SANDIA REPORT

SAND2016-9802
Unlimited Release
Printed October 2016

The Aeras Next Generation Global Atmosphere Model

William F. Spotz (PI), Peter A. Bosler, Steven W. Bova, Jeffrey A. Fike, Oksana Guba, James R. Overfelt, Erika L. Roesler, Andrew G. Salinger, Thomas M. Smith, Irina K. Tezaur, Jerry Watkins, Irina P. Demeshko

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

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.

Sandia National Laboratories

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

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

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

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

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2016-9802
Unlimited Release
Printed October 2016

The Aeras Next Generation Global Atmosphere Model

William F. Spotz, PI	Multiphysics Applications Department
Peter A. Bosler	Multiphysics Applications Department
Steven W. Bova	Computational Thermal & Fluid Mechanics Department
Jeffrey A. Fike	Aerosciences Department
Oksana Guba	Optimization & Uncertainty Quantification Department
James R. Overfelt	Computational Multiphysics Department
Erika L. Roesler	Atmospheric Sciences Department
Andrew G. Salinger	Computational Mathematics Department
Thomas M. Smith	Multiphysics Applications Department

Sandia National Laboratories
P.O. Box 5800-MS1320
Albuquerque, NM 87185

Irina K. Tezaur, Jerry Watkins
Extreme Scale Data Science & Analytics Department
Sandia National Laboratories
P.O. Box 969-MS9159
Livermore, CA 94551-9159

Irina P. Demeshko
Los Alamos National Laboratory
P.O. Box 1663
Los Alamos, NM 87545

Abstract

The Next Generation Global Atmosphere Model LDRD project developed a suite of atmosphere models: a shallow water model, an x-z hydrostatic model, and a 3D hydrostatic model, by using Albany, a finite element code. Albany provides access to a large suite of leading-edge Sandia high-performance computing technologies enabled by Trilinos, Dakota, and Sierra. The next-generation capabilities most relevant to a global atmosphere model are performance portability and embedded uncertainty quantification (UQ). Performance portability is the capability for a single code base to run efficiently on diverse set of advanced computing architectures, such as multi-core threading or GPUs. Embedded UQ refers to simulation algorithms that have been modified to aid in the quantifying of uncertainties. In our case, this means running multiple samples for an ensemble concurrently, and reaping certain performance benefits. We demonstrate the effectiveness of these approaches here as a prelude to introducing them into ACME.

Acknowledgment

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

Contents

Appendix

List of Figures

List of Tables

Nomenclature

- Names and terminology:

ACME Accelerated Climate Model for Energy. This is the DOE climate model sponsored by BER. It was split off from CESM, which was formerly sponsored by BER, in cooperation with NSF.

Aeras The next-generation global atmosphere model developed under this LDRD. Aeras is actually a suite of atmosphere models: a shallow water model, a 2D x-z hydrostatic model, and a 3D hydrostatic model.

Albany A finite element code developed at Sandia that utilizes several leading edge Sandia computational technologies, such as Trilinos, Dakota, and Sierra. Albany is the enabling technology for Aeras, which is the global atmosphere model developed under the auspices of this LDRD. Albany has been used to develop a diverse set of multiphysics applications, ranging from compressible flows, to land ice, to quantum device simulators.

BER Biological and Environmental Research office, within the DOE Office of Science.

CAM The Community Atmosphere Model. An umbrella term for a suite of atmosphere models in both ACME and CESM that can be switched out at the user's discretion. The suite of atmosphere models include Eulerian spectral transform, semi-Lagrangian spectral transform, finite volume, and spectral elements. CAM refers to the combination of both a dynamical core and physics parameterizations.

CAM-SE The Community Atmosphere Model using Spectral Elements. The default atmosphere model in both ACME and CESM, in which the horizontal discretization technique is spectral elements.

CESM Community Earth System Model. This is a climate model that was formerly cosponsored by BER with NSF. The current DOE model, ACME, was split off from this model.

Concurrent Samples A technique devised as a part of this LDRD in which multiple simulations are run concurrently, obtaining performance gains by amortizing certain common calculations, consolidating large communication buffers, and providing more work for processors to perform. Multiple simulations are needed to compute ensembles for UQ.

Dakota A software package that provides optimization and uncertainty quantification capabilities, developed at Sandia. Dakota provides optimization and UQ capabilities to

Albany.

Dynamical Core A dynamical core for an atmosphere or ocean model primarily solves the conservation equations for continuity, momentum and energy, along with tracer advection. Often abbreviated "dycore."

Embedded UQ Any UQ technique that requires alteration of the solution algorithm.

HOMME The High-Order Multiscale Modeling Environment. This is a production-quality atmospheric dynamical core that provides the basis for CAM-SE. Developed initially at NCAR, HOMME is now primarily developed and maintained at Sandia.

Horizontal For a global atmosphere model, the "horizontal" dimensions refer to a coordinate system that is locally tangent to the surface of the sphere.

HPC High performance computing. Typically refers to modeling and simulation performed on modern supercomputers.

Kokkos A software package developed at Sandia, and related programming model, that enables performance portability. Kokkos represents Sandia's strategy for developing HPC code that can run efficiently on multiple advanced computing architectures without expensive porting exercises. Kokkos comes as a part of Trilinos, or can be acquired separately. Aeras performance portability was enabled through Kokkos.

NCAR The National Center for Atmospheric Research in Boulder, Colorado.

Nonhydrostatic Model A global, 3D, atmosphere model, that does not employ hydrostatic assumptions. This is required for resolutions where grid cells are roughly 10km on a side or less. Aeras does not have a nonhydrostatic model.

Performance Portability The ability to develop computer codes that run efficiently on multiple advanced computing architectures without expensive porting exercises.

Phalanx A Trilinos package designed to solve general partial differential equations that decomposes a complex problem into a number of simpler problems (*evaluators*) with managed dependencies.

Physics (Parameterizations) A catch-all term for any sub-models within an atmosphere or ocean model that are not considered part of the dynamical core. These sub-models often describe processes that occur on scales smaller than the available resolution, and so are implemented as parameterizations. Examples include solar radiation, clouds, aerosols, chemistry, turbulence, etc.

Shallow Water Model A 2D model on the sphere that approximates the atmosphere as a shallow film of air with a variable thickness. Fluid velocities are assumed constant along the vertical extent of that thickness for every point on the surface of the sphere. This is useful for verifying 2D dynamics formulations in the horizontal direction and the handling of potential singularities at the poles due to the spherical coordinate sys-

tem. Aeras has a shallow water model.

Sierra An HPC multiphysics application framework developed at Sandia. Many Sierra capabilities have been extracted from Sierra and combined into a Trilinos package STK (Sierra Toolkit). Albany depends heavily on the STK meshing capabilities.

STK Sierra Toolkit. Capabilities that have been extracted from Sierra, put into a toolkit, or package, form, and added to Trilinos. These capabilities include meshing tools, coupling technologies, transfer operations, etc.

Stokhos A Trilinos package that enables solution of stochastic partial differential equations via embedded uncertainty quantification techniques.

Trilinos A suite of over 50 high-performance computing packages that provide solvers and other simulation support capabilities, developed primarily at Sandia. Albany leverages a significant portion of the capabilities provided by Trilinos.

UQ Uncertainty quantification. The analysis of how uncertainties in input to a model affect the range and statistics of the outputs of a model.

Vertical For a global atmosphere model, the "vertical" dimension refers to the local direction perpendicular to the surface of the sphere. Equivalent to the radial direction.

***x*-*z* Hydrostatic Model** A 2D model with one dimension along the surface of the sphere and a second dimension in the vertical direction. This is useful for verifying formulations of vertical coordinate systems. Aeras has an *x*-*z* hydrostatic model.

3D Hydrostatic Model A 3D model of the global atmosphere. This is the most common set of governing equations for the atmosphere in production-level climate models. However, the hydrostatic assumptions break down at higher resolutions (roughly speaking, grid cells 10km on a side or less), and for these grids, the non-hydrostatic equations are required, and are becoming more common. Aeras has a 3D hydrostatic model.

- Variable names from the governing equations and discretization:

a Radius of the earth, 6.37122×10^6 m

$A(\eta), B(\eta)$ Coefficients that define the hybrid vertical coordinate system

d_i Distance functions in the advection in 3D deformational flow test problem

δ Divergence, equivalent to $\nabla \cdot \mathbf{u}$

η Non-dimensional vertical coordinate

$\dot{\eta}$ Vertical velocity

f Coriolis parameter

$\mathbf{f}$ Mapping function from reference element to a shell element on the sphere

$\mathbf{f}^{-1}$ Mapping function from a shell element on the sphere to the reference element

g Gravity, 9.80616m/s^2

$(\mathbf{g}_1, \mathbf{g}_2)$ Basis vectors for covariant coordinate system

$(\mathbf{g}^1, \mathbf{g}^2)$ Basis vectors for contravariant coordinate system

h Atmospheric thickness in the shallow water equations

h_s Surface height of topography

H Fluid surface height, equivalent to $h + h_s$

$\hat{\mathbf{k}}$ Unit vector in the vertical (radial) direction

λ Longitude

ω Total derivative of pressure with respect to time

Ω Rotation rate of the earth, $7.292 \times 10^{-5}\mathrm{s}^{-1}$

p Pressure, also polynomial degree of spectral elements

ϕ Geopotential

ϕ_s Surface geopotential

ψ Stream function

$(\mathbf{r}_\lambda, \mathbf{r}_\theta)$ Basis vectors for spherical coordinates

r In shallow water test cases, the great circle distance from a test feature center

R Gas constant

ρ Density

t Time

T Temperature

T_v Virtual temperature

τ Hyperviscosity coefficient

θ Latitude

$\mathbf{u}$ Velocity vector, equivalent to (u, v)

u Longitudinal velocity component

v Latitudinal velocity component

x Horizontal coordinate for x-z hydrostatic equations

z Vertical spatial coordinate

ζ Vorticty, equivalent to $\nabla \times \mathbf{u}$

- Variable names from the shallow water test cases:

α The angle between the axis of solid body rotation and the polar axis of the spherical coordinate system

h_{s0} In shallow water test case 5, the isolated mountain height

K Test case 6 Rossby-Haurwitz constant

(λ_c, θ_c) Location of the center of the isolated mountain in shallow water test case 5, expressed in spherical coordinates

ω Test case 6 Rossby-Haurwitz constant

R The radius of test features, such as the cosine bell initial condition or the isolated mountain; also a Rossby-Haurwitz constant

u_0 Reference velocity

- Variable names from the 3D hydrostatic test cases:

b Normalization parameter, set to 0.2

Γ Temperature lapse rate, set to -0.0065 K/m

h_0 Mountain peak height, set to 2000 m

k Velocity magnitude, set to $10a/t_p$

p_0 Reference pressure corresponding to T_0, set to 1000 hPa

p_{top} Pressure at model top

q A passive tracer in the advection in 3D deformational flow test problem

r_i Horizontal great circle distance functions in the advection in 3D deformational flow test problem

r_m Great circle distance from the mountain peak

R_M Mountain radius, set to $3\pi/4$

R_t The horizontal half-width of the cosine bell in the advection in 3D deformational flow test problem, set to $a/2$

$s(p)$ A tapering function in the advection in 3D deformational flow test problem that causes the vertical velocity to smoothly decay toward zero near the upper and lower boundaries

t_p Reversibility time period, set to 12 days

T_0 Reference temperature corresponding to p_0, set to 300 K

w_0 Vertical pressure velocity magnitude, set to $230\pi/t_p$

z_s Surface height of orography

Z_t The vertical half-width of the cosine bell in the advection in 3D deformational flow test problem, set to 1000 m

Chapter 1

Introduction

The DOE-sponsored Accelerated Climate Model for Energy (ACME) [1], and its predecessor, the Community Earth System Model (CESM) [5], are comprised of components such as the Community Atmosphere Model (CAM) that are developed from code bases that are well over a decade old, and in some cases, several decades old. Porting them to the wide variety of current and future computing architectures, such as multi-core or many-core processing nodes, or graphical processing units (GPUs), is arduous, time consuming and expensive. Adapting them to new purposes, such as interfacing to human impacts models, is also difficult, as they were not designed to work with anything but the most basic uncertainty quantification (UQ) and optimization methods.

The Aeras project was funded by the Sandia National Laboratories' Laboratory-Directed Research and Development (LDRD) program. It is aimed at addressing these issues by developing a global atmosphere model that supports both performance portability and embedded UQ. The Energy and Climate investment area goals of advancing polar simulation capabilities and assessing U.S. security impact risks by modeling human response at the regional level, both depend upon higher resolution global models and UQ support. Efficient high-resolution simulations require performance portable algorithms, and the most effective UQ algorithms require embedded logic. Neither of these capabilities is provided by ACME or CESM, and both are best implemented from the ground up.

Aeras, the new atmospheric dynamical core we developed under this LDRD, unites our expertise with Sandia's Trilinos suite of high performance software packages and ACME's Community Atmosphere Model, resulting in a next generation atmosphere model that promotes machine portability, and that is designed from the onset to support embedded UQ. In addition to these features, it also uses modern software engineering techniques, flexible design, and advanced libraries.

The Aeras project has successfully demonstrated what is possible in a modern Earth system model, and provides the groundwork for using such a system model in a DOE policy and decision support role for enabling the mitigation of, or adaptation to, climate change impacts. This project has also helped Sanda earn a leadership role in computational science for climate research, as evidenced by Sandia's successful lead of the multi-institution ACME Software Modernization proposal, awarded in FY 2016 by the DOE Office of Science, Office of Biological and Environmental Research (BER), under the Climate Model Development and Validation (CMDV) program.

Chapter 2

Governing Equations

Historically, the atmospheric components of operational climate models have solved the 3D hydrostatic equations. As fidelity increases with grid cells that drop blow roughly 10 km, these models begin to violate the hydrostatic assumptions, and so high-resolution models are transitioning to solving the 3D nonhydrostatic equations, which are much harder to model. As a prototyping project limited to three years, we restricted ourselves to solving the hydrostatic equations. Our demonstrations of performance portability and embedded UQ will translate directly to the nonhydrostatic equations.

There are two 2D simplifications of the 3D hydrostatic equations, and these models are very useful for the verification of the horizontal (i.e. tangent to the surface of the sphere) and vertical (i.e. radial) discretizations in isolation before combining them in the 3D model. The first simplification is the shallow water equations, in which the vertical height or thickness of the atmosphere is reduced to an independent variable. Prognostic variables, such as velocity and temperature, are assumed constant across that thickness at any given point on the surface of the sphere. The second simplification is the *x*-*z* hydrostatic equations, in which the 2D horizontal coordinates are reduced to a single spatial coordinate.

In this section we will first present the 3D hydrostatic equations. From this point it is straightforward to apply the appropriate simplifications to obtain the shallow water equations and the *x*-*z* hydrostatic equations.

2.1 3D Hydrostatic Equations

The hydrostatic equations make several assumptions about the magnitude of vertical velocities and derivatives, and so it is natural to consider horizontal and vertical derivatives separately. To this end, the ∇ operator will denote a 2D, horizontal operator, and vertical derivatives will be denoted with $\partial/\partial\eta$. Similarly, $\mathbf{u} = (u, v)$ denotes the 2D horizontal velocity. The vertical velocity is $\dot{\eta}$. We defer the explicit definition of the vertical coordinate η until section 3.2.2.

The prognostic momentum equation is

$$\frac{\partial \mathbf{u}}{\partial t} + (\zeta + f)\hat{\mathbf{k}} \times \mathbf{u} + \nabla\left(\frac{1}{2}\mathbf{u}^2 + \phi\right) + \dot{\eta}\frac{\partial \mathbf{u}}{\partial \eta} + \frac{RT_v}{p}\nabla p = 0, \tag{2.1}$$

where $\zeta = \nabla \times \mathbf{u}$ is the vorticity, f is the Coriolis parameter, and $\hat{\mathbf{k}}$ is the unit vector in the vertical direction. For stability purposes, we often have to add some form of artificial viscosity to the momentum equations. Experience shows us that hyperviscosity is an effective technique for stabilizing the 3D hydrostatic equations when solving with the spectral element method. The 3D hydrostatic momentum equation with hyperviscosity is

$$\frac{\partial \mathbf{u}}{\partial t} + (\zeta + f)\hat{\mathbf{k}} \times \mathbf{u} + \nabla\left(\frac{1}{2}\mathbf{u}^2 + \phi\right) + \dot{\eta}\frac{\partial \mathbf{u}}{\partial \eta} + \frac{RT_v}{p}\nabla p - \tau\nabla^4\mathbf{u} = 0, \tag{2.2}$$

where τ is the artificial hyperviscosity coefficient and ∇^4 is the biharmonic operator in the horizontal direction.

