Additionally, this paper discusses the addition of the YAML language [8] as an alternative to XML [9] for Trilinos parameter lists [6]. Parameter lists are used for all runtime configuration of Trilinos packages. YAML can express the same parameter lists as XML, but in a much cleaner and simpler format. This will save time for Trilinos users.

**1. Introduction.** Block relaxation is a preconditioning algorithm that groups tightly-coupled clusters of vertices into “blocks”. It sweeps over blocks in either block Jacobi or block Gauss-Seidel fashion, solving each block’s linear system individually. Ideally, vertices are strongly coupled with the other vertices in their blocks, and weakly coupled with vertices outside their blocks [12] [11].

The PISCEES project (Predicting Ice Sheet Climate & Evolution at Extreme Scales) [14] uses Albany [13] and Trilinos [1] codes to perform simulations of the ice sheets of Antarctica and Greenland. The goal of the project is to accurately predict sea level rise in the 21<sup>st</sup> century from the melting of land ice [14].

3D ice sheet meshes tend to be sparse in the horizontal directions but dense in the vertical direction. Algebraic multigrid (AMG) preconditioners [16] are used because they are efficient and scalable, but in anisotropic problems they suffer from bad convergence rates [15]. To resolve this problem, AMG uses block relaxation as an internal line smoother.

Block relaxation employs a “line partitioner” to construct blocks such that the vertices in each block approximately follow a straight line [11]. Because vertices are densely packed in the vertical direction, the line partitioner selects vertical groups of vertices as blocks.

Block relaxation’s implementation in Ifpack2 [11], while numerically correct, was much slower than the equivalent implementation in the older ML package [5]. ML is used as a performance benchmark throughout the paper. Ifpack2’s block relaxation was copied naively from the Ifpack package and was never revisited for profiling and optimization. It was suspected that the main performance issue was bad cache performance, since each block matrix was allocated separately. Profiling the code in Intel VTune [2] revealed that most of the running time was spent managing reference counted pointers and memory allocation. This paper describes the process of refactoring Ifpack2’s block relaxation to mitigate these sources of overhead and facilitate future improvements.

Another Trilinos enhancement presented in this paper is `ParameterList` support for the YAML language [8]. The `ParameterList` class in the Teuchos package is used as a common format to express input parameters for all Trilinos packages and many libraries that depend on Trilinos, such as Albany and Drekar [10]. `ParameterLists` are either populated directly in C++ or read from XML files [6]. Unfortunately, XML files require

<sup>¶</sup>Texas A&M University, kelb150@tamu.edu

<sup>||</sup>Sandia National Laboratories, mhoemme@sandia.gov

<sup>\*\*</sup>Sandia National Laboratories, csiefer@sandia.gov

lots of boilerplate syntax. YAML is a data serialization language that can express the same hierarchical dictionaries as XML with a much cleaner syntax. This will save time for Trilinos users and developers.

Support for reading `ParameterLists` from YAML and for automatically converting XML files to YAML is now available in the Teuchos package [6] alongside the original XML utilities, so that Trilinos users can use it in real-world problems.

**2. Block Relaxation Container Refactoring.** The initial goal of the Ifpack2 block relaxation improvements was to refactor the `Container` classes to facilitate future performance improvements. There are 4 subclasses of `Container`, each with a different underlying matrix representation for the block (vertex cluster): `TriDiContainer`, `BandedContainer`, `SparseContainer` and `DenseContainer`. The first refactoring task was to modify these classes so that a single instance held all the block matrices, instead of having one `Container` per block. Also, the implementations of Jacobi, Gauss-Seidel and SGS (symmetric Gauss-Seidel) [12] were moved from the `BlockRelaxation` class to `Container`.

During this round of refactoring, the memory layout of the block matrices was made more cache-friendly. Previously, each block was allocated separately. Instead, `Container` now allocates a single scalar buffer large enough to hold all block matrices. This means that iterating over blocks is now a predictable and sequential memory access pattern.

Figure 2.1 diagrams the memory layout changes: `BlockRelaxation` owns one `Container` instead of several, multiple separately allocated matrix objects are grouped into one contiguous array, and all matrix scalars are grouped into one array.

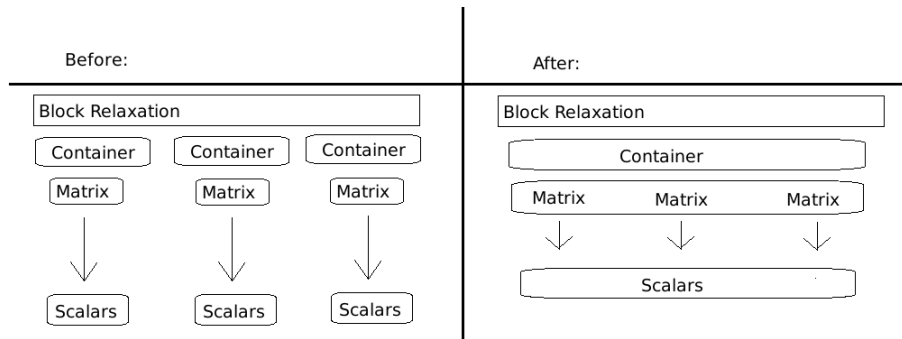


Fig. 2.1: Container memory layout: before and after refactoring

It is not possible to give the `SparseContainer` the same memory layout improvements, because it uses the advanced `Tpetra::CrsMatrix` class as the underlying matrix type. This is acceptable because the sparse container is only used for problems with very large blocks, where per-block overhead is not a major concern.

After the `Container` refactoring, profiling tools were used to find the slow sections of block relaxation. A new unit test was created in Ifpack2 to run a controlled test of performance. In this test `BlockRelaxation::apply` is called with a 50,000 vertex problem. Each block contains between 4 and 50 vertices, and the blocks are nonoverlapping. Five sweeps of Gauss-Seidel are performed in each trial. The `TriDi` container was arbitrarily chosen for performance testing because earlier tests showed that there was no significant performance difference among the container types. Ideally, the LAPACK `dgttrs` call (solve with tridiagonal matrix and LU factorization) [4] in each `Container::apply` dominates

the running time. In reality, the test program spent most of its time in the internals of `Kokkos::SharedAllocationRecord` (used by `Kokkos::View` [3] when making shallow copies). Memory allocation was the next offender, followed by the `std::string` constructor. Figure 2.2 shows the Intel VTune results.

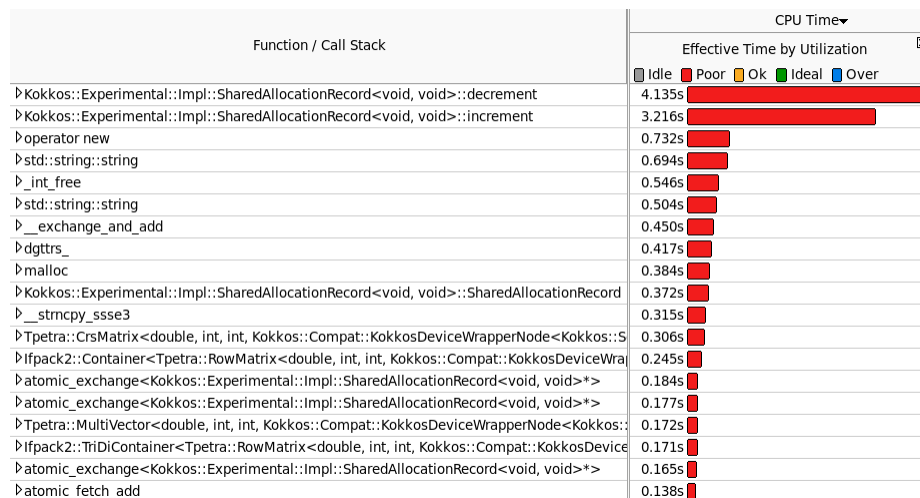


Fig. 2.2: Profiling of BlockRelaxation after first refactor (200 trials in 17.1 seconds)

**3. Replacing `Tpetra::MultiVector` with `Kokkos::View`.** The next step in the project was to investigate why block relaxation was spending the majority of its time inside `Kokkos::View` internals. All of `Container`'s computations were performed with `Tpetra::MultiVector` objects. The `MultiVector` uses `Kokkos::Views` to handle the layout and allocation of their memory. Since `MultiVector` does not allow direct indexing of its data, block relaxation created temporary nonowning, local views of the data. Normally, getting a view from a `MultiVector` is a fairly cheap operation, but block relaxation used it in the inner loops of its Jacobi, Gauss-Seidel and SGS functions. Every time a view is requested, the `Kokkos::View` inside the `MultiVector` is forced to update its reference counts, check that it is available from host memory, and mark itself as modified. These functions contributed a huge amount of overhead when run millions of times.