Three quantities are obtained from diagnostic equations, starting with ϕ, the geopotential, given by the following:

$$\phi = \phi_s + \int_{\eta_s}^{\eta} \frac{RT}{p}\,\mathrm{d}\eta'. \tag{2.3}$$

The vertical velocity $\dot{\eta}$ is obtained from

$$\dot{\eta}\frac{\partial p}{\partial \eta} = -\frac{\partial p}{\partial t} - \int_0^{\eta} \nabla \cdot \left(\frac{\partial p}{\partial \eta'}\right)\mathrm{d}\eta', \tag{2.4}$$

and the virtual temperature T_v is given by

$$RT_v = (c_p - qc_v)T. \tag{2.5}$$

For continuity, $\partial p/\partial \eta$ serves as a density-like quantity, and the governing prognostic equation is

$$\frac{\partial}{\partial t}\left(\frac{\partial p}{\partial \eta}\right) + \nabla \cdot \left(\mathbf{u}\frac{\partial p}{\partial \eta}\right) + \frac{\partial}{\partial \eta}\left(\dot{\eta}\frac{\partial p}{\partial \eta}\right) = 0. \tag{2.6}$$

Hyperviscosity stabiliazation of this equations gives

$$\frac{\partial}{\partial t}\left(\frac{\partial p}{\partial \eta}\right) + \nabla \cdot \left(\mathbf{u}\frac{\partial p}{\partial \eta}\right) + \frac{\partial}{\partial \eta}\left(\dot{\eta}\frac{\partial p}{\partial \eta}\right) - \tau\nabla^4\left(\frac{\partial p}{\partial \eta}\right) = 0. \tag{2.7}$$

The final prognostic equation is the energy equation,

$$\frac{\partial T}{\partial t} + \mathbf{u} \cdot \nabla T + \dot{\eta}\frac{\partial T}{\partial \eta} - \frac{RT_v}{c_p p}\omega = 0, \tag{2.8}$$

where ω is given by

$$\omega = \frac{\mathrm{D}p}{\mathrm{D}t} = \frac{\partial p}{\partial t} + \mathbf{u} \cdot \nabla p. \tag{2.9}$$

The energy equation often requires stabilization, and we can use the same hyperviscosity approach that we do for the momentum equation:

$$\frac{\partial T}{\partial t} + \mathbf{u} \cdot \nabla T + \dot{\eta}\frac{\partial T}{\partial \eta} - \frac{RT_v}{c_p p}\omega - \tau\nabla^4 T = 0, \tag{2.10}$$

2.2 2D Shallow Water Equations

The Williamson standard test set paper [20] gives eight different categories of the shallow water equations, and several different forms within each category. The two preferred forms for our purposes are the advective primitive variable form, and the vorticity divergence form. The advective primitive variable form is given by

$$\frac{\partial \mathbf{u}}{\partial t} = -(\zeta + f)\hat{\mathbf{k}} - \nabla\left(\frac{1}{2}\mathbf{u}^2 + gH\right), \tag{2.11}$$

$$\frac{\partial h}{\partial t} = -\nabla \cdot h\mathbf{u}, \tag{2.12}$$

where g is gravity and $H = h + h_s$ with H the fluid-surface height, h the fluid thickness, and h_s the bottom surface elevation. Hyperviscosity for stabilization gives

$$\frac{\partial \mathbf{u}}{\partial t} = -(\zeta + f)\hat{\mathbf{k}} - \nabla\left(\frac{1}{2}\mathbf{u}^2 + gH\right) + \tau\nabla^4\mathbf{u}, \tag{2.13}$$

$$\frac{\partial h}{\partial t} = -\nabla \cdot h\mathbf{u} + \tau\nabla^4 h, \tag{2.14}$$

The vorticity divergence form of the equations begins with the definitions of vorticity ζ and divergence δ:

$$\zeta \equiv \hat{\mathbf{k}} \cdot (\nabla \times \mathbf{u}), \tag{2.15}$$

$$\delta \equiv \nabla \cdot \mathbf{u}, \tag{2.16}$$

which leads to governing equations

$$\frac{\partial \zeta}{\partial t} = -\nabla \cdot (\zeta + f)\mathbf{u}, \tag{2.17}$$

$$\frac{\partial \delta}{\partial t} = \hat{\mathbf{k}} \cdot \nabla \times (\zeta + f)\mathbf{u} - \nabla^2\left(\frac{1}{2}\mathbf{u}^2 + gh\right). \tag{2.18}$$

2.3 2D x-z Hydrostatic Equations

The hydrostatic x-z equations are simply a 2D version of the 3D hydrostatic equations expressed in a vertical plane. Thus we can take the 3D hydrostatic equations and replace vector velocity $\mathbf{u}$ with scalar velocity u, and 2D derivative operator ∇ with $\partial/\partial x$. This gives momentum equation

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}\left(\frac{1}{2}u^2 + \phi\right) + \dot{\eta}\frac{\partial u}{\partial \eta} + \frac{RT_v}{p}\frac{\partial p}{\partial x} = 0. \tag{2.19}$$

The geopotential and virtual temperature diagnostic equations remain unchanged, and the vertical velocity diagnostic equation becomes

$$\dot{\eta}\frac{\partial p}{\partial \eta} = -\frac{\partial p}{\partial t} - \int_0^{\eta} \frac{\partial}{\partial x}\left(\frac{\partial p}{\partial \eta'}\right) \mathrm{d}\eta'. \tag{2.20}$$

Continuity can be expressed

$$\frac{\partial}{\partial t}\left(\frac{\partial p}{\partial \eta}\right) + \frac{\partial}{\partial x}\left(u\frac{\partial p}{\partial \eta}\right) + \frac{\partial}{\partial \eta}\left(\dot{\eta}\frac{\partial p}{\partial \eta}\right) = 0, \tag{2.21}$$

and the energy equation becomes

$$\frac{\partial T}{\partial t} + u\frac{\partial T}{\partial x} + \dot{\eta}\frac{\partial T}{\partial \eta} - \frac{RT_v}{c_p p}\omega = 0, \tag{2.22}$$

with

$$\omega = \frac{\partial p}{\partial t} + u\frac{\partial p}{\partial x}. \tag{2.23}$$

2.4 Model Implementation Strategy

We first implemented the shallow water equations on the sphere. This allowed us to verify the proper implementation of dynamics in the horizontal direction, and proper handling of velocities at the poles. Note that one of the advantages of the spectral element method used here is that the governing equations are expressed with velocities in the parameterized space, aligned with the axes of the reference element. However, the method requires transformations between this space and spherical coordinates, and velocity vectors in spherical coordinates are undefined at the poles, and derivatives of those quantities become unbounded. So this isolated verification was an important step. Also, there is a standardized suite of test problems for the shallow water equations on the sphere [20].

Next we implemented the *x*-*z* hydrostatic equations, which allowed us to verify the vertical coordinate system. This coordinate system is expressed in normalized pressure (a value of 1 at sea level and 0 at the interface to outer space), and so is non-trivial. There is not a standard set of test problems for this set of equations, so verification was less straightforward.

Finally, we implemented the 3D hydrostatic equations. The *x*-*z* hydrostatic equations have 1D spectral elements in the *x* direction, while the 3D hydrostatic equations have 2D spectral elements in the horizontal direction. Given the algorithm for implementing spectral elements (and finite elements in general), we were able to leverage a significant portion of the software developed for the *x*-*z* equations for use in the 3D equations.

Chapter 3

Numerical Models

The Aeras Next-Generation Global Atmosphere Project is focused largely on demonstrating performance portability and embedded uncertainty quantification. Since the advantages of the spectral element method are well-established for a model of the global atmosphere, the Aeras project will utilize this method. This also provided us with a production-quality baseline to compare against as we introduced next-generation capabilities.

3.1 A Brief History of Spectral Elements for Global Geophysical Flows

In 1995, Iskandarani, Haidvogel and Boyd [6] first applied spectral elements to global geophysical flows with development of the Spectral Element Ocean Model (SEOM). At the National Center for Atmospheric Research (NCAR) in 1997, Taylor and Tribbia in collaboration with Iskandarani [18] used SEOM as a starting point for developing the Spectral Element Atmosphere Model (SEAM). Within the community of global atmosphere model developers, SEAM quickly earned a positive reputation for its combination of accuracy and high efficiency on distributed computer architectures. This was due to SEAM's unstructured grids that allow for load balance and its high-order elements that provide for relatively large amounts of computational work per element that improves the compute-to-communication ratio desired in parallel applications.

In 1999, Taylor left NCAR to work at Los Alamos. NCAR researchers Loft and Thomas decided to develop a new spectral element model for the atmosphere, dubbing it the High-Order Multiscale Modeling Environment (HOMME). This name was inspired in part by the Finite Element Method Modeling Environment (FEMME) code. HOMME boasted cache-blocked algorithms and the introduction of covariant and contravariant velocity transformations, making it a more suitable target than the research code SEAM for inclusion in a production-level climate model. In 2003, Sandia National Laboratories decided to engage in high-performance climate modeling research by funding an LDRD to collaborate with NCAR to fortify HOMME, with the goal of becoming one of the atmosphere models within the Community Earth System Model (CESM) [5]. CESM already had multiple options within the Community Atmosphere Model (CAM): an Eulerian spectral transform method, a semi-Lagrangian spectral transform method, and a finite volume method.

This 2003 LDRD allowed Sandia to hire Taylor to return to his atmospheric modeling research.

The LDRD and follow-on projects were a great success: HOMME was made more robust and stable, and was coupled to the wide array of atmospheric physics parameterizations supported by CAM. The combination of HOMME coupled to physics within the CESM was dubbed CAM-SE (CAM with Spectral Elements), and is now the default atmosphere model in both CESM and the Accelerated Climate Model for Energy (ACME) [1]. ACME, of course, is the Department of Energy climate model that was split off from CESM.

In casual conversation, SEAM, HOMME, and CAM-SE can often be used interchangeably. Technically, SEAM is a research atmosphere dynamical core, HOMME is a production-level atmosphere dynamical core, and CAM-SE is HOMME coupled to atmospheric physics parameterizations and integrated into a full climate model. Aeras compares most directly to HOMME, as Aeras is not coupled to physics.

3.2 HOMME Numerical Approximations in Aeras

The governing equations outlined in section 2.1 include different types of derivatives, represented by ∇ in the horizontal spatial direction (parallel to the surface of the sphere), $\partial/\partial\eta$ in the vertical spatial direction (radial from the center of the earth), and $\partial/\partial t$ in the time dimension. In HOMME, and therefore Aeras, these derivatives are all discretized differently:

- Horizontal derivatives use spectral elements
- Vertical derivatives use finite differences
- Time derivatives use Runge-Kutta time stepping

The specific discretization methods employed by Aeras are described in the following sections. For a discussion of the background of these methods and further details, see [17].

3.2.1 Horizontal Spectral Elements

For the spectral element method utilized for HOMME and Aeras, the horizontal dimensions are discretized into quadrilaterals suitable for use as spectral elements. While the method can utilize any tiling of the surface of the sphere into quadrilaterals, the method employed by this project is the "cubed sphere," in which the surface of a cube is decomposed into quadrilaterals and projected onto the surface of a sphere. An example of such a grid is presented in Figure 3.1, for the case in which 30 elements are specified along each edge of the cube. This approach for grid generation results in a mesh with roughly equivalent cell sizes, which is considered desirable for climate simulations and subgrid physics performance.

Each quadrilateral on the surface of the sphere is then interpreted as a spectral element, enriched with a tensor product of nodes arranged using a Gauss-Lobatto distribution. In practice, these

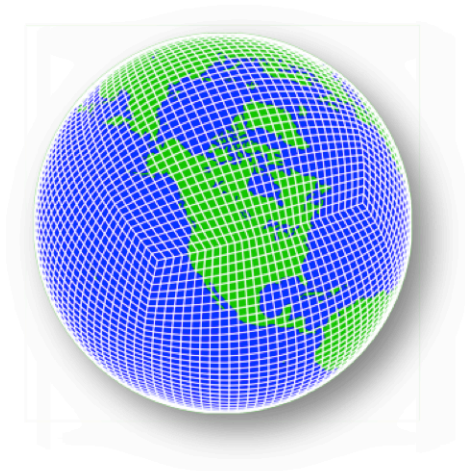

Figure 3.1. A "cubed sphere" grid, in which the surface of a cube is decomposed into quadrilaterals and projected onto the surface of a sphere. This is an *NE30* grid, indicating that the number of elements along one side of the cube equals 30.

nodes are located precisely using a reference element, as depicted in Figure 3.2, and projected from the cube along with the corner nodes.

The handling of velocity components within this system of spectral elements in non-trivial. Input and output for Aeras is handled in spherical coordinates, as this is the most natural system for users. Mathematically, spherical coordinates are undesirable, because spherical velocity vectors are undefined (or multivalued) at the poles and their derivatives are unbounded. One of the advantages of the spectral element method for the global atmosphere is that local computations can be performed in well-behaved "parametric" coordinates, thus avoiding the "pole problem." Computations are performed using velocities from the parametric coordinate system, aligned with the reference element. To transfer between spherical and parametric coordinate systems, covariant and contravariant vector bases are introduced as described below.

Spherical $(\mathbf{r}_\lambda, \mathbf{r}_\theta)$ a global vector basis independent of the mesh, locally orthogonal, and multivalued at poles.

Covariant $(\mathbf{g}_1, \mathbf{g}_2)$ a local vector basis aligned with a shell element, non-orthogonal, with components orthogonal to contravariant components. Bases are given by

$$\mathbf{g}_1 = \frac{\partial \mathbf{f}}{\partial \xi_1}, \mathbf{g}_2 = \frac{\partial \mathbf{f}}{\partial \xi_2},$$

where $\mathbf{f}$ is the mapping from the reference element to the shell element on the sphere.

Contravariant $(\mathbf{g}^1, \mathbf{g}^2)$ a local vector basis aligned with a shell element, non-orthogonal, with components orthogonal to covariant components. Bases are given by

$$\mathbf{g}^1 = \frac{\partial \mathbf{f}^{-1}}{\partial \lambda}, \mathbf{g}^2 = \frac{\partial \mathbf{f}^{-1}}{\partial \theta},$$

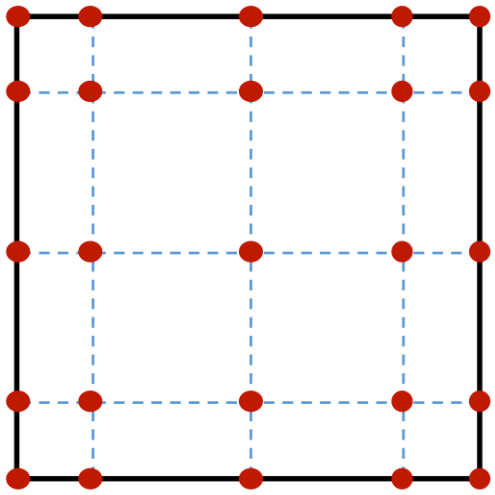

Figure 3.2. A reference *NP5* spectral element, where *NP* refers to the number of points on one side of the element. The node points along each side are arranged using a Gauss-Lobatto distribution.

where $\mathbf{f}^{-1}$ is the mapping from the shell element on the sphere to the reference element.

Parametric (ξ_1, ξ_2) a local vector basis aligned with the reference element, orthogonal.

The mapping functions $\mathbf{f}$ and $\mathbf{f}^{-1}$ can be obtained from the coordinates of the physical shell elements and the known coordinates of the reference element, and can take advantage of the high-order basis functions of the spectral elements.

3.2.2 Vertical Finite Differences

In a shallow water model, topography can be represented as a function of space; however, in the 2D *x*-*z* hydrostatic and the 3D hydrostatic models, topography represents the lower boundary of the vertical coordinate. Modelers may choose from several different vertical coordinate definitions. The simplest choices of constant pressure surfaces or constant height surfaces make it easy to express hydrostatic balance,

$$\frac{\partial p}{\partial z} = -\rho g, \tag{3.1}$$

but face the same difficulty in the presence of variable topography. The $z = 0$ or $p = p_s$ surface intersects land. Other horizontal isosurfaces of these variables can intersect the topography as well; Albuquerque, for example, lies above the 1500 m and 900 hPa surfaces. These intersections complicate the formulation of the lower boundary condition. HOMME and Aeras use the hybrid η-coordinate system due to Simmons and Burridge [15], which combines the advantages of a terrain-following coordinate system near the surface (the lower boundary, topography included, corresponds to $\eta = 1$) with a horizontal pressure coordinate at higher altitudes.

The hybrid η coordinate system used by Aeras is illustrated by figure 3.3. The surface topography is indicated by the red line; model levels are shown as solid black lines and interfaces between levels are shown as dashed lines. An example temperature profile (defined in section 3.5.2, below) is indicated by the background coloring. In this system, model levels and interfaces near the surface conform to the underlying topography as in a terrain-following σ-coordinate system. Near the model top, the model levels approach a horizontal pressure surface as in a p-coordinate system. Pressure along a model level is given by

$$p(\eta) = A(\eta)p_0 + B(\eta)p_s(\mathbf{x},t), \tag{3.2}$$

where $p_0 = 1000$ hPa is a reference pressure, $p_s(\mathbf{x},t)$ is the surface pressure, and $A(\eta)$, $B(\eta)$ are the constant hybrid coefficients defined such that

$$\eta = A(\eta) + B(\eta). \tag{3.3}$$

Aeras allows users to define the coefficients $A(\eta)$ and $B(\eta)$, subject to vertical boundary conditions $B(\eta_{top}) = 0$ and $A(1) = 0$, $B(1) = 1$. This ensures that the surface ($\eta = 1$) corresponds to the lower boundary and that the model top $\eta = \eta_{top}$ corresponds to the horizontal surface $p = p_{top}$.

The vertical discretization is defined by the hybrid coefficients $A(\eta)$ and $B(\eta)$ of each model level $l = 1,2,\ldots,L$. Level interfaces are indexed by $l+1/2$, $l = 0,1,\ldots,L$, so that index 1/2 corresponds to the model top, $\eta_{1/2} = \eta_{top}$, and interface $L+1/2$ is the surface $\eta_{L+1/2} = 1$. Levels 1 and L are the uppermost and lowermost model levels, respectively. Figure 3.3 shows an example with $L = 15$. The model levels and interfaces provide the vertical centered finite difference operators used by Aeras. For model variable X at model level l, the discrete operator δ_η approximates vertical derivatives at model levels from the adjacent interfaces,

$$\frac{\partial X}{\partial \eta}(\eta_l) \approx \delta_\eta(X_l) = \frac{X_{l+1/2} - X_{l-1/2}}{\eta_{l+1/2} - \eta_{l-1/2}}. \tag{3.4}$$

When required, values at level interfaces are computed by vertical averaging,

$$X_{l+1/2} = \frac{1}{2}\left(X_l + X_{l+1}\right). \tag{3.5}$$

Further details and discussion are provided by [17].

3.2.3 Runge Kutta Time-Stepping

The HOMME code has a variety of time-stepping options, which include the leap-frog scheme, low-storage Runge-Kutta (RK) schemes of various orders, some of which are Strong Stability-Preserving (SSP), and the trapezoidal rule. In 3D dynamics, HOMME typically uses the Kinnmark and Gray 5-stage, 3rd order RK scheme [8]. Tracers typically use a 3-stage second order SSP RK method.

For the numerical examples summarized in Section 3.5, the following time-integration schemes were used:

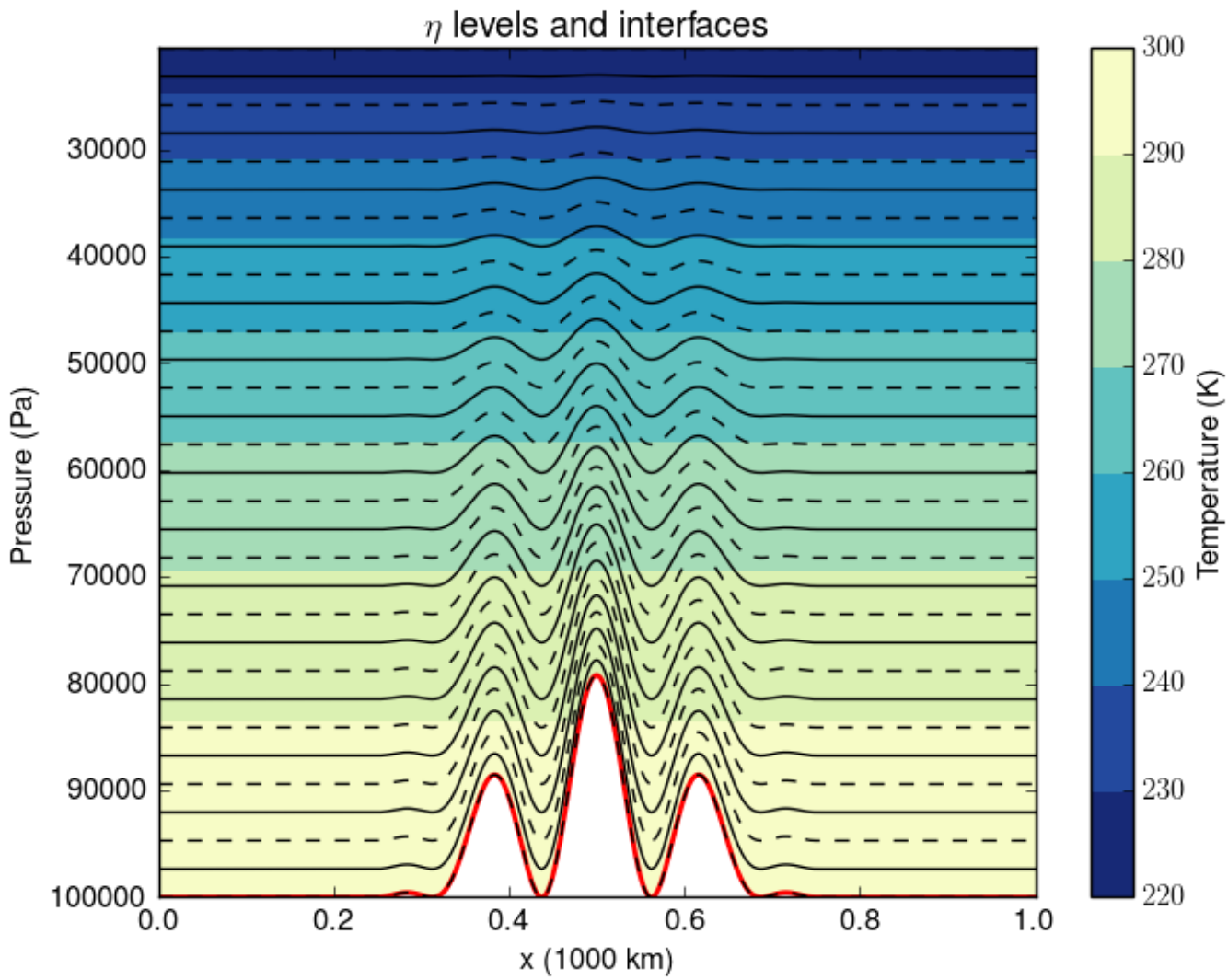

Figure 3.3. Illustration of the hybrid vertical coordinates used by Aeras. 15 model levels (solid black lines) and 16 interfaces (dashed lines) define the vertical coordinate system.

- Explicit trapezoidal rule (2nd order).
- Explicit 4-state, 3rd order RK method.
- Explicit 4-state RK method (4th order).

All schemes are explicit, and hence conditionally stable, with stability governed by the Courant-Friedrichs-Lewy (CFL) condition. For more detail on these schemes, including the Butcher tableaus that define them and their stability regions, the reader is referred to [11]. When performing code-to-code comparisons between the Aeras and HOMME codes, it was ensured that the same time-integration scheme was used in both codes with the same time step.

3.3 Albany

Described in more detail in [14], Albany is a PDE code written using the component-based approach to code development. The crux of this approach, which we have branded *Agile Components*, involves the accumulation of components across four classes of software: libraries, interfaces, software quality tools, and demonstration applications. These components form the foundation for new codes. The benefits of *Agile Components* strategy are numerous. For example, having a large collection of world-class algorithmic capabilities as a foundation for attacking new applications provides a large advantage over starting from scratch or retro-fitting a monolithic code that was designed for a different class of problems.

Albany is built almost entirely from functionality in reusable libraries from the Trilinos project. At the highest level, the FEM assembly in Albany is based on Trilinos Phalanx [13] package. Phalanx, specifically designed to solve general partial differential equations, decomposes a complex problem into a number of simpler problems (*evaluators*) with managed dependencies. Each *evaluator* encodes the variables it depends upon, the variables it evaluates, and the code to actually evaluate the desired term. Phalanx then assembles all of the evaluators for a given problem into a directed acyclic graph. Figure 3.4 shows a dependency graph created by Aeras for the shallow water equations that represents the full PDE residual evaluation for a given set of mesh cells stored in a data structure called the field manager.

Albany supports a wide variety of application physics areas including heat transfer, fluid dynamics, structural mechanics and plasticity (part of the LCM computational suite), quantum device modeling (part of the QCAD simulator) and climate modeling (the Aeras atmospheric dynamical core and FELIX land-ice dynamical core). Albany is also home to several algorithmic projects that contribute to the code's overall infrastructure and capabilities, e.g., embedded uncertainty quantification, adaptive mesh refinement, and model order reduction.

The main goal of this work is to create an architecture-portable version of Albany's Finite Element Assembly using the Kokkos library. We describe our refactoring process in the next chapter. Although this report focuses on the specific case of the Albany code, we emphasize that the refactoring approach we describe is general: it can be employed to create performance-portable

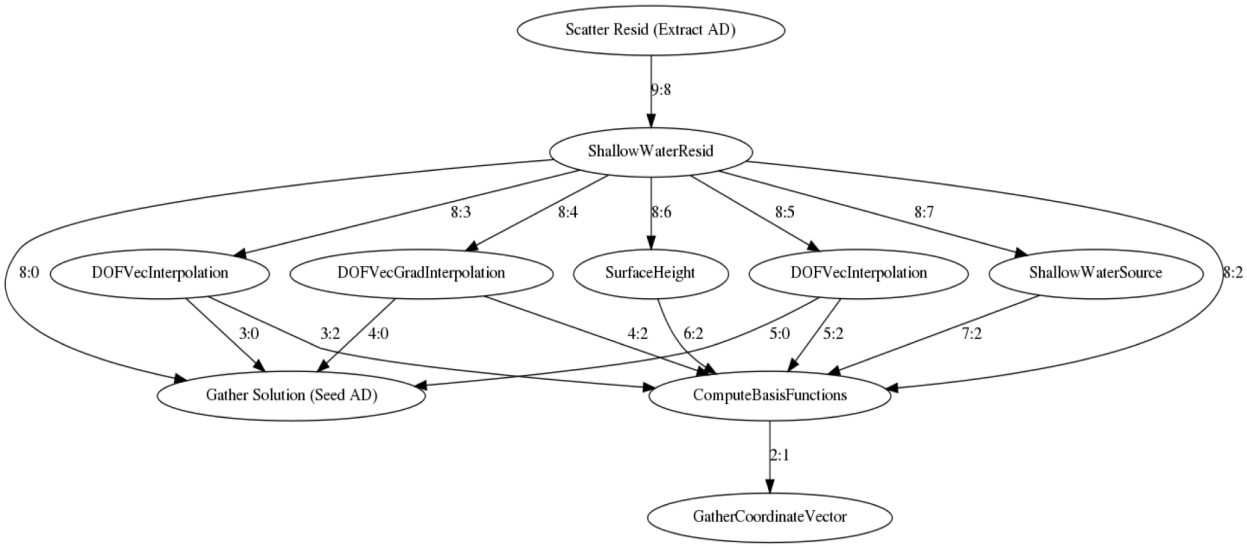

Figure 3.4. Shallow water graph of the residual evaluator for the Aeras global atmosphere model in Albany.

implementations of virtually any other finite element code.

3.4 Extensions to Albany

At the beginning of the Aeras project, Albany had already been utilized to enable a wide variety of applications, ranging from fluid flow to quantum devices. Nevertheless, Albany did not support all of the capabilities required of a global atmosphere model. The Aeras team added the following capabilities to Albany, and these capabilities are available to any other Albany application that might need them:

- Shell elements
- Spectral elements
- Efficient explicit time-stepping
- Additional explicit time-stepping methods
- Concurrent samples
- Embedded UQ for transient problems
- Spherical coordinate system transformations
- Atmospheric column data structures

These extensions are discussed in more detail below.

Shell elements For the 2D shallow water equations, and even the 3D hydrostatic equations, the spectral elements utilized by Aeras are logically two dimensional, but conform to a three dimensional surface. To support this, Albany had to be extended to recognize shell elements.

Spectral elements Albany uses the Trilinos package Intrepid [2] to support specific finite element types, and Intrepid includes a general interface that can be utilized to enable spectral elements. However, the Trilinos package STK (Sierra Toolkit), used to read and store meshes, does not support spectral elements. To work around this problem, Aeras reads in a shell element mesh of bi-linear quadrilaterals, and extends each element to a desired degree of accuracy by enriching the element with a tensor product of nodes arranged according to a Gauss-Lobatto distribution. An algorithm was designed such that this enrichment is done in parallel, ensuring that shared nodes on different processors receive the same global ID without the need for communication. Output to STK format was accomplished by interpreting each spectral element as a patch of bi-linear quadrilaterals.

Efficient explicit time-stepping Albany was designed with an assumption that all problems would have a non-trivial mass matrix (as is common for low-order finite element formulations) that would have to be solved implicitly. Thus, a significant amount of the Albany code base is dedicated to solving implicit problems efficiently. However, Aeras utilizes spectral elements coupled with an under-integration of the quadrature used to compute matrix elements, specifically chosen to result in a diagonal mass matrix. Albany handled this inefficiently, because it assumed that off-diagonal matrix elements were non-zero and thus needed to be both calculated and included in matrix-vector product computations. These assumptions had to be corrected to achieve efficient explicit time stepping.

Additional time-stepping methods Albany utilizes the Rythmos package [11] of Trilinos for time-stepping. Although the same Kinnmark and Gray 5-stage, 3rd order RK scheme that is implemented in HOMME was added to the Rythmos package as a part of the Aeras project, it was not employed to generate the results presented in this report.

Concurrent samples This is the capability to run multiple simulations (as would be needed for uncertainty quantification sampling methods) simultaneously, and reap certain performance improvements. This represents a new area of research, and is detailed in chapter 5. Albany's rich template-based coding environment was a key factor towards implementing concurrent samples.

Embedded UQ for transient problems Albany supported embedded UQ and computations of sensitivities for steady problems, but for Aeras, this capability had to be extended to transient problems.

Spherical coordinate system transformations Both HOMME and Aeras support multiple coordinate systems for velocity vectors. First is the spherical coordinate system, in which the local velocity components are aligned with the local longitude and latitude. This coordinate system is the most natural for users to interact with, and so all I/O and certain other

operations are conducted in this system. However, spherical vector components present serious mathematical problems at the poles, and so are undesirable for most computations. These computations are typically done in reference element-aligned coordinates, in which the velocity unit vectors are aligned with a square unit reference element. Two other velocity coordinate systems, covariant and contravariant, are used in transforming between spherical and reference coordinate systems. While these transformations seem specific to atmosphere models, the FELIX team (a land ice model also built on top of Albany) has expressed interest in using them in order to accurately capture the curvature of Antarctica and Greenland.

Atmospheric column data structures The HOMME and Aeras data structures for field data are hybrid in nature: spectral elements in the horizontal direction and columns suitable for finite differencing in the vertical direction, with a parallel decomposition in the horizontal direction only. This requires a unique data structure that is different from what Albany supports, as well as new evaluators for certain algorithms such as gather and scatter operations surrounding communication. While the specific changes made in this respect are unlikely to directly benefit another application, the changes we made do provide a template for other applications that may require specialized data structures.

3.5 Model Verification

Significant time and resources were expended verifying that the developed models were producing accurate results. The following sections detail this verification process.

3.5.1 Shallow Water Model

The shallow water models on the sphere provide a 2D simplification of the 3D hydrostatic equations that allow developers to focus on the horizontal discretization method and demonstrate how the method handles the pole problem. In 1992, the community banded together to prescribe a set of test cases for the shallow water equations [20]. This suite of problems included seven test cases of progressing difficulty. Over the years, some of these test cases have fallen out of favor, while others have risen to prominence. Currently, it is expected that new models implement test cases 1, 2, 5 and 6.

Test Case 1: Advection of a Cosine Bell

Test case 1 is a purely advective problem. That is to say, the velocity field is specified, and the momentum equation is ignored, leaving only the height equation to be solved. The advecting wind

is given by

$$u = u_0(\cos\lambda\cos\alpha + \sin\theta\cos\lambda\sin\alpha), \tag{3.6}$$
$$v = -u_0\sin\lambda\sin\alpha, \tag{3.7}$$

where λ is longitude, θ is latitude, and α is the angle between the axis of solid body rotation and the polar axis of the spherical coordinate system. The initial height field specifies a cosine bell shape to be advected:

$$h(\lambda,\theta) = \begin{cases} (h_0/2)(1+\cos(\pi r/R)), & \text{if} \quad r < R \\ 0, & \text{if} \quad r \geq R, \end{cases} \tag{3.8}$$

where $h_0 = 1000$ m and r is the great circle distance between (λ,θ) and the center of the cosine bell, initially taken as $(\lambda_c,\theta_c) = (3\pi/2, 0)$:

$$r = a\arccos[\sin\theta_c\sin\theta + \cos\theta_c\cos\theta\cos(\lambda-\lambda_c)]. \tag{3.9}$$

The radius $R = a/3$ and the advecting wind velocity $u_0 = 2\pi a/(12 \text{ days})$. This set of conditions will advect the cosine bell around the earth once in 12 days. See Figure 3.5 for a plot of the solution at the end of the simulation.

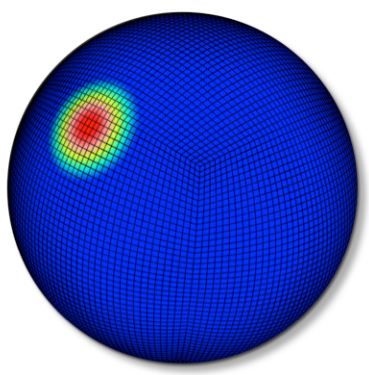

Figure 3.5. Height results for shallow water model test case 1.

Test Case 2: Global Steady State Nonlinear Zonal Geostrophic Flow

This case is a steady state solution to the non-linear shallow water equations. It consists of solid body rotation (or zonal flow) with the corresponding geostrophic height field. The Coriolis parameter is a function of latitude and longitude so the flow can be specified with the spherical coordinate poles not necessarily coincident with earth's rotation axis. The steady solution is given by initial conditions

$$u = u_0(\cos\theta\cos\alpha + \cos\lambda\sin\theta\sin\alpha), \tag{3.10}$$
$$v = -u_0\sin\lambda\sin\alpha, \tag{3.11}$$
$$gh = gh_0 - \left(a\Omega u_0 + \frac{u_0^2}{2}\right)(-\cos\lambda\cos\theta\sin\alpha + \sin\theta\cos\alpha)^2, \tag{3.12}$$

and the Coriolis parameter is recast as

$$f = 2\Omega(-\cos\lambda\cos\theta\sin\alpha + \sin\theta\cos\alpha). \tag{3.13}$$

Figure 3.6 is a plot of the height field for this steady-state problem.

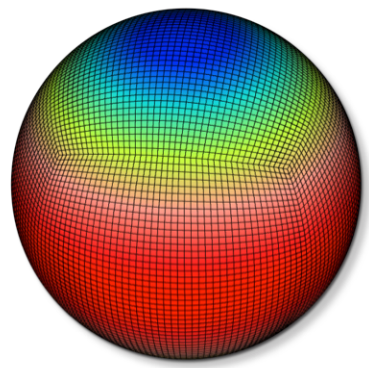

Figure 3.6. Height results for shallow water model test case 2.

Figure 3.7 is a plot of relative L^2 errors of the solution and maximum value of the shallow water test case 2 solution, varying element degree from 2 to 13. The solution error is smooth and exponential, and the maximum value error is largely smooth with the exception of glitches at $p = 7$ and $p > 11$. Given that the maximum value error is a much harder norm to achieve, and that the higher p behavior is likely due to machine precision, we conclude that the variable p capabilities of Aeras are correctly implemented.

In Figure 3.8, we vary NE, the number of elements along each side of the cube from 2 to 16, and plot the relative L^2 error for each p from 2 to 10. Each curve converges as the degree of accuracy would predict.

Test Case 5: Zonal Flow over an Isolated Mountain

Test case 5 has no analytic solution, but there are high-resolution spectral solutions available for comparison. This test starts with zonal flow as in test case 2 with $\alpha = 0$, $h_0 = 5960$ m and $u_0 = 20$ m/s that impinges on a mountain. The surface or mountain height is given by

$$h_s = h_{s0}(1 - r/R), \tag{3.14}$$

where $h_{s0} = 2000$ m, $R = \pi/9$, and $r^2 = \min[R^2, (\lambda - \lambda_c)^2 + (\theta - \theta_c)^2]$, and the center of the conical mountain is given as $\lambda_c = 3\pi/2$ and $\theta_c = \pi/6$.

Even without an analytical solution, test case 5 is a favorite example simulation to run because of its relative simplicity and the complexity of the flow field it generates. This makes it a reasonable test case for performance studies, as most of the shallow water results in this report are. Figure 3.9 shows longitudinal velocities for days 0, 5, 10 and 15. Figures 3.10 – 3.12 are all for test case 5 at

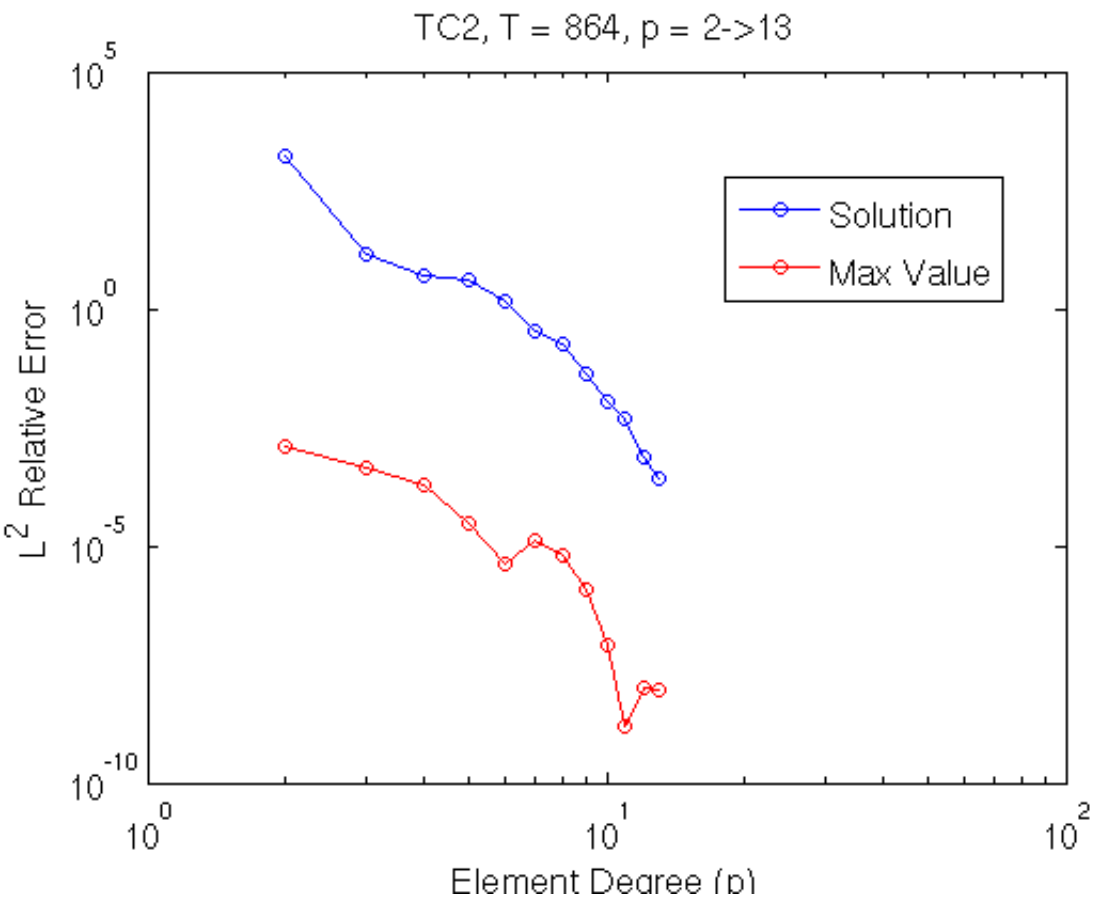

Figure 3.7. Relative errors under p refinement for shallow water model test case 2 at $T = 864$.