In order to mitigate this source of overhead, the underlying `Kokkos::View` is now used directly in the computations, because `Kokkos::View` allows direct element indexing. Rather than pass a `MultiVector` to the computational methods, the `Kokkos::View` is extracted and synchronized to the host memory space only once. This is necessary when running on heterogeneous machines, i.e. GPUs or Xeon Phi. Then, all computations are done with that `View`. The internals of `Container` and the four `Container` subclasses were modified to use `Kokkos::View` in both the iterative methods and the `apply` functions.

An added benefit of using `Kokkos::View` is that it is no longer necessary to create a `Tpetra::Map` for each block. Previously, these `Maps` were used to create local `MultiVector` objects in `Container::apply`. The distributed capabilities of the `MultiVector` were never needed by block relaxation, because blocks are always local to a single processor (a convention used in `Ifpack2`).

With the overhead of `Tpetra::MultiVector` and `Map` eliminated, the largest sources of overhead are gone. Profiling the code confirms that `DoGaussSeidel` and `LAPACK`'s

Block Size	Time per Apply (sec)			Speedup Factor	
	New Ifpack2	Old Ifpack2	ML	Old/New	New/ML
4	0.004197	0.029137	0.000414	6.94	10.1
5	0.003606	0.023142	0.000457	6.42	7.89
8	0.002660	0.014971	0.000572	5.63	4.65
10	0.002419	0.011906	0.000665	4.92	3.64
20	0.001806	0.006542	0.001088	3.62	1.66
50	0.001498	0.003326	0.002553	2.22	0.59

Table 3.1: Performance data after refactoring `BlockRelaxation`. Old/New is the speedup factor obtained from refactoring, and New/ML is the speedup factor of ML compared to the refactored (new) Ifpack2.

`dgttrs` are now dominating the running time (see Figure 3.1). 1500 trials are completed in 24.1 seconds (note that the results in Figure 2.2 are for only 200 trials). The first row is `DoGaussSeidel`.

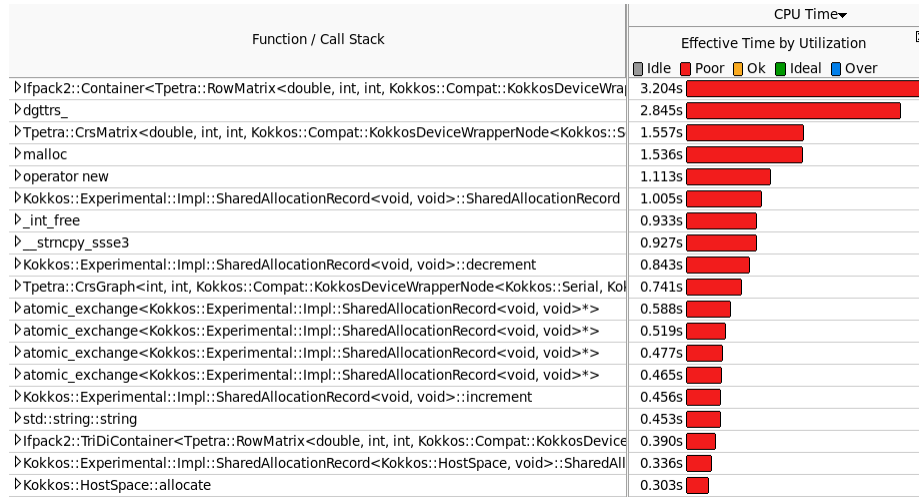


Fig. 3.1: Profiling of `BlockRelaxation` using `Kokkos::View` for computations (1500 trials in 24.1 seconds). The 1<sup>st</sup> row is `DoGaussSeidel`, and the 3<sup>rd</sup> row is `CrsMatrix::getLocalRowView`.

The elimination of `MultiVector` in the `Container` computations caused a 6.94x improvement in running time for the test with 12,500 blocks of 4 rows each (relative to the old Ifpack2). However, in the same problem, ML is still 10.1x faster. Table 3.1 contains all data from the final performance test. Time per apply call is given for the newly refactored Ifpack2, Ifpack2 before the refactoring, and ML. As block size increases, both new and old Ifpack2 speed up because per-block overhead is very high. ML drops below the new Ifpack2 with large blocks because its custom solver implementation is slower than LAPACK. Figure 3.2 is a plot of these figures. The table also contains the speedup factor comparing new vs. old Ifpack2, and new Ifpack2 vs. ML.

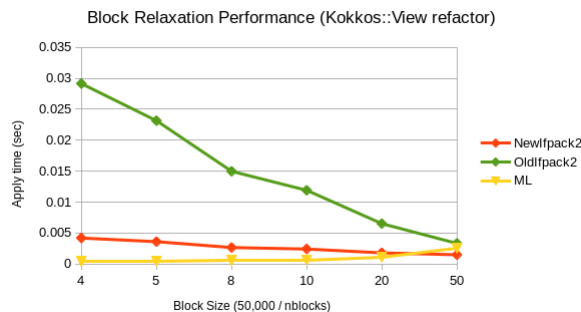


Fig. 3.2: BlockRelaxation performance test with computations using `Kokkos::View`

**4. Future Block Relaxation Work.** The final Intel VTune results for block relaxation suggest that overhead is still very high compared to ML. Input matrix access, memory allocations, and `Kokkos::SharedAllocationRecord` functions are still collectively taking around twice as long as the GaussSeidel and LAPACK solve put together.

In the future, thread-level parallelism will be added to `Container::DoJacobi`. Non-overlapping Jacobi is simple to parallelize because the block-vector products can be copied into the solution vector independently.

**5. YAML Parameter Lists.** The XML `Teuchos::ParameterList` format is heavily used by all Trilinos packages and many Trilinos users. XML is widely understood and capable of expressing all the data types needed for input decks. However, XML's verbose syntax makes it difficult for humans to read and write. The YAML serialization language [8] provides a more user friendly alternative to XML. YAML has a simple and intuitive syntax. The motivation for YAML parameter list support in Trilinos was to save the time of both users and developers when reading and writing input decks.