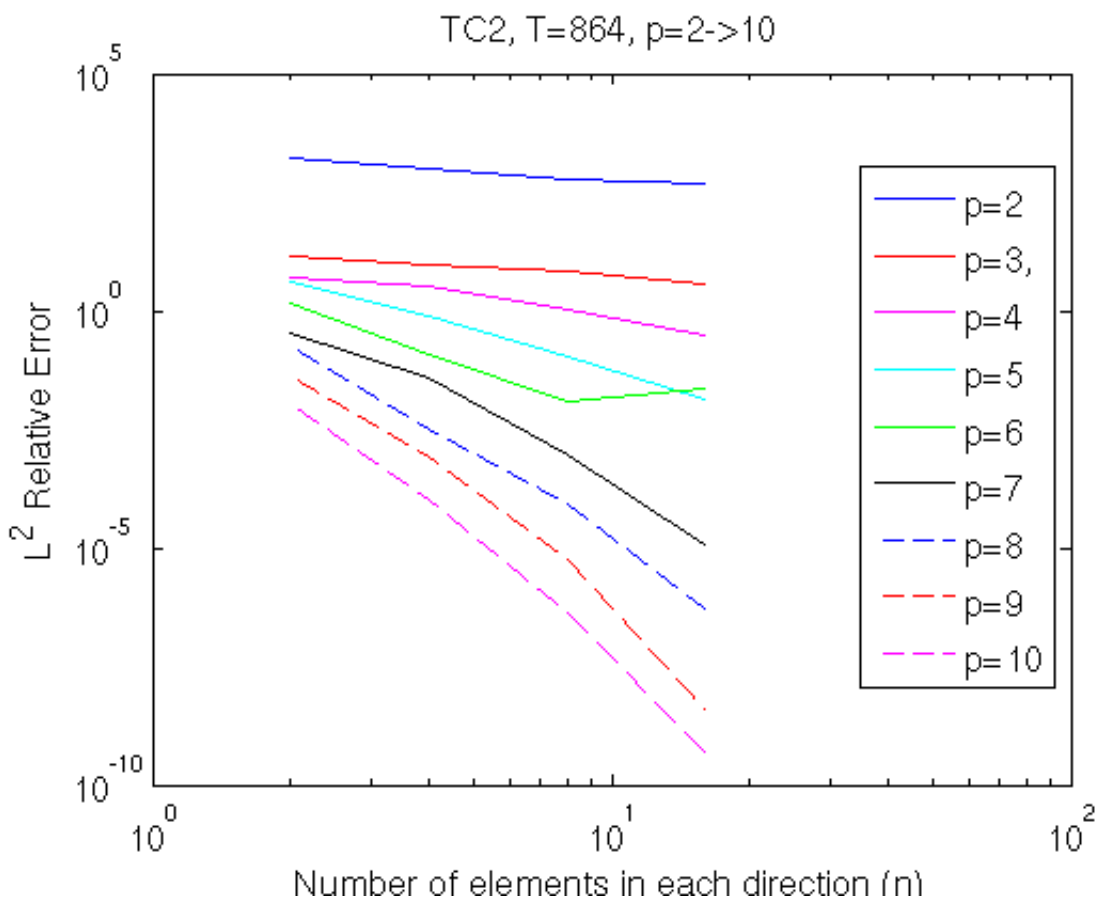

Figure 3.8. Height convergence study under *h* refinement for shallow water model test case 2 at various values for *p*.

day 5: Figure 3.10 shows an implicit solution (not typically obtained for a shallow water model, but achievable with no extra effort here because Albany has strong support for implicit solvers); Figure 3.11 provides a more common explicit solution; and Figure 3.12 demonstrates sensitivities utilizing Albany's built-in uncertainty quantification capabilities.

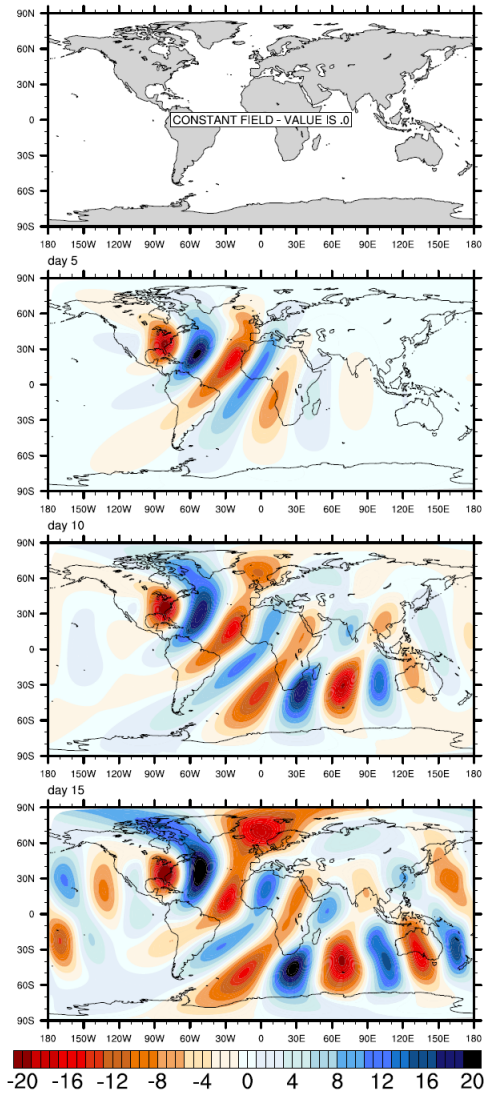

Figure 3.9. Longitudinal velocities in meters per second for Aeras Shallow Water TC5 at the following times, from top to bottom: Initial time, Day 5, Day 10, Day 15.

Test Case 6: Rossby-Haurwitz Wave

The Rossby-Haurwitz waves of test case 6 are analytic solutions of the nonlinear barotropic vorticity equation on the sphere, but not analytic solutions of the shallow water equations. A specification

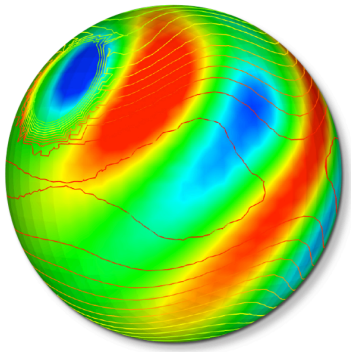

Figure 3.10. Implict results for shallow water model test case 5. Color contours show latitudinal velocity, line contours show height field.

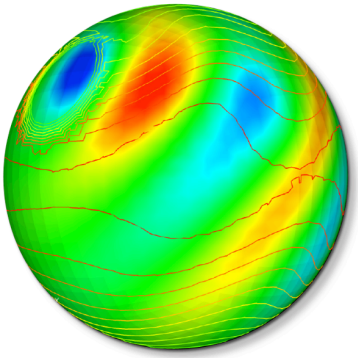

Figure 3.11. Explict results for shallow water model test case 5. Color contours show latitudinal velocity, line contours show height field.

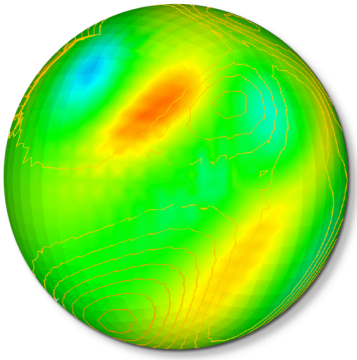

Figure 3.12. Sensitivities with respect to mountain height, for shallow water model test case 5. Color contours show latitudinal velocity sensitivities, line contours show height field sensitivities.

of the stream function provides the initial velocity field:

$$\psi = -a^2\omega\sin\theta + a^2K\cos^R\theta\sin\theta\cos R\lambda, \tag{3.15}$$

where ω, K, and R are constants. This configuration runs west to east while maintaining its shape with angular velocity

$$v = \frac{R(3+R)\omega - 2\Omega}{(1+R)(2+R)}. \tag{3.16}$$

See [20] for full implementation details. The model is typically run with $\omega = K = 7.848\times10^{-6}\mathrm{s}^{-1}$ and $h_0 = 8\times10^3$ m, with wave number $R = 4$. Figure 3.13 shows results for the height field for test case 6.

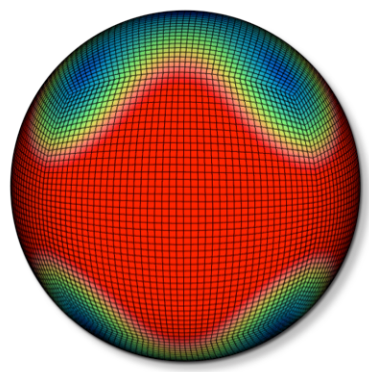

Figure 3.13. Height results for shallow water model test case 6.

3.5.2 3D Hydrostatic Model

In 2012 NCAR hosted the second Dynamical Core Model Intercomparison Project (DCMIP). The project had two primary goals: 1) to define a standard set of test cases for 3D dynamical cores [19] and 2) to document the results of the current state-of-the-art models (both research and operational) on those test cases. In the years following the project the test cases have come to provide an similar baseline for verifying 3D models as the test cases of [20] provide for the shallow water equations.

Advection in 3D deformational flow

An advection test case does not test the full 3D hydrostatic equation solver; rather, it tests the underlying discretizations of the model using a prescribed velocity field. Consequently, advection tests are often completed first to verify the implementation of a model's numerical method. We use test case 1-1 from [19] for this purpose. A passive tracer, defined as a pair of cosine bells,

$$q(\lambda,\theta,z) = \frac{1}{2}(2+\cos(\pi d_1)+\cos(\pi d_2)), \tag{3.17}$$

is specified at $t = 0$ using the distance functions,

$$d_i(\lambda,\theta,z) = \min\left[1, \left\{\frac{r_i(\lambda,\theta)^2}{R_t^2} + \frac{(z-z_c)^2}{Z_t^2}\right\}\right], \text{ for } i = 1,2, \tag{3.18}$$

where $z_c = 5000$ m is the vertical center of the initial tracer and $r_i(\lambda,\theta)$ is the horizontal great circle distance from the cosine bell centers at $\lambda_1 = 5\pi/6$ and $\lambda_2 = 7\pi/6$ along the equator,

$$r_i(\lambda,\theta) = a\arccos(\cos\theta\cos(\lambda-\lambda_i)), \text{ for } i = 1,2. \tag{3.19}$$

The constants $R_t = a/2$ and $Z_t = 1000$ m define the horizontal and vertical half widths of the cosine bells.

The velocity field is a deformational flow in all three dimensions: zonal, meridional, and vertical. The flow is reversible with period t_p so that the exact solution at $t = t_p$ matches the initial conditions. Velocity components are given by

$$u(\lambda,\theta,p,t) = k\sin^2(\lambda - 2\pi t/t_p)\sin(2\theta)\cos(\pi t/t_p) + \frac{2\pi a}{t_p}\cos\theta \tag{3.20}$$

$$+ \frac{w_0 a}{b p_{top}}\cos(\lambda - 2\pi t/t_p)\cos^2\theta\cos(2\pi t/t_p)\left[-\exp\left(\frac{p-p_0}{b p_{top}}\right) + \exp\left(\frac{p_{top}-p}{b p_{top}}\right)\right], \tag{3.21}$$

$$v(\lambda,\theta,p,t) = k\sin(2(\lambda - 2\pi t/t_p))\cos\theta\cos(\pi t/t_p), \tag{3.22}$$

$$\dot{\eta}(\lambda,\theta,\eta,t) = \frac{w_0}{p_0}\sin(\lambda - 2\pi t/t_p)\cos\theta\cos(2\pi t/t_p)s(p), \tag{3.23}$$

where $s(p)$ is a tapering function that causes the vertical velocity to smoothly decay toward zero near the upper and lower boundaries,

$$s(p) = 1 + \exp\left(\frac{p_{top}-p_0}{b p_{top}}\right) - \exp\left(\frac{p-p_0}{b p_{top}}\right) - \exp\left(\frac{p_{top}-p}{b p_{top}}\right). \tag{3.24}$$

Constants not previously defined in the above formulation are the velocity magnitude $k = 10a/t_p$ m/s, the test case period $t_p = 12$ days, i the vertical pressure velocity magnitude $w_0 = 230\pi/t_p$ hPa/s, a normalization parameter $b = 0.2$, and the pressure at the model top $p_{top} = 254.944$ hPa. For the complete test case derivation, see [19, ch. 1]. While the test definition called for 60 levels, at this point attempts to run Aeras with 60 levels were unsuccessful. The timestep size used for both grids was 20s and the coefficient for hyperviscosity was $1e16$. An attempt was made to reduce the hyperviscosity coefficient to $1e15$, however, this proved to be unstable. Estimated errors are presented in Table 3.1.

Aeras results are shown in the following figures which may be compared with the CAM-SE results available from the DCMIP 2012 web page at `https://earthsystemcog.org/projects/dcmip-2012/cam-se`. Figures 3.14 and 3.15 show the effects of increasing horizontal and vertical resolution within the Aeras code.

Generally, the Aeras results show more numerical smoothing than the CAM-SE results for the same resolutions; we attribute this to the fact that in Aeras the hyperviscosity operator is applied

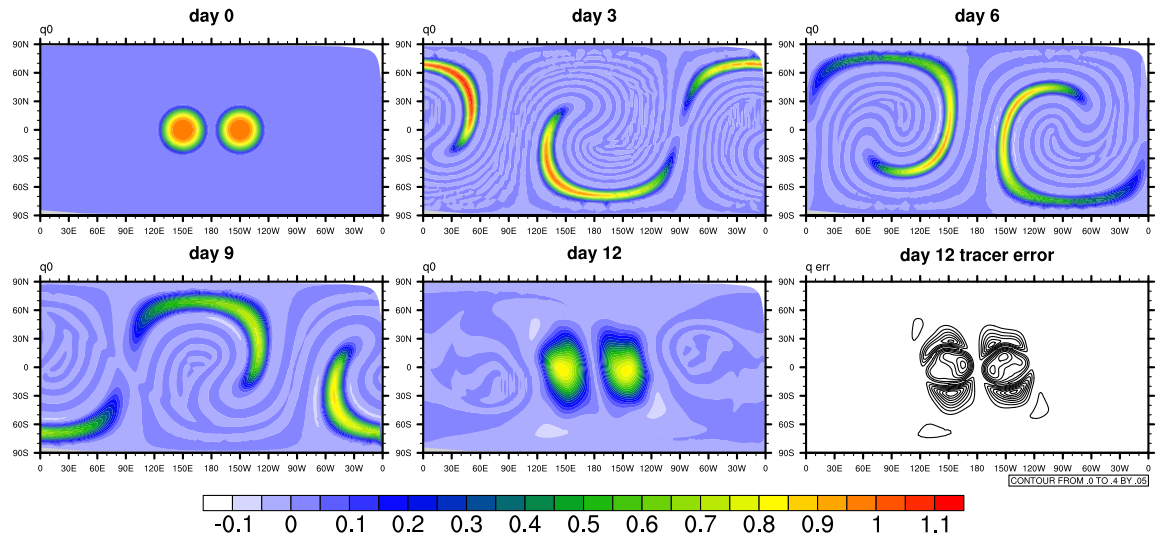

Figure 3.14. 3D advective results for deformational tracer with $\approx 5^\circ$ horizontal resolution and 30 vertical levels.

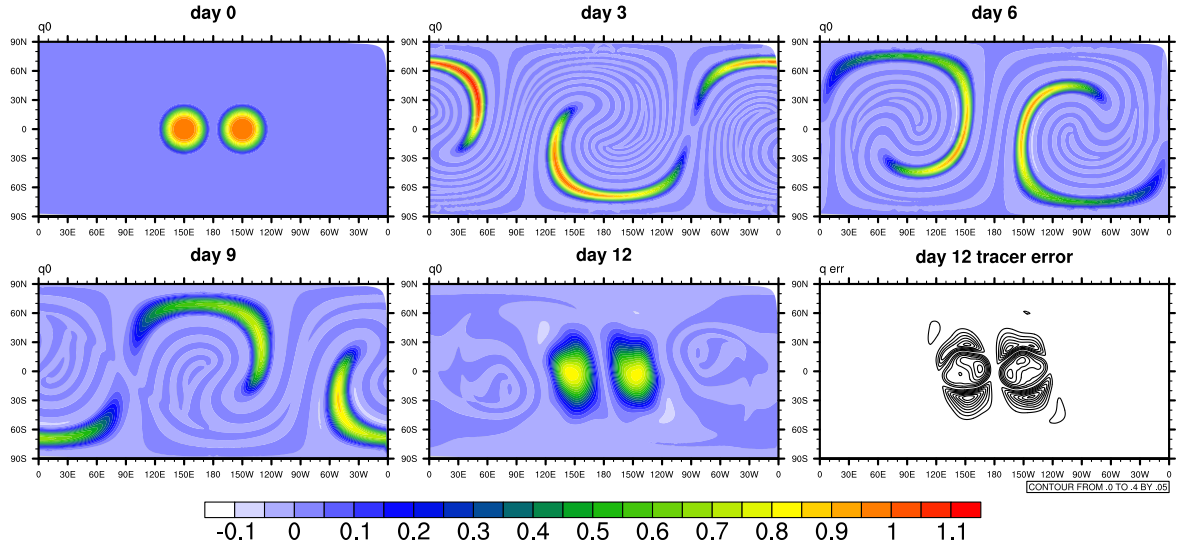

Figure 3.15. 3D advective results for deformational tracer with $\approx 1^\circ$ horizontal resolution and 30 vertical levels.