`yaml-cpp` [7] is the third-party C++ library for reading and writing YAML files. It reads files into a tree structure. Copying the data and structure from the tree to a `Teuchos::ParameterList` is simple and fast. Some unit tests were created that read in equivalent XML and YAML parameter lists and verify that their contents are the same. Here is one pair of files (XML followed by equivalent YAML):



```

<ParameterList name="MueLu">
  <Parameter name="coarse: max size" type="int" value="1000"/>
  <Parameter name="multigrid algorithm" type="string" value="unsmoothed"/>
  <Parameter name="smoother: type" type="string" value="RELAXATION"/>
  <ParameterList name="smoother: params">
    <Parameter name="relaxation: type" type="string" value="Jacobi"/>
    <Parameter name="relaxation: sweeps" type="int" value="5"/>
    <Parameter name="relaxation: damping factor" type="double" value="0.9"/>
  </ParameterList>
  <Parameter name="aggregation: type" type="string" value="uncoupled"/>
  <Parameter name="aggregation: drop tol" type="double" value="1e-7"/>
  <Parameter name="aggregation: export visualization data" type="bool" value="true"/>
</ParameterList>
%YAML 1.1
---
MueLu:
  "coarse: max size": 1000
  "multigrid algorithm": unsmoothed
  "smoother: type": RELAXATION
  "smoother: params":
    "relaxation: type": Jacobi
    "relaxation: sweeps": 5
    "relaxation: damping factor": 0.9
  "aggregation: type": uncoupled
  "aggregation: drop tol": 1e-7
  "aggregation: export visualization data": true
...

```

The improvement in readability is obvious. The same information is being expressed more efficiently: the XML file is 755 characters, and the YAML file is only 352 characters (a 53% reduction in size). YAML focuses on presenting the information in the simplest possible way, with minimal syntactic overhead.

The main benefit of YAML is type inference. In XML, the type must be explicit for every parameter. For example:

```

<!-- In an xml parameter list...-->
<!-- myString is a string-->
<Parameter name="myString" type="string" value="true"/>
<!-- myBool is a bool-->
<Parameter name="myBool" type="bool" value="true"/>

```

In YAML, type is never explicit. Every value is stored as a string. The YAML parameter list reader determines type by attempting conversion to int, double and bool, and using the most specific conversion that succeeds. For example, “5” can be parsed as an int or a double, but since the int values are a subset of the double values, the value is interpreted as an int. “5.0” is interpreted as a double.

While type inference is convenient in most situations, it sometimes leads to undesired behavior. For example, any string value that could be interpreted as a boolean (i.e. “true”) is assumed to be a boolean, and there is no way to explicitly mark the value as a string. Even though the YAML spec includes the “!!str” tag to mark a scalar as a string [8], `yaml-cpp` does not retain this information. In XML it is possible to have a string of value “true”. For example:

```

# In a yaml parameter list

```

```
# myString is a string with value "true"
myString: ~true
# a bool
myBool1: true
# also a bool, even with !!str
myBool2: !!str true
```

Some parameter lists in Drekar contain strings with value “true”, so a workaround in the YAML reader is necessary. The solution is to prefix the string with `~`. The `~` is removed and everything after it becomes a string value. This prefix also works for strings that could be converted to int or double. Note that this is a feature of the YAML parameter list reader, and is not part of the YAML specification [8] or the `yaml-cpp` library [7].

**6. Conclusions & Outlook.** Ifpack2’s implementation of the block relaxation preconditioner suffered from very high overhead. This paper outlined the process of identifying and eliminating the sources of overhead, and plans for future performance improvements. Currently the `BlockRelaxation::apply` operation is 6.94 times faster than before for problems with many small blocks. However, there is more work to be done. Ifpack2 is still 10.1 times slower than ML for the same small block problems. In the future, overhead will be further reduced, and thread-level parallelism will be implemented. After more work Ifpack2’s block relaxation will likely be faster than ML. This will greatly benefit the PIS-CEES project because block relaxation is an excellent preconditioner for ice sheet simulation problems.

`ParameterList` support for the YAML language provides a user-friendly alternative to XML. A familiar C++ interface and automated conversion from XML to YAML make it easy for Trilinos users to migrate their input decks to YAML if they choose. This feature is currently available in the Teuchos package.

## REFERENCES

- [1] About Trilinos. <https://trilinos.org/about>.
- [2] Intel VTune Amplifier 2017. <https://software.intel.com/en-us/intel-vtune-amplifier-xe>.
- [3] Kokkos, a Manycore Device Performance Portability Library for C++ HPC Applications. <https://cfwebprod.sandia.gov/cfdocs/CompResearch/docs/Kokkos-ManyCore.pdf>.
- [4] Lapack routine DGTTRS. <http://www.netlib.org/lapack/explore-3.1.1-html/dgttrs.f.html>.
- [5] ML. <https://trilinos.org/packages/ml/#ml-overview>.
- [6] Teuchos::ParameterList Class Reference. [https://trilinos.org/docs/dev/packages/teuchos/doc/html/classTeuchos\\_1\\_1ParameterList.html](https://trilinos.org/docs/dev/packages/teuchos/doc/html/classTeuchos_1_1ParameterList.html).
- [7] J. BEDER, *yaml-cpp README*, May 2016.
- [8] O. BEN-KIKI, C. EVANS, AND I. DÖT NET, *YAML Ain’t Markup Language Version 1.1*, January 2005.
- [9] T. BRAY, J. PAOLI, AND C. M. SPERBERG-McQUEEN, *Extensible Markup Language (XML) 1.0*, February 1998.
- [10] R. P. PAWLOWSKI, E. C. CYR, J. N. SHADID, AND T. M. SMITH, Building an Open-source Multiphysics PDE Research Tool using Trilinos. <https://trilinos.org/oldsite/events/trilinos\user\group\2011/presentations/Pawlowski\TUG\Drekar.pdf>.
- [11] A. PROKOPENKO, C. M. SIEFERT, J. J. HU, M. HOEMMEN, AND A. KLINVEX, Ifpack2 User’s Guide 1.0. <https://trilinos.org/wordpress/wp-content/uploads/2016/06/ifpack2guide.pdf>, June 2016.
- [12] Y. SAAD, *Iterative Methods for Sparse Linear Systems*, SIAM, second ed.
- [13] A. G. SALINGER, R. A. BARTLETT, A. M. BRADLEY, Q. CHEN, I. P. DEMESKO, X. GAO, G. A. HANSEN, A. MOTA, R. P. MULLER, E. NIELSEN, J. T. OSTIEN, R. P. PAWLOWSKI, M. PEREGO, E. T. PHIPPS, W. SUN, AND I. K. TEZAUR, Albany: Using Component-Based Design to Develop a Flexible, Generic Multiphysics Analysis Code. <http://www.sandia.gov/~ikalash/Albany.pdf>.
- [14] I. TEZAUR, Computational Methods in Ice Sheet Modeling for Next-Generation Climate Simulations. <http://www.sandia.gov/~ikalash/tezaur\duke\nov2015\final.pdf>, November 2015.

- [15] I. TEZAUR, A. SALINGER, M. PEREGO, AND R. TUMINARO, On the Development & Performance of a First Order Stokes Finite Element Ice Sheet Dycore Built Using Trilinos Software Components. [http://www.scidac.gov/PISCEES/documents/tezaur\\_cse2015\\_felix\\_final.pdf](http://www.scidac.gov/PISCEES/documents/tezaur_cse2015_felix_final.pdf), March 2015.
- [16] T. WIESNER, A. PROKOPENKO, AND J. HU, Introduction to MueLu. [https://trilinos.org/wordpress/wp-content/uploads/2015/03/MueLu\\_talk.pdf](https://trilinos.org/wordpress/wp-content/uploads/2015/03/MueLu_talk.pdf), March 2015.

## A C++ EQUATION PARSER FOR NEXT-GENERATION COMPUTING PLATFORMS

JORGE A. PEREZ\* AND DAVID J. LITTLEWOOD†

**Abstract.** This paper discusses the construction of a light-weight, user-friendly, and fast recursive descent Equation Parser for use in simulations on high performance computing platforms. This paper also discusses a method of directly compiling user-defined expressions into C++ code and loading it into an application during runtime through use of dynamic linking.

**1. Introduction.** An Equation Parser translates a string of text describing an equation into an entity that can be used to evaluate the equation for a given set of inputs. For the recursive descent Equation Parsers discussed in this paper, an evaluation tree was used. Each object in the tree is responsible for computing a subsection of the equation; evaluation of the entire equation is done through recursive calls down the evaluation tree on progressively smaller sections of the equation.