Simulator	Horizontal Resolution	Vertical Resolution	error ($\|\|l\infty\|\|$)
CAM-SE	1°	60	0.294822
Aeras	5°	15	0.441523
Aeras	5°	30	0.455426
Aeras	1°	15	0.433306
Aeras	1°	30	0.445020

Table 3.1. Estimated errors for 3D deformational flow.

at each stage of the time stepping scheme, while in CAM-SE hyperviscosity is only applied once per timestep. Nevertheless, the day 12 Aeras results are within $\pm 10\%$ of the other spectral element models in DCMIP which serves to validate the implementation.

Resting atmosphere over steep topography

In this test we exercise Aeras in the full solution of the 3D hydrostatic equations. The initial conditions correspond to an atmosphere at rest with a stably stratified vertical profile. Additionally, for this test case we turn off the rotation of the sphere. The combination of no rotation and stable stratification creates a situation with no physical sources of motion. Any nonzero velocity or other departures from the initial conditions are the result of numerical error.

Similarly to the advection test cases, this test case can be used to verify that the code produces a correct result. However, we can also add additional complexity to this test by adding steep orography. In the presence of steep orography, the model levels near the surface depart from the horizontal and the components of the velocity vector $\vec{u} = [u, v, \dot{\eta}]^T$ are no longer orthogonal. Hence, with this test we can assess the numerical errors caused by the terrain-following hybrid coordinate system.

Hydrostatic stability is achieved by prescribing a linear temperature profile,

$$T(z) = T_0 + \Gamma z, \tag{3.25}$$

where the constant lapse rate $\Gamma = \partial T/\partial z = -0.0065$ K/m ensures that potential temperature increases with height. Orography is defined by the surface height function,

$$z_s(\lambda, \theta) = \begin{cases} \frac{h_0}{2}\left(1 + \cos(\pi r_m(\lambda,\theta)/R_m)\right)\cos^2(\pi r_m(\lambda,\theta)/R_t) & \text{if } r_m < R_m \\ 0 & \text{otherwise} \end{cases}, \tag{3.26}$$

where r_m is the great circle distance from the mountain's peak,

$$r_m(\lambda, \theta) = a \arccos(\cos\theta \cos(\lambda - 3\pi/2)), \tag{3.27}$$

with constants maximum height $h_0 = 2000$m, mountain radius $R_M = 3\pi/4$, oscillation half-width $R_t = \pi/16$. The model top is set at $p_{top} = 205.448$ hPa with reference pressure $p_0 = 1000$ hPa corresponding to the reference temperature $T_0 = 300$ K. The surface height, equation (3.26), and temperature (3.25), may be converted from z-coordinates to pressure p-coordinates using the hydrostatic equation (3.1) and then to the corresponding η-coordinate representation. For details, see

[19, ch. 2]. The temperature profile (3.25) and surface height (3.26) were used to create figure 3.3, which illustrates the η-coordinate system.

The left column of figure 3.16 shows the exact steady-state solution of this test case at the surface and the model top for two variables. The upper left panel provides a visualization of the surface pressure p_s. One hemisphere has nearly flat orography while one hemisphere has a radially symmetric mountain range. The lower right panel shows the temperature at model level 1, the uppermost model level. This η-surface is nearly horizontal and we do not expect to see much temperature variation since the temperature profile (3.25) is constant along horizontal surfaces. Examining the color bar, we see that the temperature is in fact nearly constant at a value just over 227 K.

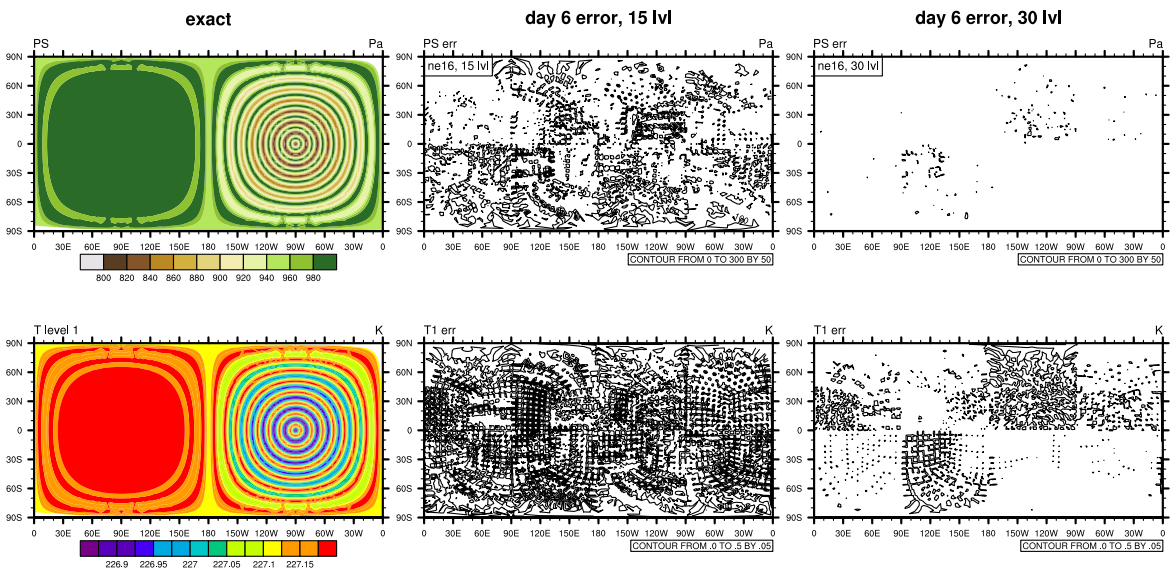

Figure 3.16. 3D hydrostatic results for resting atmosphere under vertical refinement.

Since this is a steady-state problem, the initial conditions can be used to compute error for any $t > 0$. The second and third columns of the figure show the error at day 6 for a simulation using 15 vertical levels (second column) and 30 vertical levels (third column) on a mesh with approximately 5° horizontal resolution. The error magnitude is at the level of truncation error and comparing the data associated with columns 2 and 3, we confirm the expected convergence rate (2nd order) of the vertical finite difference scheme. This test case therefore verifies the Aeras 3D hydrostatic solver and the implementation of the vertical η-coordinate and its associated discrete operators.

Baroclinic instability

Perhaps the most well-known 3D dynamical core test case for the sphere is the perturbed baroclinic instability test originally from [7] and included as test case 4-1 from [19]. This test is easy to set up since its initial conditions are given as functions; however, it has no analytic solution for $t > 0$.

discretization	Δt	τ
ne16, 15 levels	100 s	1×10^{15}
ne16, 30 levels	100 s	1×10^{15}
ne30, 15 levels	20 s	1×10^{16}
ne30, 30 levels	10 s	5×10^{15}

Table 3.2. Discretization, time step Δt, and hyperviscosity coefficients τ used to produce figure 3.17.

Instead, it offers a deterministic test that provides a basis for comparison between different model solutions. Additionally this idealized test case resembles an isolated winter-hemisphere storm and is therefore a physically relevant test for a dynamical core, unlike the advection problem and resting atmosphere test cases considered above. The initial conditions correspond to a laminar polar front jet with a small perturbation superimposed to trigger the wave's development. The wave begins to develop in earnest after day 4 and breaks into a closed midlatitude cyclone by approximately day 9.

Since an analytic solution is not available for error calculation, we instead plot several solutions using different combinations of horizontal and vertical resolutions. Figure 3.17 shows the developing baroclinic wave at day 9, as computed by Aeras, on the uniform 16 grid ($\approx 5^\circ$ horizontal resoution) with 15 vertical levels (top row) and 30 vertical levels (second row). Results from the uniform 30 grid with 15 levels and 30 levels are shown in rows three and four, respectively. The first column shows surface pressure and the second column shows temperature along the model level closest to the 850 hPa level (level 12 in the 15 level simulations and level 25 in the 30 level simulations). The day 9 time step and the variables shown were chosen to provide a direct comparison with figure 6a,b,c,f,g,h from [7]. The horizontal and vertical discretizations, time step, and hyperviscosity coefficient used to produce this figure are given in table 3.2.

The location of the wave agrees across all resolutions, but its intensity is resolution dependent – the depth of the surface low and the extent of the poleward heat transport induced by the wave are sensitive to the vertical resolution. Jablonowski and Williamson [7] report that most models have converged at horizontal resolutions $\leq 1^\circ$ with > 25 vertical levels, so the last row of the figure can be considered a converged solution. Its similarity to other models' results from [7] and DCMIP 2012 provide additional confidence in the Aeras results.

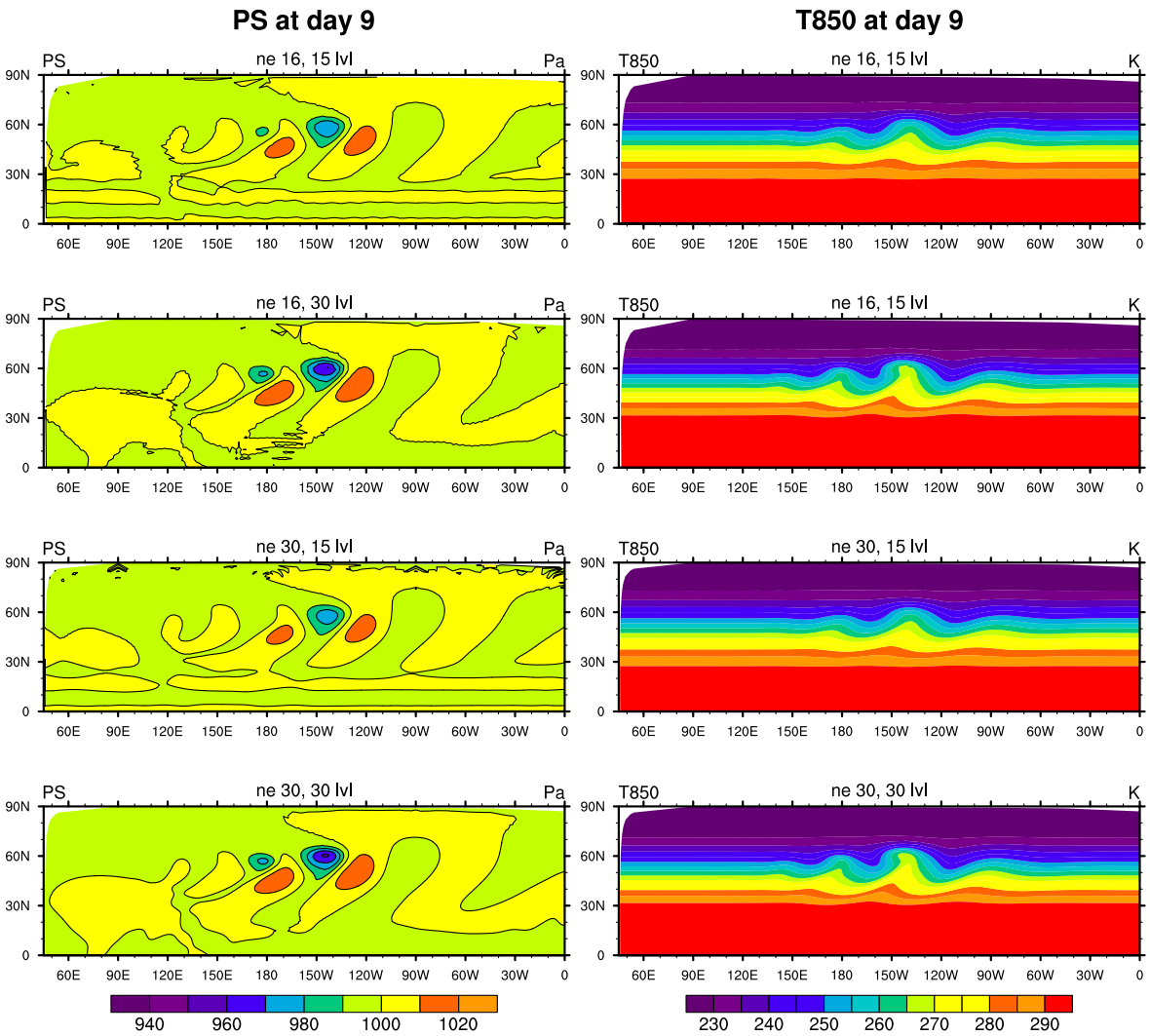

Figure 3.17. Perturbed baroclinic instability. Aeras solutions for surface pressure (1st column) and temperature at $\approx$ 850 hPa (2nd column) on the uniform 16 grid (rows 1 and 2) and the uniform 30 grid (rows 3 and 4). Computations that used 15 vertical levels are shown in rows 1 and 3; computations using 30 vertical levels are shown in rows 2 and 4.

Chapter 4

Performance Portability

Note: much of this chapter has been submitted as an article to the *J. HPC Appl.* [3].

Kokkos is a library-based programming model, which has been developed at Sandia National Laboratories to provide scientific and engineering codes with user-accessible, many-core, performance portable capabilities. For more information on Kokkos, the reader is referred to [4]. The key premises on which Kokkos is based are reviewed here.

Performance portability is the primary objective of Kokkos: it is designed to maximize the amount of user code that can be compiled for diverse devices and obtain the same (or nearly the same) performance as a variant of the code that is written specifically for that device.

There are two primary abstractions in Kokkos:

1. Kokkos polymorphic multidimentional array: `Kokkos::View`.
2. Kokkos parallel dispatch functions: `Kokkos::parallel_for`, `Kokkos::parallel_reduce` and `Kokkos::parallel_scan` .

The Kokkos API has a thin *back-end* implementation layer that maps portable user code to a number of device-specific programming models: CUDA [10], OpenMP [12], and Pthreads [9]. Kokkos separates the *Memory Space* (where data resides) from the *Execution Space* (where functions execute, such as NVIDIA GPUs, Intel Phi etc.). The integration of these abstractions enables user code to satisfy multiple architecture-specific memory access pattern performance constraints, all without requiring modifications to the application source code.

4.1 Kokkos Multi-Dimentional Array

The data access pattern of a data parallel computational kernel can have a significant impact on its performance: different architectures require different memory access patterns for optimal performance. For example, computations on an NVIDIA GPU must use a coalesced global memory access pattern, while Intel Phi will give you the best performance if the memory access is continuous. The main advantage of Kokkos is that the Kokkos multi-dimensional array has a polymorphic data layout that can be changed to have optimal access pattern on a specific *Execution Space*.

Here is an example of `Kokkos::View` allocation:

```
View<double*[6], Device> a("A", N);
```

In this example, the `View` construction allocates a two-dimensional `Nx6` array in `Device` memory space with default memory layout for this memory space. The first dimension is supplied at run-time. The label `"A"` is used for error messages. One can change the default layout as well as the execution space through additional template parameters. The parentheses operator implements the layout map:

```
a(i,j) = value;
```

A `Kokkos::View` also handles its own memory management via reference counting so that the view automatically deallocates itself when all of the variables that reference it go out of scope. This makes memory management much simpler across multiple architectures.

4.2 Kokkos Parallel Pattern

In data parallel computations, multi-dimensional arrays are partitioned among the threads of a many-core device, and each thread applies one or more computational kernels to its designated subset of these arrays. Kokkos currently implements data parallel execution with `parallel_for`, `parallel_reduce` and `parallel_scan` operations.

A `parallel_for` is trivially parallel in that the computational kernel's work is fully disjoint. In a `parallel_reduce`, each application of the computational kernel generates data that must be reduced among all work items. A `parallel_scan` is for taking a view and creating a running sum of values to replace the values of the view. Expressing an application in terms of these patterns allows the underlying implementation or compiler to make reasonable choices about valid transformations.

A data parallel computational kernel is currently implemented as a functor or C++ lambda. A functor is a C++ class that encapsulates one or more callback functions, shared parameters, and references to data upon which the callback function operates. The C++11 standard lambda feature significantly improves the syntax and usability of the functor pattern, but is not used in the current work since support for lambdas is not available on some platforms we target.

The Kokkos parallel pattern syntax includes the *Execution Policy* and the user function as input data:

```
parallel_pattern(Policy<Space>,
                 UserFunction)
```

An *Execution Policy*, together with an *Execution Pattern*, determines how a function is executed. Some policies can be nested in others. The most simple form of an execution policy is a *Range Policy*. This is used to execute an operation once for each element in a range. *Team Policies* are

used to implement hierarchical parallelism. For this purpose, Kokkos groups threads into teams, which are collections of one or more parallel threads of execution.

Users may nest parallel operations. Teams may perform one parallel operation (`for`, `reduce`, or `scan`), and threads within each team may perform another, possibly different parallel operation. Different teams may do entirely different things. For example, all the threads in one team may execute a `parallel_for`, and all the threads in a different team may execute a `parallel_scan`. Different threads within a team may also do different things.

An example of the Kokkos `parallel_for` pattern is shown in section 4.3 of this report.

Local parallel reductions are supported in Kokkos through atomic reduction operations, e.g., atomic addition. An atomic operation serializes concurrent updates to a datum but does not guarantee the ordering of these updates among threads. Thus, a non-associative local reduction operation (e.g., floating point addition) is likely to yield nondeterministic results for local parallel reductions. Kokkos wraps a collection of commonly available and compiler dependent atomic update functions under a portable interface. An example of a Kokkos atomic implementation looks like the following:

```
KOKKOS_INLINE_FUNCTION
void operator()(const int i) const
{
  for (int j=0; j<numNodes_; ++j)
    for (int k=0; k<numFields_; ++k)
      Kokkos::atomic_fetch_add(&f_[ElemID_(i,j,k)], R_(i,j,k));
}
```

The composition of parallel work dispatch and polymorphic array layout capabilities enables performance portable implementations of parallel algorithms. Atomic operations support thread-safe implementations of algorithms with local parallel reductions, which could be performant given an adequate ratio of computation to atomic operations and a low frequency of collisions.

4.3 Albany-to-Kokkos Refactoring

The procedure for porting C++ code to Kokkos is as follows:

1. Replace array allocation with Kokkos multi-dimensional arrays (`Kokkos::View`);
2. Replace numeric kernel functions with functors and run in parallel on the `Host`;
3. Enable dispatching (offloading the model) for GPU execution;
4. Optimize algorithms for threading; and
5. Optimize kernels to work efficiently on different architectures.