An Equation Parser is similar to a compiler, and in some contexts they can be considered synonymous. The term compiler is generally used to refer to a program which converts code into a list of machine instructions, while an Equation Parser is often more limited in scope: it deals only with mathematical equations. The Equation Parsers discussed in this paper are better thought of as interpreters than as compilers, although one made use of the systems compiler and dynamic linking to perform actual compilation of expressions.

Equation parsing forms the core of graphing calculators, spread-sheet software, software to produce graphs and charts, and of symbolic equation solvers. Compilers for all major programming languages parse mathematical expressions in the process of producing the compilation.

Simulation software provides users with the ability to enter equations to control features of the simulation such as the boundary conditions and other simulated behavior; in order for this to be possible, an Equation Parser must be used to convert the textually entered equation into something that can be used for calculation.

There are three stages to equation parsing, as described in this paper:

1. Parsing: The plain-text equation is parsed into a computable expression object.
2. Optimization: If at all possible, the computable expression object created in the first stage is modified to speed up evaluation.
3. Evaluation: Through use of the computable expression object, the equation is evaluated as many times as necessary, for a range of inputs.

**2. Equation Parsing Methodology.** Work on Equation Parsers focused mainly on strategies for equation parsing involving recursive descent, with an additional investigation into direct compilation of mathematical expressions during runtime. The most advanced of the Equation Parsers built allowed for the definition of functions, even non-primitive recursive functions; however, this paper focuses on a slightly more basic but better developed Equation Parser designed for use in conjunction with simulations. Both the recursive descent strategy and the direct compilation strategy were implemented in C++ using a standard GNU Compiler.

**2.1. Implementation.** The Equation Parser is implemented in three parts. An “EquationContext” object, which keeps track of variables and manages memory; the

---

\*Albuquerque Academy, Antonio@PerezExcelsior.com

†Sandia National Laboratories, djlittl@sandia.gov

“ParseEquation” function, which is a member function of the EquationContext that constructs the evaluation tree; and the evaluation tree. Parsing takes place as a series of recursive calls to the ParseEquation function on a series of gradually smaller subsections of the equation, each a valid equation in itself. These sections are delimited by parenthesis, mathematical operators, and function calls. Recursion halts when the ParseEquation function is called on a constant or variable. Constants are parsed into 64 bit floating point values using the C++ standard library, and variables are parsed using a lookup table stored in the EquationContext object.

**2.1.1. Computable Expression Objects.** The Equation Parser receives an equation input as a string of text, and outputs an evaluation tree that can be used to evaluate the equation. Within the context of this paper, the term “computable expression object” refers to a node within an evaluation tree. Conceptually, a computable expression object represents an individual element of a mathematical expression. Each computable expression object is either a constant, a variable, or an operation or function applied to one or more inputs. These inputs themselves may be either constants, variables, or computable expression objects in and of themselves (hence the tree structure). Numeric literals and numbers such as  $\pi$  or  $e$  are constants; any piece of text registered in the ExpressionContext object’s lookup table is a variable.

Computable expression objects store variables as a pointer to a location in memory from which they can retrieve the value of the variable during the course of evaluation. The lookup table maintained by an ExpressionContext object is a one-to-one map between the name of a variable and a pointer to the location in memory where the value of that variable is stored. Variables are registered with the EquationContext objects AddVariable function. An example usage is shown in Fig. 2.1.

For user convenience, a structure called BoundaryConditionFunctor is provided. When initialized with a string, a BoundaryConditionFunctor object will create and store the necessary EquationContext object, register the variables  $x$ ,  $y$ ,  $z$ , and  $t$ , and call on the EquationContext object to parse the string and produce the computable expression object:

---

```
//To parse the equation, just create a new boundary condition functor:
BoundaryConditionFunctor func("x^2 + y^2 + z^2 + t^2");

//To evaluate the equation, call the functor as though it were a
function:
//(Evaluates the equation with x=1.3, y=4.2, z=2.7, and t=2.7)
double output = func(1.3, 4.2, 2.7, 2.9);

//You can change the equation as so. It will parse the new one
automatically.
func = "sin(t)*sqrt(x*x+y*y+z*z)";
```

---

**2.2. Conversion of an equation into an evaluation tree.** The tree is constructed through recursive descent parsing, described above, with a worst-case running time for parsing of  $O(n^2)$ , with  $n$  being the number of characters. The space requirement is  $O(n)$ , however, and evaluation of the expression also occurs in  $O(n)$ . Naturally, equations containing more computationally expensive functions such as sine and cosine take longer to evaluate. The Equation Parser trees expressions first by lower-precedence operators such as addition, and then by higher-precedence operators such as multiplication and power. Precedence for operators is arranged in the following order, from lowest precedence (parsed first) to highest precedence:

---

```

struct BoundaryConditionFunctor {
    double x = 0, y = 0, z = 0, t = 0;
    EquationContext* context;
    //The computable expression object
    RealValuedExpression* expression = 0;
    std::string equation;
    BoundaryConditionFunctor() : context(0) {} //Sets context to nil
    BoundaryConditionFunctor(const std::string& equation)
    {
        if(equation == "")
        {
            //Acts as default constructor if given an empty string
            context = 0;
            return;
        }
        try
        {
            context = new EquationContext();
            //Usage: context->AddVariable(name, reference);
            context->AddVariable("x", x); //Passes a reference to x
            context->AddVariable("y", y); //Passes a reference to y
            context->AddVariable("z", z); //Passes a reference to z
            context->AddVariable("t", t); //Passes a reference to t
            //Creates the computable expression object
            expression = context->ParseEquation(equation);
            //Optimizes the computable expression object
            expression = expression->optimize(context->mem);
            this->equation = equation; //Stores a copy of the equation
        }
        catch(std::invalid_argument arg)
        {
            //Occurs if the equation couldn't be parsed
            if(context) delete context;
            context = 0;
            throw arg;
        }
    }
    //...
    //Not Shown: Code for Copy Constructor, Copy Assignment Constructor
    //...

    //Overload of parenthesis operator so that the object acts as a functor
    inline double operator()(double x, double y, double z, double t)
    {
        this->x = x;
        this->y = y;
        this->z = z;
        this->t = t;
        return *expression; //Evaluates the computable expression object
    }
};

```

---

Fig. 2.1: A section of the code for the BoundaryConditionFunctor structure

1. Ternary operator (parsed left to right)
2. Mod operator (parsed right to left)
3. Addition (parsed right to left)
4. Subtraction (parsed right to left)
5. Multiplication (parsed right to left)
6. Division (parsed right to left)
7. Power (parsed left to right)
8. Negation (parsed left to right)
9. Function arguments and statements in parenthesis

Figs. 2.2, 2.3 and 2.4 show the structure of trees generated when parsing equations using the rules described above, with each cell representing a node. Before optimization, trees generated from parsing equations contain one node for every operation or function, and one leaf for every constant or variable. Prior to optimization, leaves are stored as a computable expression object wrapped around the variable or constant, although after optimization leaves are stored directly as a constant or variable. Further, any section of a tree whose leaves are all constant is evaluated during the course of optimization, and replaced with a constant. Nodes are always computable expression objects. Again, variables are stored internally as a pointer to the location in memory with the actual value of the variable. This is because the value of a variable is expected to change between different evaluations of the expression, and the expression may use a variable multiple times, so it is simplest to have different instances of the same variable all point to the same place in memory. All computable expression objects inherit from the same abstract base class, called a Unit, with the following methods:

1. An abstract method to compute the value of the expression stored by the Unit
2. A virtual method to attempt optimization. This method either returns a pointer to a new optimized Unit, or it returns a pointer to the same Unit. By default, this method just returns a pointer to the Unit from which it was called.
3. A virtual destructor (necessary to avoid memory leaks). The EquationContext object keeps a record of all computable expression objects created during the course of parsing and optimization, and deletes all of them when its own destructor is called.