We describe each step in detail below.

4.3.1 Replacing data with Kokkos::View

The Phalanx package manages dependencies between *Albany evaluators*. It also manages memory allocation and memory access. Phalanx supports arbitrary user defined data types and evaluation types. Therefore, in order to replace our array data allocations with `Kokkos::Views`, we needed to first refactor the Phalanx package to use Kokkos.

In order to support backward compatibility of the Phalanx package and to reduce the effort in moving Albany and other Phalanx-based codes to the new Kokkos version of Phalanx, we relied on using the unified memory support (UVM) for the CUDA execution space, which became available in CUDA 6.0. UVM abstracts the memory management away from the programmer: with unified memory, programmers can access any resource or address within the legal address space, regardless of which pool the address actually resides in, and operate on its contents without first explicitly copying the memory over.

Though the CUDA 6.0 unified memory system does not resolve the technical limitations that require memory copies, it offers the ability to have CUDA do memory management, which simplifies CUDA programming by removing the need for programmers to do it themselves. Although UVM implicit memory copies have a performance penalty, it enabled us to move the Albany code to multi-core architectures with minimum effort. The implicit UVM data management in Albany will be replaced with manual data management at a later point in time.

After refactoring the Phalanx package by wrapping `Kokkos::Views` in Phalanx PHX::MDFields, which are used in Albany as a default array type, we focused on refactoring temporary data types used in Albany. This needed to be done to provide optimized memory access inside of the Kokkos kernels. The two temporary array types that were used in Albany before the refactoring process are: `Intrepid::FieldContainer` and `std` containers (e.g. `std::vector`). `Intrepid::FieldContainer` is a container from the Trilinos package Intrepid [2]. Intrepid is a library of interoperable tools for compatible discretizations of PDEs. It is used in Albany for computing the basis functions and their derivatives, as well as obtaining quadrature points and weights for evaluating integrals involving the basis functions. In the refactor, `Intrepid::FieldContainer` was replaced with `Intrepid2::FieldContainer_Kokkos`, which provides functionality close to `Kokkos::View`. `Std` containers were replaced with analogous containers from the Kokkos package (e.g., `std::vector` has been replaced with `Kokkos::vector`).

4.3.2 Replacing Albany Evaluators with Kokkos Functors

After replacing array allocations with `Kokkos::Views` in Albany, the next step was to create Kokkos functors for each *Albany evaluator* (shown in Figure 3.4) used in the code. An example of a Kokkos functor implementation in Albany is presented in Figure 4.1. *Albany Evaluator to*

Initial code:

```
template<typename EvalT>
void CoordGrad<EvalT>::evaluateFields()
{

// Outer loop over a Workset ofElements
for(int cell = 0; cell < NumCells; cell++) {
 for(int qp = 0; qp < numQPs; qp++) {
    for(int row = 0; row < numDims; row++){
      for(int col = 0; col < numDims; col++){
        for(int nd = 0; nd < numNodes; nd++){
          coordGrad[cell][qp][row][col] +=
             coordVec[cell][nd][row]
             * basisGrads[nd][qp][col];
        }
      }
    }
  }
} // cell loop
}
```

Kokkos Functor implementation:

```
template<typename EvalT>
void CoordGrad<EvalT>::evaluateFields()
{
  // Outer loop over a Workset of Elements
  Kokkos::parallel_for (NumCells, *this);
}

**********************************************
template<typename EvalT>
KOKKOS_INLINE_FUNCTION
void CoordGrad<EvalT>:: operator ()
  (const int cell) const
{
  for(int qp = 0; qp < numQPs; qp++) {
    for(int row = 0; row < numDims; row++){
      for(int col = 0; col < numDims; col++){
        for(int nd = 0; nd < numNodes; nd++){
          coordGrad(cell, qp, row, col) +=
             coordVec(cell, nd, row)
             * basisGrads(nd, qp, col);
        }
      }
    }
  }
}
```

Figure 4.1. Example illustrating Albany function to Kokkos functor refactoring. This includes: 1) Replacing the outer loop with a `parallel_for`, and 2) moving the inner kernel to an `operator()` functor.

Kokkos Functor refactoring includes:

1. Replacing the outer loop with a call to `parallel_for`,
2. Moving the inner kernel to the C++ method `operator()`,
3. Making the class a functor.

Note that when C++11 lambda functionality is sufficiently supported on all architectures targeted by Albany, this strategy can be superseded by a syntactically simpler lambda strategy.

The first two steps of the finite element assembly in Albany are (1) gathering data to local data structures and (2) performing element integrations, e.g., local stiffness matrices and body force terms. These are inherently thread safe (there are no thread conflicts), so we simply replace nested loops with `Kokkos::parallel_for` as presented in Figure 4.1.

The last step is assembly of local data into the global matrix or vector. This operatior is not thread safe, since a single equation gets contributions from multiple elements. In order to avoid thread collision in the assembly step, Kokkos atomic operations have been used: when a thread performs an atomic operation, the other threads see it as happening instantaneously and avoid accessing the same data.

Albany divides the total number of elements into a *workset* of elements, and then executes each workset in a cycle. For our Kokkos kernels we use a simple *Parallel Range* execution policy over the number of elements per *workset*. This number can be changed at run-time so we can easily adjust it for different architectures. We could implement more complicated execution policies to get better use of accelerators and co-processors, but we keep it simple in Albany in order to enhance code readability. Code readability is very important for Albany due to the fact its evaluators are used by different projects and modified by multiple collaborators.

4.3.3 Enable GPU Execution

While performing the first two steps of our migration strategy (see the beginning of this section) is usually enough to enable threaded execution on CPUs or Xeon Phi, it is often necessary to add explicit management of the different memory spaces for GPUs (i.e., CPU-GPU communications). In our implementation, data flow is implicitly managed by CUDA 6.0 UVM, which significantly simplifies the refactoring process.

The idea behind UVM is to use page fault memory in the virtual memory systems to detect when a piece of memory is being accessed on the GPU and move the pages to the device. The page is then moved back to the CPU when the CPU accesses it. With UVM, explicit data management ("cudaMemcpy" function call) should become optimizations rather than requirements for data movement between the CPU and GPU in the future.

Though the current implementation of UVM in CUDA introduces significant overheads (according to our experiments, UVM is up to $10\times$ slower than explicit data management), the next-generation NVIDIA GPU architecture should introduce a number of hardware improvements to increase performance and flexibility of the unified virtual memory.

At the current stage of the Kokkos project, we are more focused on creating and optimizing Kokkos kernels in Albany. The Kokkos team plans to address improvement of memory management in future work.

4.3.4 Code Optimizations

Once the code is running thread-parallel on all devices supported by Kokkos, it may be necessary to optimize the kernels in order to get better performance. The major optimization we implemented was reducing communication overhead by moving more code to the *Execution Space*.

Our Kokkos kernels in Albany are based on the Kokkos *Range Policy*. In order to improve the use of co-processors and accelerators, we examined a more complicated *Team Policy*, but we believe that the performance gain (about 10% for tested examples) is not worth the complexity of the resulting code and decided instead to use a simple *Range Policy*. As mentioned above, code readability is a critical feature for Albany, as it facilitates collaboration among diverse teams that span a variety of projects.

4.4 Evaluation Results

4.4.1 The Evaluation Environment

We evaluate the performance of the Kokkos implementations of the Aeras module in Albany on the Shannon testbed cluster, located at Sandia National Laboratories, and Titan, a Cray supercomputer established at Oak Ridge National Laboratory. Specification details for each of these evaluation environments can be found in Table 4.1.

Name	Titan	Shannon
Nodes	18688	32
CPU	2x AMD Opterons	2? Intel E5-2670 HT-off
Co-Processor	2? K20X ECC on	2? K80 ECC on
Memory/node	32 GB	128 GB
Interconnect	Gemini	QDR IB
OS	Cray Linux Environment	RedHat 6.2
Compiler	gcc 4.8.2	gcc 4.7.2
MPI	cray-mpich/7.2.4	openmpi/1.8.4
Nvcc compiler version	7.0.18	7.0.2

Table 4.1. Evaluation environments

4.4.2 Aeras Performance Results

Shallow Water Model

We evaluate our Kokkos implementation of Aeras on a common test case for the shallow water equations (2.13)-(2.14), which involves zonal flow over an isolated mountain, referred to herein as "Test Case 5", or simply "TC5" [20]. The test case consists of a zonal flow impinging on a conical mountain, and is described in more detail in section 3.5.1.

Name	Degree resolution	# elements
uniform_30	1.0°	5400
uniform_60	0.5°	21,600
uniform_120	0.25°	86,400
uniform_180	0.167°	345,000

Table 4.2. Cubed-sphere mesh resolutions considered for Aeras performance results

For the performance and scalability studies summarized here, five uniform cubed-sphere meshes (Figure 3.1) of varying resolutions were considered. These meshes are summarized in Table 4.2.

Bicubic shell quadrilateral spectral elements, that is, 2D elements with 16 nodes, were employed on each mesh. For energy dissipation and stabilization, a constant-coefficient hyperviscosity was employed, with hyperviscosity coefficient $\tau = 1.0 \times 10^{15}$. The SEM discretized system is integrated forward in time using an explicit 4th order Runge-Kutta (RK4) time-stepping scheme. A time step was selected to satisfy the Courant-Friedrichs-Lewy (CFL) condition, and stability was verified for a 15 day run (Figure 3.9). For the evaluation of the Kokkos implementation of Aeras, a short-time simulation was considered (up to time $T = 1$ day (86400 seconds)). It is emphasized that the same finite element code base was used for all the runs presented below, but each with a different configuration option for the Kokkos `ExecutionSpace` template parameter.

Figure 4.2 (a) and (b) provide performance results in the form of the total time and total compute time (total time minus gather/scatter), respectively, for the Aeras Shallow Water TC5 run on a 0.5° mesh on the Shannon cluster at Sandia. Here and below (Figure 4.2), the curve labeled *Serial* corresponds to the original implementation of Albany (no Kokkos kernels) with 1 MPI thread per node; the curves labeled *OpenMP* and *CUDA* represent performance results for the Kokkos implementation of Albany with Kokkos *Execution Space = OpenMP*, *OMP_NUM_THREADS=16* and Kokkos *Execution Space = CUDAUVM*, respectively.

The total time includes CPU-GPU communications for the CUDA implementation. As mentioned before, CPU-GPU communications are managed by CUDA UVM in Albany. In order to compare CUDA UVM data transfer overheads with the overheads for explicitly managed CPU-GPU dataflow, we run Kokkos unit test examples on two different *Execution Spaces*: *CUDA* and *CUDAUVM*. Performance comparison results showed that communication overheads with the *CUDAUVM Execution Space* are about 10× larger than with the *CUDA Execution Space*. Note that the performance of *CUDAUVM* is going to be significantly improved in future versions of NVIDIA compilers on future CPU-GPU hybrid architectures. CPU-GPU communications are included in the execution time of the "Gather" and "Scatter" evaluators of Albany. We present *compute time* results in order to show the performance we get if we eliminate communication overheads from the total Finite Element Assembly time in Albany.

The total Kokkos OpenMP implementation time (16 threads) is up to 8 times faster than for the single-CPU implementation. The CUDA implementation is slower than both the OpenMP and Serial implementations due to the large CPU-GPU communication overheads (see Figure 4.2 (a)). While the OpenMP and the Serial implementation results are almost flat, CUDA performance improves with higher number of elements per workset. This can be explained by the fact that there is not enough work for the GPU with smaller workset sizes and we are essentially measuring the kernel's call latency and CPU-GPU communication.

Comparing performance results for the total time (Figure 4.2 (a)) with results for the compute time (Figure 4.2 (b)), we can see that compute time for the CUDA implementation is now faster than both Serial and OpenMP implementations for the number of elements per workset larger than 1000. Compute time for the OpenMP implementation is about 15 times faster that the time for the Serial implementation. While most of the Albany evaluators are "thread-safe", the "Scatter" eval-

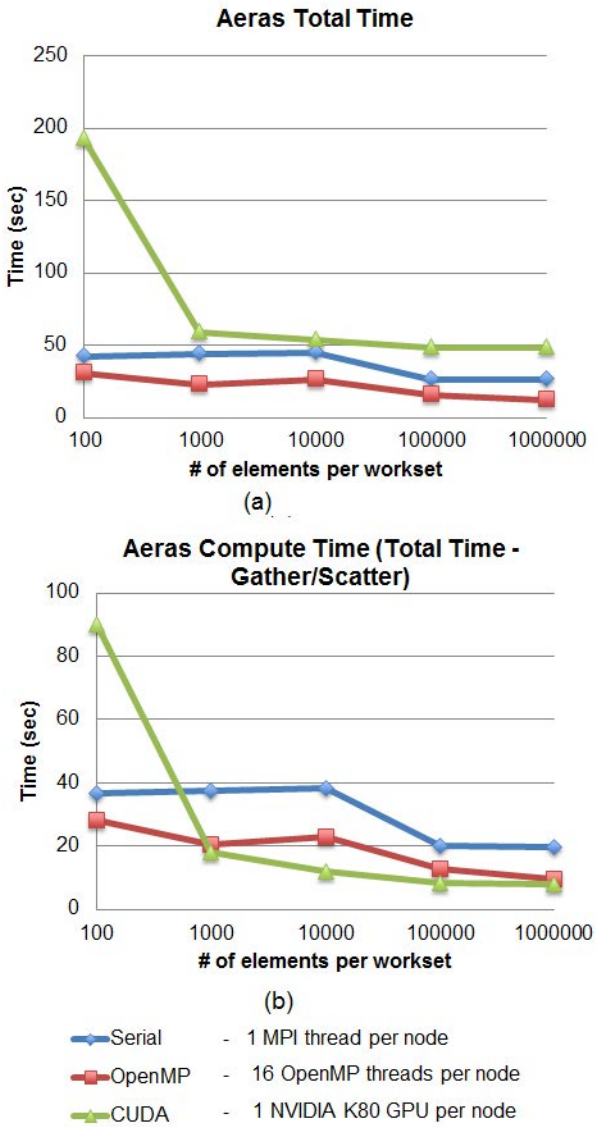

Figure 4.2. Strong scalability results for Aeras Shallow Water TC5 on Shannon for the uniform_60 (0.5°) mesh: (a) total time as a function of the number of elements per workset; (b) time without gather/scatter as a function of the number of elements per workset

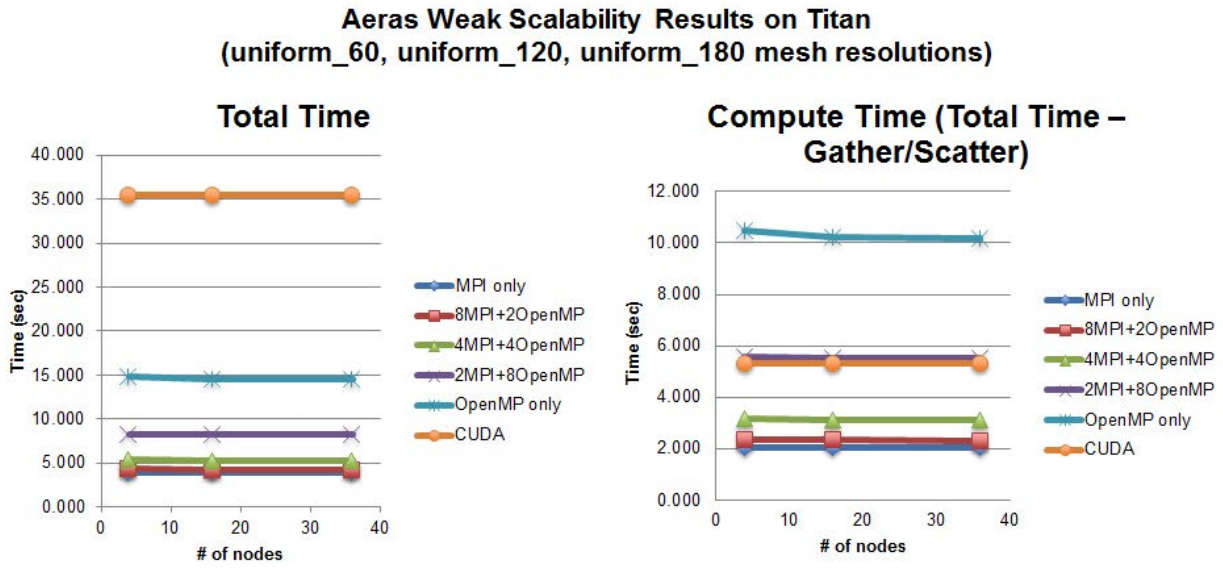

Figure 4.3. Weak scalability results for Aeras Shallow Water TC5 on Titan (about 5600 elements per node): total time (left); compute time (right)

uator represents local parallel reduction and its Kokkos implementation is based on using Kokkos atomic operations that serialize concurrent updates to a datum. Therefore, eliminating the "Scatter" evaluator from the total time significantly improves total OpenMP performance.

In Figure 4.3, we compare execution time as a function of compute nodes, holding the total number of elements constant in order to study weak scalability. Results are obtained on Titan Cray supercomputer, established at Oak Ridge National Laboratory. Experiments are run on different architectures (NVIDIA K20 GPU and Intel Xeon) along with a single CPU run. Excellent weak scalability is observed for all runs.

Next, we compare performance of the original implementation run with the configuration of 16 MPI threads per node (MPI-only), the Kokkos OpenMP implementation with different number of OpenMP threads per MPI thread and Kokkos CUDA implementation on Titan. As for the Shannon results, we compare the total and compute times.

It is shown that the MPI-only version of the code gives the best performance results on Titan. Although the Kokkos OpenMP multi-threaded implementation is expected to run faster than the MPI-only version, the overhead of creating, destroying and synchronizing threads may be very high, especially for the data-intensive problem as the Aeras problem considered here. CUDA implementation is about 6 times slower than the MPI-only implementation for the total time, and about 2.7 times slower for the compute time. These results are very different form the ones on Shannon due to the fact that Titan has older version of the GPU (NVIDIA k20) and PCI express bus with higher latency.

3D Hydrostatic Model

In this section, the computational performance of the Kokkos implementation for the Aeras 3D Hydrostatic equations is characterized by using OpenMP and CUDA. We focus our performance analysis on the baroclinic instability test case. Table 4.3 shows the three cubed-sphere mesh resolutions used in these simulations along with other notable parameters.