**2.3. Evaluation.** To compute the value of an expression that has already been parsed, the value of each of the variables is set, and then the root node of the evaluation tree is evaluated. The root node of the tree will recursively call its child nodes to evaluate, all the way down to the leaves, which will simply return whatever value they store (in the case of a constant) or reference (in the case of a variable). Given a tree with  $n$  nodes, this evaluation takes approximately  $O(n)$ , with a larger constant factor for more expensive operations such as division or trigonometric functions, and a smaller constant factor for less expensive operations such as addition or subtraction. The structure of the tree does not (theoretically) affect evaluation time.

These nodes, each a computable expression object, contain a virtual “evaluate” function, which will calculate and return the value of the expression as a floating-point value.

**2.4. Runtime compilation of an expression into machine code.** This strategy uses a compiler already present on the system to compile an equation entered as text directly into machine code. Direct compilation during runtime is possible because a mathematical expression can be compiled into a shared object file, which is then dynamically linked to the main program during runtime. Calculating the output of an equation with this strategy is more computationally efficient than any strategy involving anything akin to evaluation trees.

<b><math>x_1x_2 + y_1y_2 + z_1z_2 - t_1t_2</math></b>							
<b><math>x_1x_2 + y_1y_2</math></b>				<b><math>z_1z_2 - t_1t_2</math></b>			
<b><math>x_1 * x_2</math></b>		<b><math>y_1 * y_2</math></b>		<b><math>z_1 * z_2</math></b>		<b><math>t_1 * t_2</math></b>	
<b><math>x_1</math></b>	<b><math>x_2</math></b>	<b><math>y_1</math></b>	<b><math>y_2</math></b>	<b><math>z_1</math></b>	<b><math>z_2</math></b>	<b><math>t_1</math></b>	<b><math>t_2</math></b>

Fig. 2.2: Tree generated from parsing the equation for the Lorentzian Inner Product for two vectors,  $x_1x_2 + y_1y_2 + z_1z_2 - t_1t_2$ . Constants and variables are bold; the operation that splits each node is red. Addition has lower precedence than subtraction, so at each level the right-most addition sign is parsed first.

C0+C1*x+C2*y+C3*z + C4*t							
C0+C1*x+C2*y + C3*z						C4 * z	
C0+C1*x + C2*y				C3 * z		C4	z
C0 + C1*x		C2 * y		C3	z		
C0	C1 * x	C2	y				
	C1						
	C1	x					

Fig. 2.3: Tree generated from parsing the equation  $C_0 + C_1x + C_2y + C_3z + C_4t$ . Constants and variables are bold; the operation that splits each node is red. Addition has lower precedence than multiplication, so at each level the right-most addition sign is parsed first.

t <= 0 ? 1.1^1.1^1.1^1.1^t*2^2*sqrt(x*x+y*y+z*z) : cos(-y+3*x*x)+sin(y)																			
t <= 0		1.1^1.1^1.1^1.1^1.1^t*2 * sqrt(x*x+y*y+z*z)								cos(-y+3*x*x)+sin(y)									
t	0	1.1^1.1^1.1^1.1^1.1^t * 2						sqrt(x*x+y*y+z*z)				cos(-y+3*x*x)		sin(y)					
		1.1^1.1^1.1^1.1^1.1^t						2		x*x+y*y+z*z					-y+3*x*x		y		
		1.1	1.1^1.1^1.1^1.1^t					x*x+y*y				z*z			-y			3*x*x	
			1.1	1.1^1.1^1.1^t				x*x		y*y		z	z		y	3*x		x	
				1.1		1.1^t		x	x	y	y					3			x
				1.1		t													

Fig. 2.4: Tree generated from parsing the equation seen below. Constants and variables are bold; the operation that splits each node is red. The ternary operator has lower precedence than addition, and so is parsed first.

$$f(x, y, z, t) = \begin{cases} 1.1^{1.1^{1.1^{1.1^t}}} 2\sqrt{x^2 + y^2 + z^2}, & \text{if } t \geq 0 \\ \cos(-y + 3x^2) + \sin(y), & \text{otherwise} \end{cases}$$

An expression is converted into code for a C or C++ function, written to an empty code file, compiled into a shared object file using a standard system compiler (such as the gcc compiler), and then dynamically loaded into the program as a full-fledged function. The expression only has to be compiled once, as multiple nodes can load the function from the same shared object file, and provided the function is without side-effects, its thread-



safe. Because the shared object file containing the function is dynamically linked into the simulation, neither is it necessary to re-start the application.

A potential downside of this approach is its reliance on an external compiler. If the compiler available on the system differs from the one originally used to compile the simulation, there is a risk that the main simulation will be unable to dynamically link to the shared object file containing the newly-compiled function.

Shown below is an example usage of the implementation:

---

```
// To compile an equation, just call CompileEquation
// with the equation and the names of the variables
double(*f)(double, double, double, double) =
    CompileEquation("x*x + y*y + z*z + t*t", "x", "y", "z", "t");

//Can be called like any other function pointer
double output = f(3, 4.2, 5, 7);

//Compiles logistic function
//All standard library math functions are allowed
double(*logistic)(double) =
    CompileEquation("1 / (1 + exp(-x))", "x");

//Compiling a named function is easy too:
//Compiles recursive fibonacci function
double(*fib)(double) =
    CompileFunction("fib", "n < 2 ? n : fib(n-1) + fib(n-2)", "n");

//Prints fibonacci sequence for 0 to 10
for(int i=0; i <= 10; i++)
{
    printf("fib(%i) = %g\n", i, fib(i));
}
```

---

**3. Application of Expression Parsing to Computational Simulation.** Equation Parsers are an important component of simulations and other software that require a method of evaluating mathematical and programmatic expressions not known at compile time.

For example, if a user needs to define a boundary condition for a simulation, it is impractical, and in the case of proprietary closed-source code impossible, to recompile the simulation with the newly specified boundary condition. If the simulator contains an Equation Parser, the user can enter the boundary condition as a mathematical equation. During runtime, the Equation Parser will convert the equation into a function-like object the simulation code can use to calculate the boundary condition. Due to resource and time limitations, simulation codes must be reusable in a variety of situations, and on multiple HPC architectures. Use of equation parsing and related methods allows the creation of highly flexible codes that can easily be adapted to new problems and situations. This flexibility saves time, money, human resources, and it allows for greater specialization of labor: a person need only learn how to properly input new boundary conditions and other mathematical rules defining the behavior of the situation in order to apply a simulation code to their problem.

C++ is a language of choice for large programming projects: it provides a combination of speed and versatility, of both higher and lower-level programming features. As research efforts and work move onto a new generation of high-performance computing machines with heterogeneous architecture, there are concerns that current Equation Parsers may not

perform well on the new systems. To be sure, the issue is not with the performance of the parser, but of the rate at which it can be evaluated with different inputs once parsed.

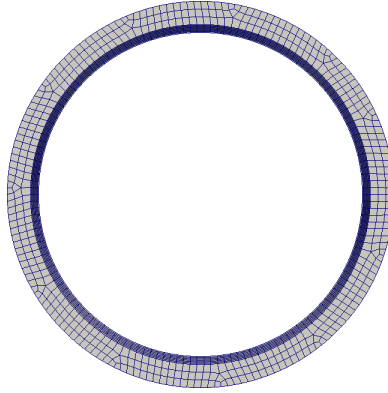
**3.1. Application to the MiniMultiscale Code.** As part of the MiniMultiscale project, a new Equation Parser was built that allows users to define the boundary conditions of simulations as an equation or set of equations. The purpose of MiniMultiscale is to demonstrate the capabilities of DARMA [1], a software package for dynamically load-balancing simulations. In general, however, simulations with non-static boundary conditions will need to compute those boundary conditions at every, or most, time-steps in the simulation. When the boundary conditions are location-dependent, i.e. they vary from one location to another, it is necessary to compute them many millions or billions of times; once for each element of the simulation that interacts with the boundary of the simulation. So it is important that the computation of the boundary conditions is as fast and as efficient as possible.