Name	Degree resolution	# elements	Fixed dt	Hyperviscosity Tau
uniform_30	1.0°	5400	30	5.00e15
uniform_60	0.5°	21,600	10	1.09e14
uniform_120	0.25°	86,400	5	1.18e13

Table 4.3. Cubed-sphere mesh resolutions considered for Aeras 3D Hydrostatic performance results

Bicubic shell quadrilateral spectral elements were again used for this test case and 10 levels were used in the vertical direction. Each simulation is advanced in time using 100 iterations of an explicit 4 stage, 3rd order Runge-Kutta time-stepping scheme. The wall-clock time of each simulation is computed by substracting the setup time from the total wall-clock time of the simulation.

Figures 4.4 and 4.5 show the wall-clock time and OpenMP and GPU speedup over MPI for the uniform_30 mesh for different workset sizes on the Shannon cluster. The Kokkos OpenMP implementation time (8 threads) is up to 4 times faster than the single-CPU implementation. The speedup in the Kokkos CUDA implementation is much smaller because of the small workset size. Larger workset sizes could not be used because of memory limitations on the GPUs.

Figure 4.6 shows the wall-clock time for the three mesh resolutions shown in Table 4.3 on the Titan cluster. In this case, weak scalability is analyzed by using 32 MPI ranks, 32 MPI ranks + 8 OpenMP threads and 32 MPI ranks + 32 GPUs for the uniform_30 mesh, 128 MPI ranks, 128 MPI ranks + 8 OpenMP threads and 128 MPI ranks + 128 GPUs for the uniform_60 mesh and 512 MPI ranks, 512 MPI ranks + 8 OpenMP threads and 512 MPI ranks + 512 GPUs for the uniform_120 mesh. The results show near perfect weak scaling. Figure 4.7 shows OpenMP and GPU speedup over MPI. The speedup in the Kokkos OpenMP implementation (8 threads) is approximately 3 which is a bit less than the 4 times speedup on Shannon. The Kokkos CUDA implementation is approximately 2 times slower than the single CPU implementation. This is due to the small workset sizes on each GPU (approximately 168 elements per GPU). Larger workset sizes would be more efficient but could not be achieved because of memory limitations on the GPU.

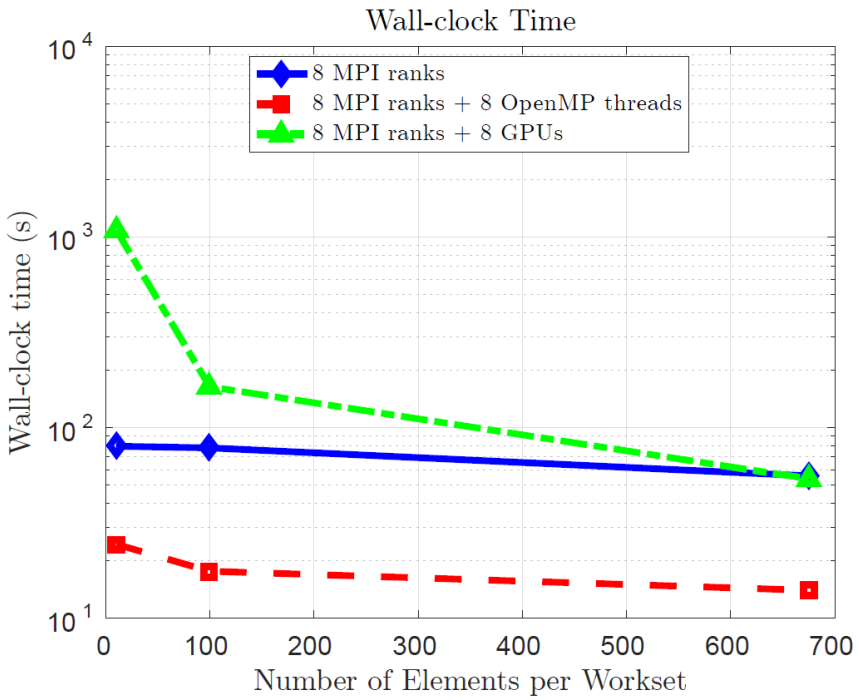

Figure 4.4. Wall-clock time as a function of the number of elements per workset for Aeras 3D Hydrostatic baroclinic instability on Shannon for the uniform_30 mesh

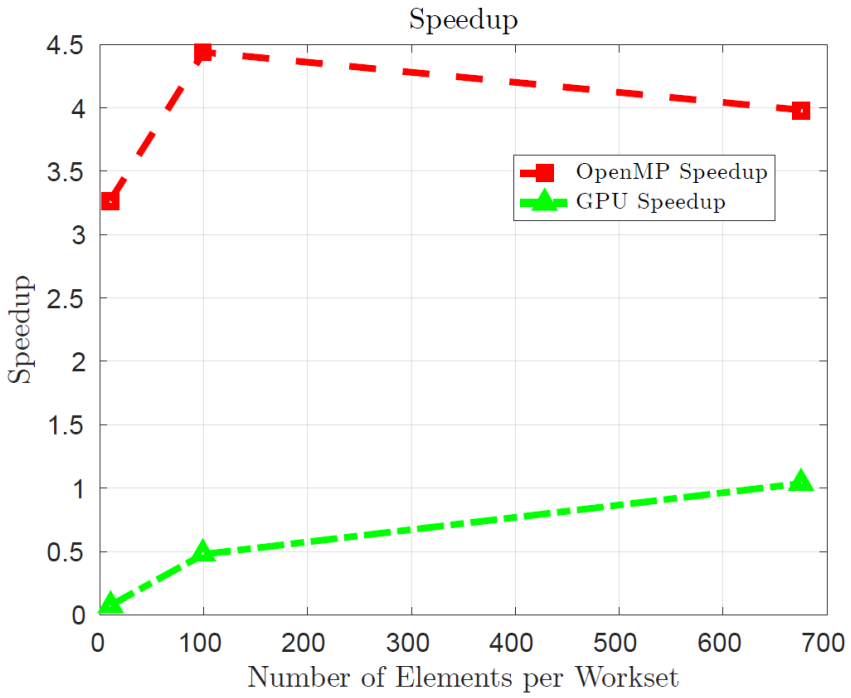

Figure 4.5. OpenMP and Nvidia K80 GPU speedup over MPI as a function of the number of elements per workset for Aeras 3D Hydrostatic baroclinic instability on Shannon for the uniform_30 mesh

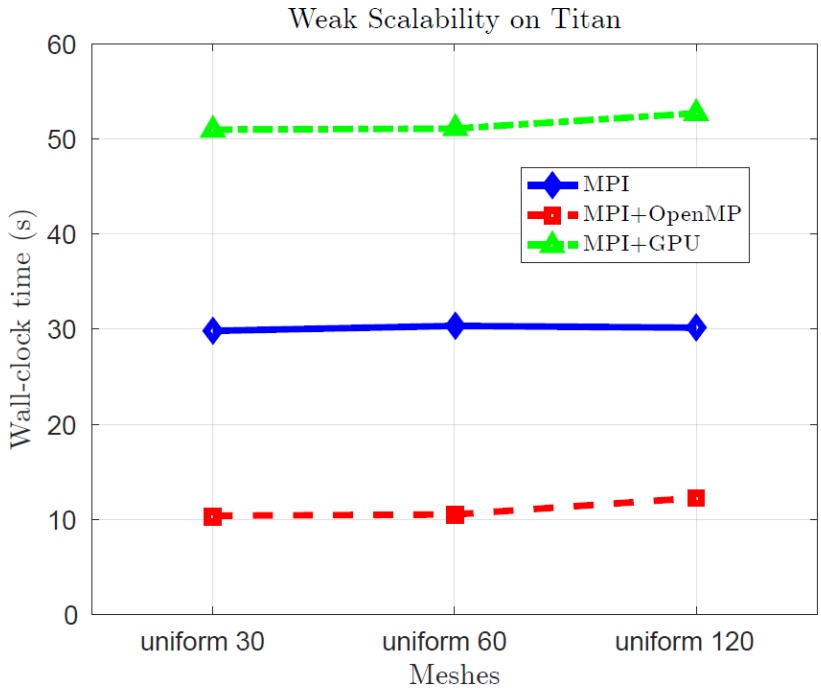

Figure 4.6. Weak Scalability results for the Aeras 3D Hydrostatic baroclinic instability test case on Titan

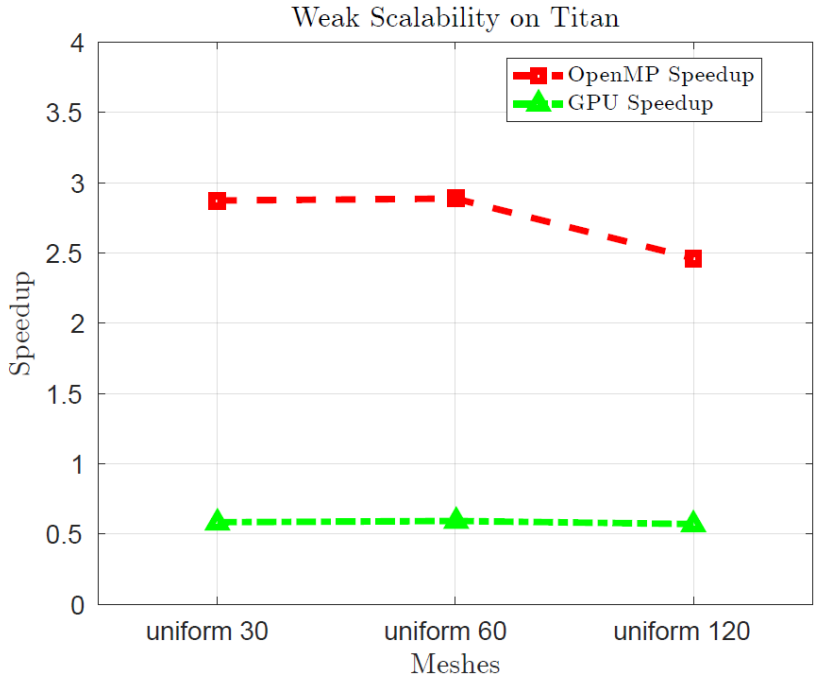

Figure 4.7. OpenMP and Nvidia K20X GPU speedup over MPI for the Aeras 3D Hydrostatic baroclinic instability test case on Titan

Chapter 5

Uncertainty Quantification

The analysis of real-world systems using computational analysis tools has increased as the tools have matured and computational resources have increased. These computational analysis tools often make simplifying assumptions that affect their ability to accurately predict the real-world phenomena that they are trying to simulate. These assumptions are often made to overcome a lack of knowledge of the true physical processes that are part of the system (epistemic uncertainty) or to account for the inherent variability in the environment (aleatoric uncertainty). Uncertainty quantification (UQ) methods seek to assess the effect of these assumptions on the predicted quantities of interest. The work presented here is applicable to aleatoric uncertainties, including variations in initial conditions and parameters that affect how the computational model behaves, but not epistemic or model-form uncertainty.

Global atmosphere models have many sources of uncertainty. Typical global atmosphere models directly solve for the atmospheric circulation but use empirical sub-grid scale models to characterize other physical processes such as clouds and precipitation or solar radiation. A standard approach to characterizing the effects of these uncertainties is to run an ensemble of simulations. In the atmospheric modeling community, there are three main categories of multiple ensembles. Multi-model ensembles involve running several atmospheric models developed by independent groups in an attempt to address the model-form uncertainty introduced by assumptions about how the various physical processes in the atmosphere interact with each other. Perturbed physics ensembles seek to address the parametric uncertainty of a model by varying the parameters that influence the interactions within the model. Initial condition ensembles address the uncertainty associated with the initial state of the atmosphere and/or ocean that are used to start the simulation. The embedded UQ methods discussed here are appropriate for use in perturbed physics ensembles or initial condition ensembles.

Uncertainty quantification methods can be broadly categorized into intrusive (or embedded) and non-intrusive methods. Non-intrusive methods, such as Monte-Carlo sampling, treat the simulation code as a black box, and repeatedly run the analysis code while varying the input parameters. These methods are straight forward to use as they do not require any modifications to the analysis code, but may not be very computationally efficient. Embedded or intrusive UQ methods, such as stochastic Galerkin methods, require modifications to the analysis code. The approach we take for Aeras can be thought of as a hybrid of intrusive and non-intrusive. We use non-intrusive sampling based methods, but we modify the analysis code to more efficiently evaluate these samples.

5.1 Concurrent Ensemble Sample Propagation

Albany, and by extension Aeras, can employ both intrusive and non-intrusive UQ methods. Non-intrusive UQ methods are enabled by Dakota, which provides sampling based methods such as Monte Carlo and non-intrusive polynomial chaos expansions. Intrusive (or embedded) UQ methods are enabled in Albany through the use of the Trilinos package Stokhos. Stokhos uses the concept of operator overloading to define embedded UQ datatypes that minimize the required modifications to an analysis code. Stokhos provides tools to enable stochastic Galerkin methods and concurrent ensemble sample propagation. Aeras makes use of the concurrent ensemble sample propagation capability of Stokhos to improve the computational efficiency of sample-based UQ methods.

In traditional sample-based UQ methods, each sample is evaluated independently of each other. This can be thought of as a loop over the number of samples, where the analysis code is called inside the loop. Running the analysis in this fashion does not require any modifications to the code. However, there is often a substantial amount of duplicated effort with this approach, such as reading in the mesh or computing the basis functions for finite element calculations.

In concurrent ensemble sample propagation, a set of the requested samples are evaluated concurrently by the analysis code. For each ensemble of samples, routines that do not depend on the values of the samples are called only once, such as reading in the mesh for cases without geometric variations. Concurrent ensemble sample propagation can be thought of as a reordering of the loops so that the loop over the samples occurs at the scalar level of the analysis code. This means that at the smallest scales of the code each operation that would normally operate on scalar values is instead carried out on an ensemble of scalar values.

The use of concurrent ensemble sample propagation offers many possibilities for improving the computational efficiency of sampling based UQ. Operating on arrays of scalar values that are stored contiguously in memory offer improved memory access patterns and an increase in the ability to vectorize the operations. For parallel calculations using MPI, message size is increased and fewer messages are sent, reducing the overall amount of communication overhead. In general, increasing the amount of work done at the smallest scales provides increased opportunities for fine-grained parallelism and efficiency.

The use of Stokhos for embedded uncertainty quantification has been a capability of Albany for steady state problems. In order to apply these capabilities to Aeras, the use of Stokhos in Albany was extended to transient problems. During this effort, the default storage type for the Stokhos multipoint type was changed to the more efficient static vector storage. These new capabilities were then applied to several Aeras test problems.

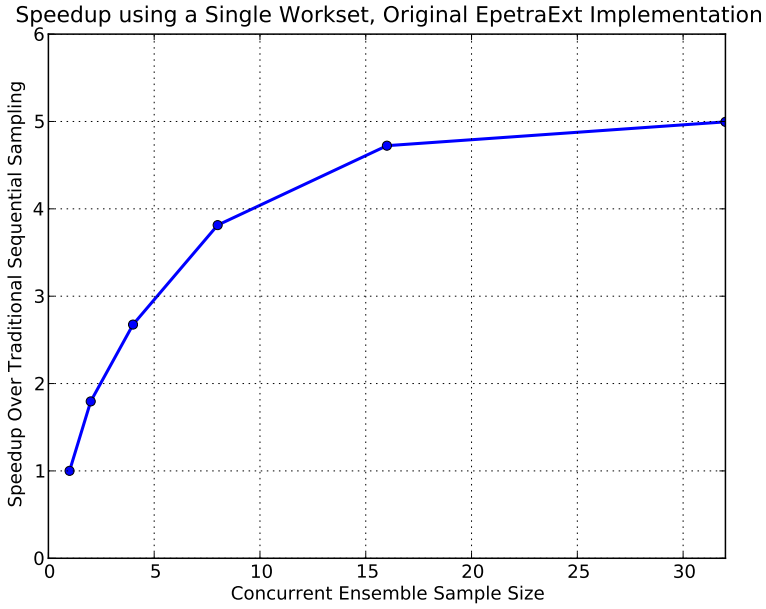

Figure 5.1. Observed speedups for original EpetraExt concurrent sample implementation when using a single workset.

5.2 Concurrent Ensemble Samples Speedup Results

The results presented here are for shallow water test case 5. Similar results were observed for a 2D hydrostatic test case, but are not presented here. Shallow water test case 5 simulates flow over an isolated mountain. For the purposes of these embedded UQ runs, the height of the mountain is treated as a random variable. The test case is run for a total of 3000 time steps using a time step of 9 seconds.

Figure 5.1 shows the observed speedups when using concurrent ensemble sample propagation compared to traditional sequential sampling. For these results, a total of 32 samples are requested. Results are presented for ensemble sample sizes of 1, 2, 4, 8, 16, and 32. When an ensemble size of 32 is used, a single evaluation of Aeras is performed. When an ensemble size of 16 is used, two evaluations of Aeras are performed. When an ensemble size of 1 is used, 32 evaluations of Aeras are performed. This should be equivalent to traditional sequential sampling, but some small timing differences are observed in some cases.

The results in Figure 5.1 show the speedup increasing to around 5 as the ensemble sample size is increased to 32. After these results were generated, several optimizations were made to Aeras to try to improve the baseline timings. In Albany, worksets are used to decompose a problem into smaller parts in order to control the memory required for the calculations. These optimizations have the largest effect when Albany is run using a single workset. Figure 5.2 shows the observed speedups when using a single workset after these optimizations were put in place. The maximum speedup observed is now around 2.2 and occurs with an ensemble size of 8. If Aeras is run with

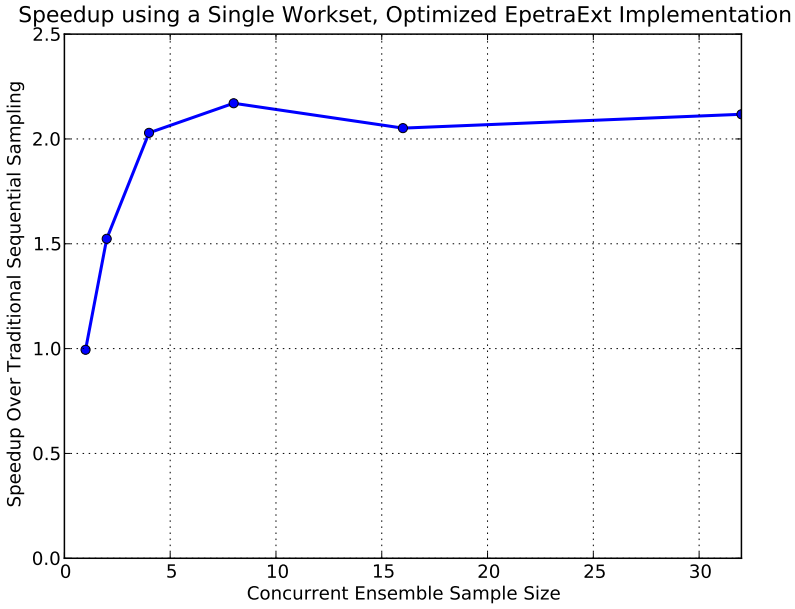

Figure 5.2. Observed speedups for optimized EpetraExt concurrent sample implementation when using a single workset.

multiple worksets, then a speedup of around 5 is observed for an ensemble size of 32, as shown in Figure 5.3.