If the boundary conditions are known at a simulation's compile-time, they can be directly incorporated into the code without issue. In MiniMultiscale, the equation defining a boundary condition is read from a text file at run-time, and cannot be directly incorporated into the original simulation code as that would necessitate that MiniMultiscale reads its own source code files, modifies them, recompiles them, and then restarts itself. Doing that would be time-consuming, it would be difficult to make modifications to MiniMultiscale, and it would expose the source code to users. Were the sourcecode proprietary information, or should it take hours to compile, this would be untenable.

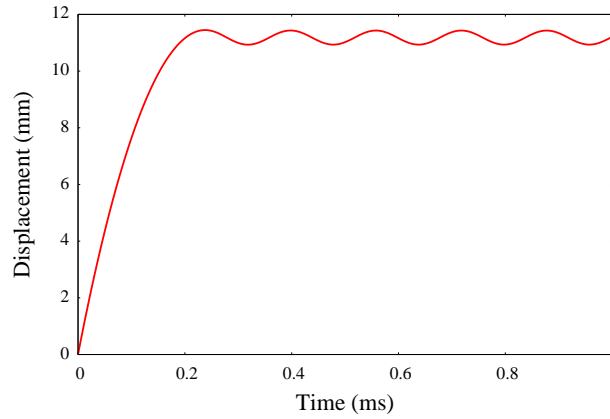
A common application of an Equation Parser in a computational simulation is for the specification of initial conditions and boundary conditions. The application of initial velocities to a solid mechanics simulation of an elastic-plastic ring is illustrated in Fig 3.1. In this case, an initial outward radial velocity is applied to the entire domain using the expressions:

$$\begin{aligned}v_x &= 1.0e5 * \cos(\text{atan2}(y, x)) \\v_y &= 1.0e5 * \sin(\text{atan2}(y, x)) \\v_z &= 0.0\end{aligned}$$

The resulting velocity of a point at the surface of the ring over the course of the simulation is given in Figure 3.1B. The ring initially expands outward, experiencing permanent plastic deformation. After the initial outward expansion is complete, the ring oscillates elastically.



A Finite element model of a ring.



B Surface velocity resulting from the application of an initial outward radial velocity to an elastic-plastic ring.

Fig. 3.1: Application of an Equation Parser to a solid mechanics simulation for the specification of initial conditions.

**4. Conclusions.** Equation parsing is a necessary component of many simulation codes. Depending on its usage in the simulation, evaluation of user-defined expressions can make up a significant fraction of the total computing power expended over the course of running a simulation. It is extremely important therefore that we have Equation Parsers that provide a means of evaluating equations in a computationally-efficient manner. In addition, the corresponding software implementation must be amiable to emerging hardware platforms, e.g. those containing GPUs. If stability can be ensured, dynamic linking of compiled expressions is ideally suited to satisfy this need, however for the time being, strategies based on computable expression objects should suit our needs. For future projects, we recommend a combined approach, in which a program attempts to use dynamic linking as its primary method, but has the capacity to fall back on computable expression objects should dynamic linking fail.

## REFERENCES

- [1] J. J. WILKE, D. S. HOLLMAN, N. L. SLATTENGREN, J. LIFFLANDER, H. KOLLA, F. RIZZI, K. TERANISHI, AND J. C. BENNETT, DARMA 0.3.0-Alpha Specification, Tech. Rep. SAND2016-5397, Sandia National Laboratories, Albuquerque, NM and Livermore, CA, 2016.

## Kokkos Task-DAG Thread-Team Collectives on GPUs and CPUs

JAMES D. STEVENS\* AND H. CARTER EDWARDS†

**Abstract.** A directed acyclic graph of tasks (task DAG), or dependence-imposed topological ordering of algorithm sub-tasks, arises in many application domains. This kind of execution pattern is a valuable capability in any parallel programming model. For some applications, the tasks may be large enough to warrant parallel execution within individual tasks on some architectures. We present enhancements to the task-DAG functionality in the Kokkos programming system. Kokkos provides an architecture-independent C++ programming model enabling within-node parallelism on diverse high performance computing architectures, currently implemented with OpenMP, Pthreads, and CUDA back ends. Kokkos provides task-DAG execution with both intra- and inter-task parallelism, with each task executed by either a single thread or a team of threads. We enhance Kokkos’ OpenMP and CUDA back end for task-DAG execution capabilities with fast thread-team collective operations, including parallel reduce and parallel scan.

### 1. Introduction.

**1.1. Kokkos.** Kokkos [3, 2], an intra-node (shared memory) parallel programming model and C++ library implementation, enables applications written in C++ to be performance portable across next-generation high performance computing (HPC) node architectures including multicore CPU, Intel Xeon Phi, and NVIDIA GPU. When used with C++11 *lambda* expressions, applications can implement intra-node parallel algorithms with the same degree of ease as using OpenMP [1] or OpenACC, and subsequent optimization of non-trivial algorithms is easier [4]. The foundation of Kokkos’ programming model is the integrated, polymorphic mapping of parallel and data patterns to HPC node architecture through architecture-specialized *back ends*. This mapping is transparently inserted in application code through C++ template metaprogramming techniques, as opposed to language extensions or source-to-source translators.

Kokkos was initially released at [github.com/kokkos](https://github.com/kokkos) in March 2015 with mappings for data parallel patterns (*parallel for*, *parallel reduce*, and *parallel scan*) and multidimensional arrays. Parameters of these mappings included *parallel patterns and array dimensions* that define a “skeleton” for computations and data structures, *execution and memory spaces* that identify where computations should be run and where data should be allocated, and *execution policy and array layout* that specify how computation should be scheduled and how array data should be laid out. The key to data parallel performance portability is the integrated mapping of parallel computations and multidimensional array data structures such that computations have architecture-appropriate memory access patterns; *e.g.*, caching on CPU and coalescing on GPU.

A major portion of a typical application’s intra-node parallel algorithms can be expressed in terms of data parallel patterns, especially when augmented with atomic operations or coloring scheme to scalably manage race conditions. These patterns fail to address the algorithms that conform to the *parallel task-DAG* pattern; *i.e.*, whose “skeleton” is a heterogeneous collection of potentially parallel computations that have a directed acyclic graph (DAG) of dependencies. For example, Cholesky matrix factorization can be implemented as a collection of block submatrix operations (tasks).

Kokkos’ parallel task-DAG pattern was developed over several years through a Sandia laboratory directed research and development (LDRD) project. Portability and performance for GPU architectures with their large number of very simple cores and “bare bones” runtime was a major challenge and unique focus of this research and development. This challenge

\*University of Illinois at Urbana-Champaign, [jdsteve2@illinois.edu](mailto:jdsteve2@illinois.edu)

†Sandia National Laboratories, [hcedwar@sandia.gov](mailto:hcedwar@sandia.gov)

was exacerbated by the goal to enable a task running on a GPU to create a new task with inter-task dependences that can subsequently run on the GPU without communicating with the host CPU or its memory. This functionality is required for performance of algorithms with a *dynamic* task DAG.

An additional unique feature of Kokkos’ parallel task DAG is that a task may have internal data parallelism. This capability is implemented by running a task on a team of “hardware threads.” These task-thread-teams are comprised of those threads that share their entire memory subsystem; *e.g.*, hyperthreads of a single CPU core or lanes of a GPU warp. Tasks with sufficient computational intensity utilize their thread team to extract additional algorithmic parallelism and additional speedup of the computation.

**1.2. CUDA.** A brief overview of the aspects of thread execution on CUDA-enabled GPUs most relevant to this paper is provided here. For more detail on CUDA, see [5].

Nvidia GPUs contain hundreds of individual streaming multiprocessors. CUDA threads are grouped into collections called *blocks*, with each block of threads running on a single multiprocessor. Within each block, threads are executed in batches of 32 threads called *warps*. Threads within a warp all execute the same operation simultaneously in lock-step.

The maximum number of blocks that can run concurrently on a single multiprocessor is limited, so blocks with too few warps may under-utilize the GPU, yielding poor performance. The block size that yields best performance depends on the application and the hardware, but block sizes between 128 and 256 threads are common choices for good performance.

Kokkos task thread teams were initially implemented on GPUs as CUDA thread blocks. However, the task DAGs generated for many problems, including the Cholesky factorization mentioned earlier, contain tasks that do not require enough parallel work to be efficiently completed by 128 GPU threads. Thus, for the reasons listed above, this can lead to less-than-ideal performance. This is the primary motivation for allowing task teams that are warp-sized or smaller.

**1.3. Overview.** In this paper, we discuss the implementation of thread-team collective operations for Kokkos. First we describe CUDA shuffle operations and how we use them to implement efficient warp-level team reductions and scans for GPU execution. Then we describe our implementation of the same team collective operations for multi-core CPU execution with OpenMP, as well as serial CPU execution.

## 2. Task Team Collectives.

**2.1. GPU Task Team Collectives.** We make extensive use of CUDA *shuffle* operations in implementing team collectives for warp-sized Kokkos task teams. Shuffle operations, available in CUDA versions 3.X and later, allow threads to access data held in registers by *other* threads within the same warp. They provide an often faster alternative to shared memory usage when threads within a warp need to share data, and reduce the amount of CUDA block-shared memory required by an algorithm. During a shuffle operation, threads have direct access to the registers “owned” by other threads participating in the shuffle operation. For more details about CUDA shuffle operations, see [5]. Figure 2.1 illustrates the kinds of shuffle operations we use in our warp-level collectives.

Kokkos task team *members* may each contain their own internal vector parallelism. For warp-sized GPU task teams, this leads to warps of threads that represent fewer than 32 team members, with the vector parallelism expressed across threads within the warp. For example, a warp of 32 threads might have 4 team members with 8 threads each. These are implemented with the `threadIdx.y` variable representing the member’s id within the team, and the `threadIdx.x` variable representing the vector lane within each member, as shown in Figure 2.2.

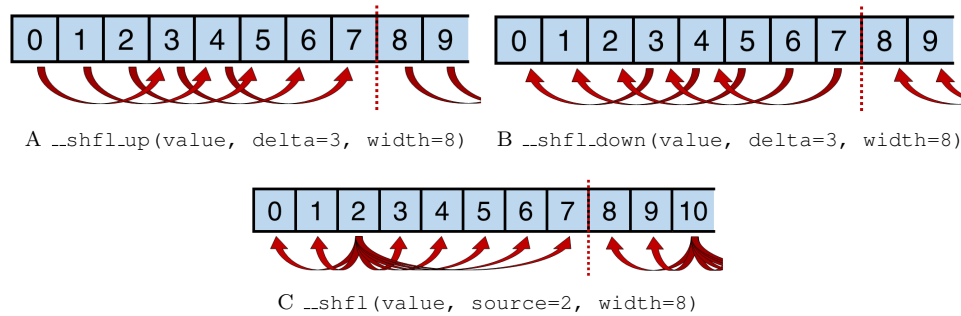


Fig. 2.1: CUDA shuffle operations allow data movement (red arrows) between threads within a warp. In each example there is one thread per data element, but threads gain access to other threads' data via a shuffle operation.

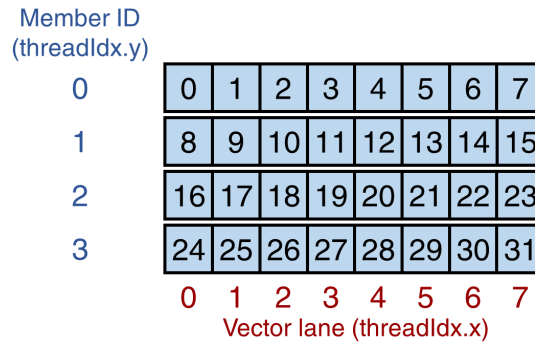


Fig. 2.2: Viewing a CUDA warp as team of four “thread” members, each of which has eight vector lanes

We want to be able to perform collective operations both across the vector lanes within a team member *and* across corresponding vector lanes between team members. We will refer to these as intra-member collectives and inter-member collectives, respectively. Figure 2.3 provides a graphical depiction of intra- and inter- member collectives.

To enable an inter-member collective within a warp containing  $m$  members, each with  $v$  vector lanes (i.e.  $m \times v == \text{warp size} == 32$ ), we use shuffle operations to perform  $v$  simultaneous stride- $v$  collectives, each on  $m$  elements. Similarly, to enable an intra-member collective, we use shuffle operations to perform  $m$  simultaneous stride-1 collectives, each on  $v$  elements. Algorithms 2.1 and 2.2 display inter-member scan and reduction algorithms, and Algorithms 2.3 and 2.4 display intra-member scan and reduction algorithms. The user may specify the reduction `join()` operator, or use the default (addition). When the range of indices on which the team is to perform the collective operation is larger than the team size, the team iterates over the indices, one team-sized chunk at a time. This aspect of the algorithm is not shown.

According to current documentation, CUDA only provides shuffle operations for 32-bit data types. Multiple shuffle calls can be used to shuffle larger data types, with each call sending one 32-bit section of the larger piece of data. Kokkos already has built-in shuffle

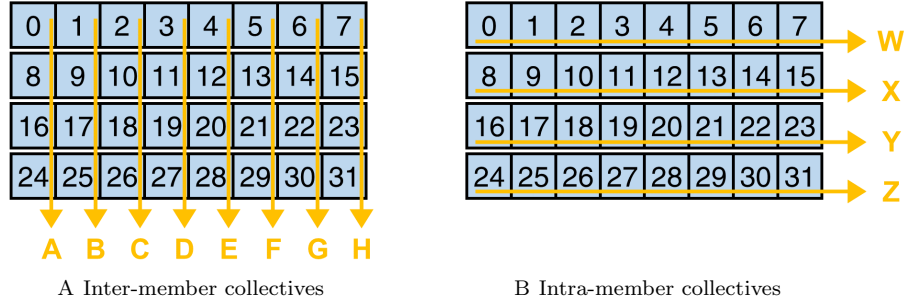


Fig. 2.3: Two kinds of task team collective operations

functions that use this method to shuffle larger data types, so we use these functions in our implementation of the algorithms described.

---

**Algorithm 2.1** Inter-member scan (CUDA)

Exclusive scan on val for each vector lane (called by each thread)

---

```

function INTERMEMBERSCAN(val)
    stride ← blockDim.x                                ▷ Stride == vector length
    team_size ← blockDim.y
    member_id ← threadIdx.y
    temp ← 0
    offset ← stride
    while offset < WARPSIZE do                        ▷ First do inclusive scan
        temp ← shfl_up(val, offset, WARPSIZE)
        if member_id × stride ≥ offset then
            val ← val + temp
        offset ← offset × 2
    val ← shfl_up(val, stride, WARPSIZE)              ▷ Make scan exclusive
    if member_id == 0 then
        val ← 0
    return val

```

---



---

**Algorithm 2.2** Inter-member reduction (CUDA)

Reduce val to member 0 for each vector lane (called by each thread)

---

```

function INTERMEMBERREDUCTION(join(), val)
    stride ← blockDim.x                                ▷ Stride == vector length
    team_size ← blockDim.y
    lane_delta ← team_size × stride / 2
    while lane_delta ≥ stride do
        temp ← shfl_down(val, lane_delta, team_size × stride)
        val ← join(val, temp)
        lane_delta ← lane_delta / 2
    return val                                         ▷ val only defined for member 0

```

---



**Algorithm 2.3** Intra-member scan (CUDA)

Exclusive scan on val within member vector (called by each thread)

---

```

function INTRAMEMBERSCAN(val)
  vector_length  $\leftarrow$  blockDim.x
  vector_lane  $\leftarrow$  threadIdx.x
  member_id  $\leftarrow$  threadIdx.y
  temp  $\leftarrow$  0
  offset  $\leftarrow$  1
  while offset < vector_length do                                 $\triangleright$  First do inclusive scan
    temp  $\leftarrow$  shfl_up(val, offset, vector_length)
    if vector_lane  $\geq$  offset then
      val  $\leftarrow$  val + temp
    offset  $\leftarrow$  offset  $\times$  2
  val  $\leftarrow$  shfl_up(val, 1, vector_length)                         $\triangleright$  Make scan exclusive
  if vector_lane == 0 then
    val  $\leftarrow$  0
  return val

```

---

**Algorithm 2.4** Intra-member reduction (CUDA)

Reduce val to vector lane 0 for each member (called by each thread)

---

```

function INTRAMEMBERREDUCTION(join(), val)
  vector_length  $\leftarrow$  blockDim.x
  team_size  $\leftarrow$  blockDim.y
  lane_delta  $\leftarrow$  vector_length/2
  while lane_delta  $\geq$  1 do
    temp  $\leftarrow$  shfl_down(val, lane_delta, vector_length)
    val  $\leftarrow$  join(val, temp)
    lane_delta  $\leftarrow$  lane_delta/2
  return val                                                          $\triangleright$  val only defined for vector lane 0

```

---

**2.2. OpenMP Task Team Collectives.** We implement the same collective operations for OpenMP execution on CPUs. For a reduction each member of a team (one OpenMP thread) first performs a serial reduction over its subset of the data. We could then perform a parallel reduction across team members, however since the OpenMP thread teams will usually have only a few members, it should be faster for just one “team lead” thread to perform a serial reduction across team members. First each thread places its result into pre-allocated team shared memory, and then the lead thread performs a serial reduction on the data in shared memory. When the lead thread finishes, the result is broadcast back to all team members using the same shared memory.

The OpenMP thread team scan algorithm is similar to the reduction. First, each member performs a serial scan on its subset of the data, with all members performing serial scans simultaneously. Second, each member places its serial scan total in shared memory and the lead thread performs a serial scan on these totals. Finally, each member re-scans its subset of the data, this time adding in the running total, which is now stored in shared memory.

The caller passes the scan operation to be performed on each index as a lambda expression, i.e.,

```
[&](int i, long &val, bool final){ if(final){result[i] = val;} val
```

```
+= i; },
```

and can choose an inclusive or exclusive scan by incrementing `val` before or after storing it, respectively. The `final` variable is used to determine whether this is the first (intra-member) scan pass or the final pass.

**2.2.1. Thread Team Synchronization on CPU Architectures.** Thread team synchronization is a performance-critical operation used in the team collectives described above. For example a task’s internal thread team `parallel_reduce` operation synchronizes hardware threads with a `team_barrier` and then joins hardware threads’ individual values into a team total. We use the barrier algorithm summarized in Figure 2.4 which minimizes the number of shared variables accessed and minimizes writes to those shared variables.

The variable `sync_shared` is a cache-aligned array of two 64-bit integer values that is shared by the team. Within each call to `team_barrier` each thread team member writes exactly one byte within this array to a designated value, and then waits for all team members to write their respective bytes. The synchronization variable alternates between calls to `team_barrier` and the synchronization value is flipped every other call.

**2.3. Serial Task Team Collectives.** We implement the same collectives for task teams consisting of a single member. These perform the same reduction and scan operations described above in serial.

---

```

void team_barrier_init() {
    sync_shared[0] = 0 ;
    sync_shared[1] = 0 ;
    for ( int i = 0 ; i < team_size ; ++i ) {
        sync_value |= int64_t(1) << (8*i);
        sync_mask |= int64_t(3) << (8*i);
    }
}

void team_barrier() {
    // alternate between two team-shared synchronization
    // variables
    int64_t volatile * const sync = sync_shared
        + ( sync_step & 0x01 );

    // this thread's portion of team-shared sync. variable
    int8_t volatile * const sync_self = ((int8_t volatile*) sync)
        + team_rank;

    // Write single byte to signal team that this thread arrived
    *sync_self = int8_t( sync_value & 0x03 );

    // Spin-wait for whole team to arrive at this statement
    while( sync_value != *sync );

    // An alternating pattern for the synchronization variable
    // and value minimizes writes to the synchronization variable
    // and prevents confusion between iterative calls to the
    // barrier. If team_size == 4 then:
    // step 0: while( sync_shared[0] != 0x0000000001010101 );
    // step 1: while( sync_shared[1] != 0x0000000001010101 );
    // step 2: while( sync_shared[0] != 0x0000000002020202 );
    // step 3: while( sync_shared[1] != 0x0000000002020202 );
    // step 4: while( sync_shared[0] != 0x0000000001010101 );
    // etc.

    ++sync_step ; // how many times the barrier has been called

    if ( 0 == ( 0x01 & sync_step ) ) { // every other step
        sync_value ^= sync_mask ; // flip the sync value
    }
}

```

---

Fig. 2.4: On CPU architectures the hardware thread team of up to eight threads synchronizes by each thread setting a byte within a shared synchronization variable to a specific value and then all threads wait for all team members' values to be set.

**3. Conclusions.** We have implemented task-team collective operations for the CUDA, OpenMP, and Serial Kokkos back ends. Task-DAGs with opportunities for intra-task parallelism, including the collective operations presented here, arise frequently in the kinds of

scientific applications Kokkos targets; these capabilities further Kokkos' goal of providing platform-independent task-DAG execution for scientific applications.

The next step will be to evaluate the performance of these collective operations in a typical application. We will use Kokkos' task teams to perform a Cholesky matrix factorization, in particular comparing performance of the (previous) CUDA block-sized task teams to the new warp-sized task teams. As stated previously, we expect the warp-sized task teams to perform better since they should more fully utilize the GPU.

#### REFERENCES

- [1] L. DAGUM AND R. MENON, OpenMP: An Industry Standard API for Shared-Memory Programming, *IEEE Computational Science and Engineering*, 5 (1998), pp. 46–55.
- [2] H. C. EDWARDS, D. SUNDERLAND, V. PORTER, C. AMSLER, AND S. MISH, Manycore Performance-Portability: Kokkos Multidimensional Array Library, *Scientific Programming*, (2012), pp. 89–114.
- [3] H. C. EDWARDS, C. R. TROTT, AND D. SUNDERLAND, Kokkos: Enabling Manycore Performance Portability through Polymorphic Memory Access Patterns, *J. of Parallel and Distr. Comp.*, 74 (2014), pp. 3202–3216.
- [4] R. HORNUNG, H. JONES, J. KEASLER, R. NEELY, O. PEARCE, S. HAMMOND, C. TROTT, P. LIN, C. VAUGHAN, J. COOK, R. HOEKSTRA, B. BERGEN, J. PAYNE, AND G. WOMELDORFF, ASC Tri-Lab Co-design Level 2 Milestone Report 2015, technical report, Lawrence Livermore National Laboratory, Sandia National Laboratories, and Los Alamos National Laboratory, September 2015.
- [5] C. NVIDIA, Cuda C Programming Guide, Nvidia Corporation, (2015).