The concurrent ensemble sample propagation implementation in Aeras discussed so far is based upon the EpetraExt model evaluator. Unfortunately, this means that other features of Aeras are not available, such as the use of spectral elements and Kokkos. These features require the use of the Thyra model evaluator. In principle, converting the concurrent ensemble implementation to use the Thyra model evaluator should be straightforward.

The Thyra model evaluator is templated on the scalar type, so switching from the standard double precision to the Stokhos multipoint type should be fairly simple. However, there were a number of packages in Trilinos that needed to be modified to get this to work. In particular the following Trilinos packages required some level of modification: Stokhos, TriKota, Rythmos, Stratimikos, and Piro. There were also a number of changes required to Albany and Aeras. Currently, these changes exist in local repositories and of this writing have not yet been pushed to the public repositories for Trilinos or Albany. A plan is being developed to push these changes once they are cleaned up and shown not to negatively impact other work.

The conversion to use concurrent ensembles with Thyra is not entirely complete. Currently, time stepping using Rythmos is disabled so the results that will be presented here are for a single residual evaluation. Getting time stepping with Rythmos to work may require substantial modifications to Stratimikos. Stratimikos is currently not templated, so the effort required to get it to work with concurrent ensembles is likely beyond the scope of this project.

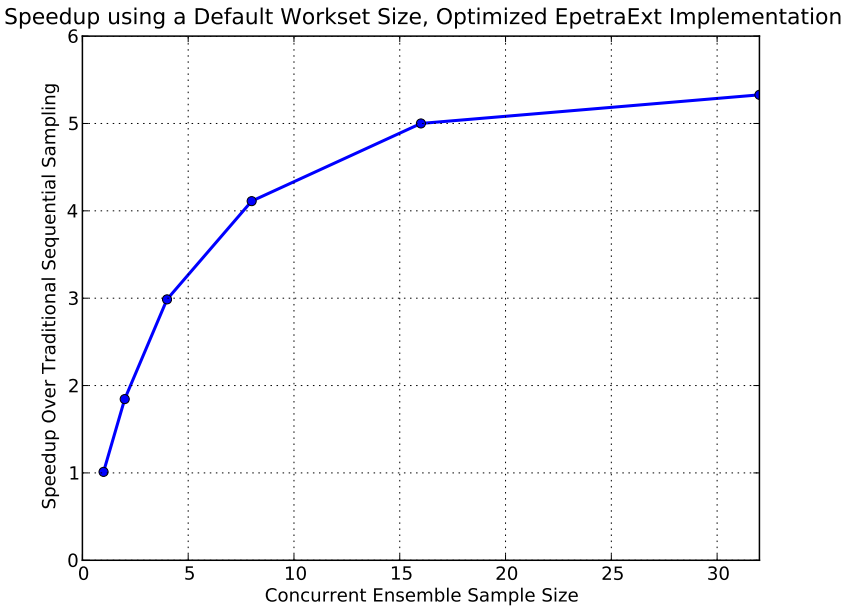

Figure 5.3. Observed speedups for optimized EpetraExt concurrent sample implementation when using the default workset size.

Figure 5.4 shows the observed speedups when using the Thyra implementation. These results show a maximum speedup of around 2.2 for an ensemble size of 8. This is comparable to the EpetraExt results shown in Figure 5.2, however the Thyra implementation exhibits lower speedups for ensemble sizes of 16 or 32. These results are for a single workset with the optimizations enabled. If the optimizations are disabled, then a speedup of around 5 is observed for an ensemble size of 32, as can be seen in Figure 5.5.

The conversion to use the Thyra model evaluator enables the use of Kokkos, and thus performance portability. Figure 5.6 shows the observed speedups when using Kokkos with the default serial node. This should be roughly equivalent to running without Kokkos, but differences in the speedups can be seen. The maximum speedup still occurs for an ensemble size of 8, but the maximum value has decreased to around 1.8.

Figure 5.6 shows the observed speedups when using Kokkos with OpenMP. This plot compares runs using different numbers of threads. There are slight differences between the runs and some observations can be made, but these trends may not extend to other cases. With 1, 2, or 4 threads the maximum speedup occurs at an ensemble size of 8. For 8 threads, ensemble sizes of 4 and 8 have comparable speedups.

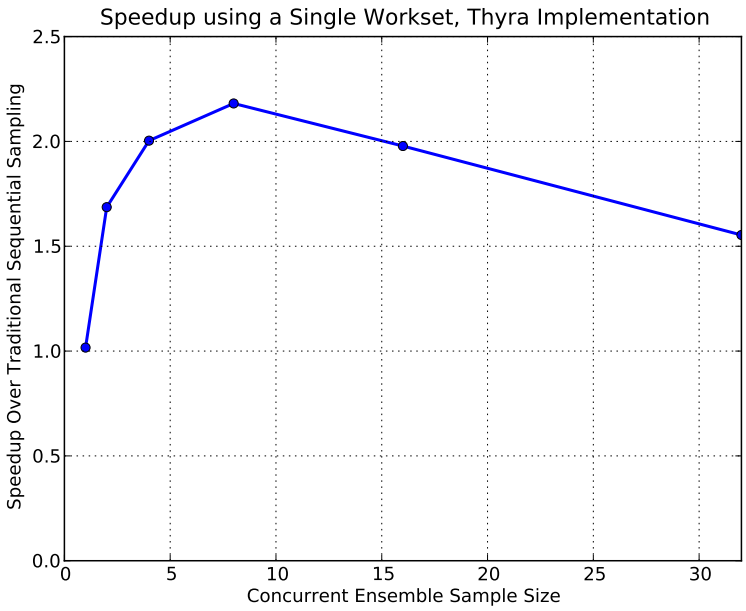

Figure 5.4. Observed speedups for Thyra concurrent sample implementation when using a single workset.

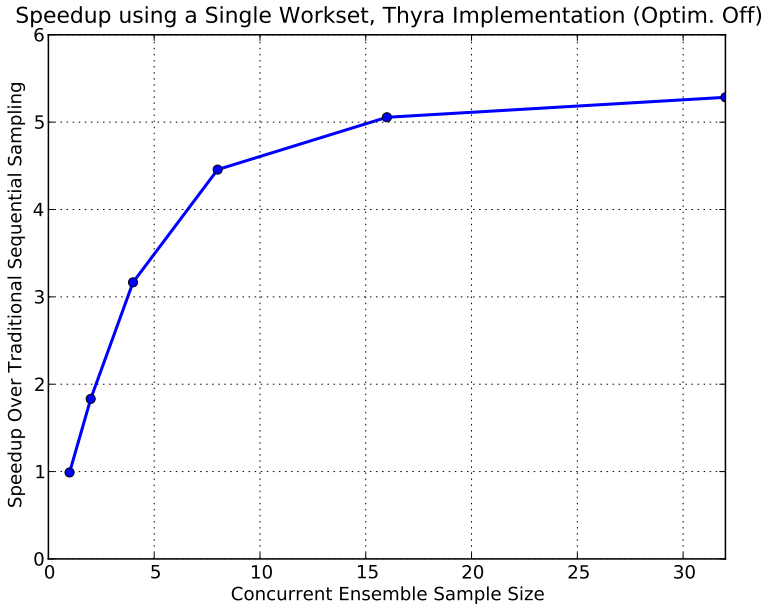

Figure 5.5. Observed speedups for Thyra concurrent sample implementation when using a single workset with optimizations disabled.

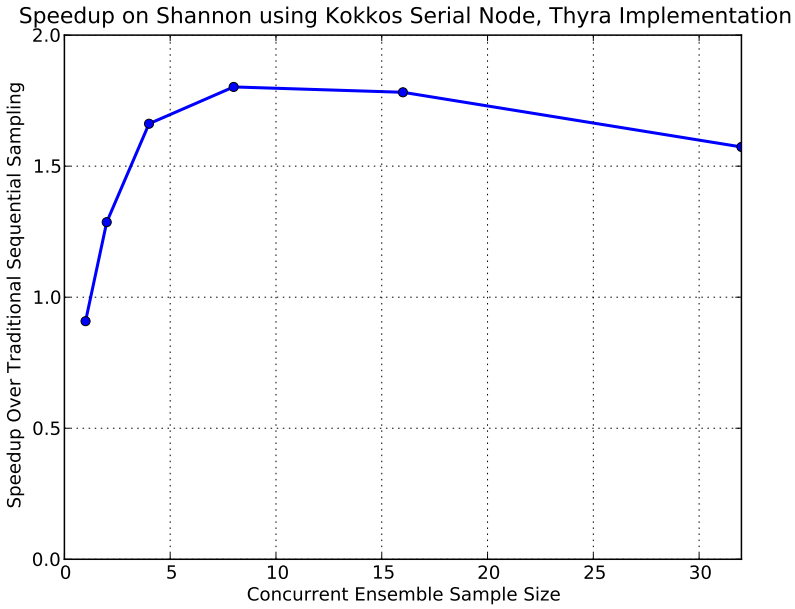

Figure 5.6. Observed speedups for Thyra concurrent sample implementation when using Kokkos serial node.

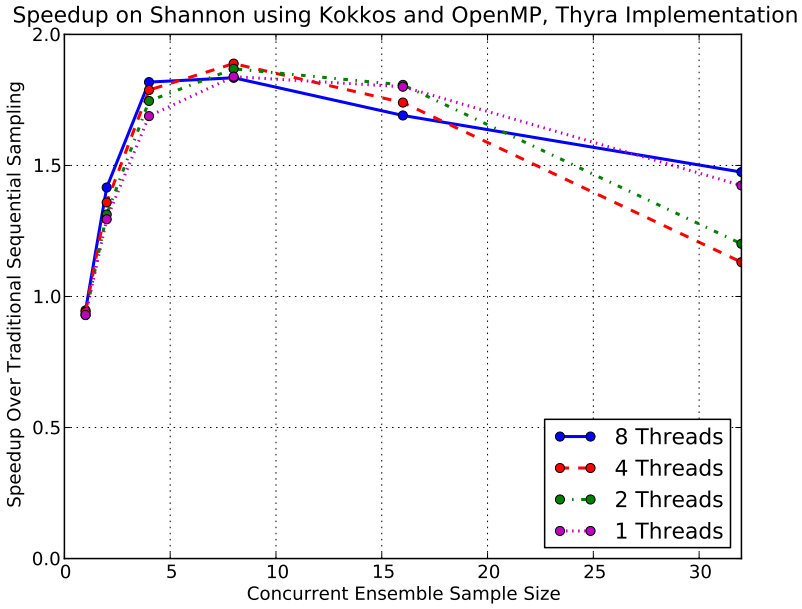

Figure 5.7. Observed speedups for Thyra concurrent sample implementation when using Kokkos with OpenMP.

Chapter 6

Future Work

In late 2015, Sandia hosted a site visit by ACME leadership, including Dorothy Koch, program manager at BER for ACME, and Dave Bader, PI for ACME. During this site visit, Aeras PI Bill Spotz, 1446, presented current results for performance portability and concurrent samples for UQ in Aeras. A few weeks later, Sandia gave a program update to all of BER management in Germantown, MD, including a similar update on progress with Aeras.

In early 2016, BER released a Funding Opportunity Announcement for the *Climate Modeling Development and Validation* program. Based in part on the success of the Aeras project, Sandia was invited to lead a multi-institution proposal for the modernization of ACME software. Led by Andy Salinger, 1442, the *ACME Software Modernization Surge* proposal was awarded for $4.75M/yr for three years. Of that total, Sandia will receive roughly $1.8M/yr to improve various aspects of the computational science foundation for ACME.

A little over half of that $1.8M/yr will be devoted to the Spectral Element Dycore Upgrade task, a direct extension of the Aeras project. This task will systematically convert the ACME CAM-SE computational kernels from Fortran to C++ so that the Kokkos programming model can be introduced. At each step, unit and regression tests will be run to ensure that performance is not sacrificed. At the end of this process, code that has been duplicated and modified for at least three different architectures will be consolidated into a single code base, will run efficiently on all the same computers, and will be significantly easier to maintain. They will also be ready for any new high performance programming model that might be introduced by the Exascale Computing Project, by seamlessly leveraging the porting efforts of the Kokkos team. Once these kernels are converted to C++, a similar process will be followed to introduce concurrent samples, which, like Kokkos, depends on C++ templates.

The Areas models will serve as templates for the Kokkos conversion, provide benchmarks for performance on advanced architectures, and former Aeras staff will provide lessons learned to ease the transition. The end result will be a production-quality atmosphere model for climate with the next-generation capabilities developed in this LDRD project.

Chapter 7

Conclusions

The Aeras project was started with the goal of developing a prototype global atmosphere model with next generation capabilities considered important by the climate modeling community. These next generation capabilities included performance portability, in which a single code base can support efficient execution on multiple diverging computer architectures; and embedded uncertainty quantification (UQ), in which the code itself can provide feedback that improves a UQ algorithm.

On the first front, performance portability, we were able to utilize the Kokkos software package and programming model to develop a 2D shallow water model, a 2D x-z hydrostatic model, and a 3D hydrostatic model that are performance portable: they all run efficiently on serial nodes, threaded nodes and GPU nodes. For the serial case, we achieved an efficiency of roughly one-half that of HOMME, the model that serves as the atmospheric dynamical core of the production climate models ACME and CESM. While a more efficient serial performance for Aeras would have been desirable, it should be noted that HOMME has been optimized for well over a decade, and the efficiency that we did achieve actually speaks quite well for Albany, the enabling technology on top of which we rapidly prototyped Aeras. Relative to this serial performance, the threaded performance consistently showed speedups at or near what one would expect. The GPU performance was a little more mixed. Newer machines such as Shannon showed better relative speedup than older machines such as Titan, which is not unexpected. CPU-GPU communications have always been a bottleneck for GPU performance. Our strategic decision to use CUDA UVM made this bottleneck worse in the short term, but will be an advantage long-term. Eliminating the gather-scatter operation (and thus the CPU-GPU communication) from the timings, we found certain configurations where MPI-CUDA was slower than MPI-only, but other configurations where MPI-CUDA was faster. Obtaining efficient GPU performance still requires some specialized tuning, and will always be dependent on large workset sizes. Nevertheless, the fact that improved performance was achieved with a single code base represents a significant saving potential in future porting efforts.

On the second front, embedded UQ, the research underwent a slight course correction when we noted the growing consensus that embedded techniques are of little help in fundamentally chaotic systems. This will limit UQ for climate to ensemble techniques, and so we pursued a new approach called concurrent samples that allows us to run multiple simulations simultaneously. In this mode, we can amortize calculations that are common to all of the simulations, reduce latency over all of the simulations by combining communication, and increase efficiency by providing processing units with more work to do. Under a large variety of different configurations, we typically saw a doubling of runtime efficiency when utilizing concurrent samples.

The Aeras project proved its relevance to the DOE climate modeling community by serving as one of several catalysts to Sandia's invitation to lead a proposal for the Climate Model Development and Validation (CMDV) program. The successful proposal resulted in a total award of $4.75M/yr for three years to modernize several aspects of the ACME software base. Sandia receives roughly $1.8M/yr of this total, and just over half of that amount is earmarked for the extension of the Aeras project: to introduce Aeras technologies into the ACME atmospheric model to provide both performance portability and concurrent samples. The awarding of this top-level project, dubbed CMDV-Software, cements Sandia's leadership role for computational science in climate. Sandia's willingness to expend LDRD funds on this project, and its subsequent success, played an important role in earning that leadership role.

Appendix A

Papers and Presentations

A.1 Papers

- W. Spotz, T. Smith, I. Demeshko and J. Fike, "Aeras: A Next Generation Global Atmosphere Model," *Procedia Computer Science: International Conference On Computational Science*, ICCS 2015 Computational Science at the Gates of Nature, v51, K. Sloawomir, et.al., editors, 2015, pp. 20972106.
- I. Demeshko, O. Guba, R. Pawlowski, M. Heroux, A. Salinger, W. Spotz and I. Tezaur, "Towards Performance-Portability of the Albany Finite Element Analysis Code Using the Trilinos Library Kokkos," submitted to *J. HPC Appl.*, 2015.
- J. Fike, W. Spotz, E. Phipps, A. Salinger and I. Tezaur, "Embedded Uncertainty Quantification in Aeras, a Next-Generation Global Atmosphere Model," *in preparation*.

A.2 Presentations

- W. Spotz, A. Salinger, S. Bova, J. Overfelt and M. Taylor, "Extending Albany to Solve PDEs on the Sphere," poster at *Workshop for the Solution of PDEs on the Sphere*, Boulder, CO, April, 2014.
- W. Spotz, "A Next-Generation Global Atmosphere Model," presented at *International Conference on Computational Science*, Reykjavic, Iceland, June, 2015.
- W. Spotz, J. Fike, I. Tezaur, A. Salinger, E. Phipps, "Uncertainty Quantification for a Next-Generation Global Atmosphere Model," poster at *Workshop on Uncertainty Quantification in Climate Modeling and Projection*, Trieste, Italy, July, 2015.
- W. Spotz, "A Next-Generation Global Atmosphere Model," presented at *Workshop for the Solution of PDEs on the Sphere*, Seoul, South Korea, October, 2015.
- W. Spotz, "UQ for the Global Atmosphere," presented at *SIAM Conference on Uncertainty Quantification*, Lausanne, Switzerland, April, 2016.

- I. Tezaur, I. Demeshko, A. Salinger and W. Spotz, "Building Next-Generation Atmosphere and Land-Ice Models Using the Kokkos Trilinos Library," presented at *Computational Methods for Water Resources*, Toronto, Canada, June, 2016.

A Note on Conferences Attended We presented results from this project at a number of conferences held in foreign countries, indeed more than we expected at the onset of the project. One reason for this is that several of the meetings are both highly appropriate for this area of research and development, typically or often held in the United States, but happened to be held in foreign countries during the timeframe of this project. For example:

PDEs on the Sphere This is a workshop aimed at the community of global atmosphere model developers, and so we considered it a required meeting. Historically, it used to be held solely at US locations, but increasing participants from Europe and Asia has led to new international hosts in recent years, such as England, Germany, Japan, and in 2015, South Korea.

SIAM Conference on Uncertainty Quantification SIAM meetings are prestigious conferences and a perfect place to report our UQ and concurrent samples work. Typically held in the US, the 2016 SIAM UQ meeting was nevertheless held in Switzerland.

Other conferences presented special opportunities:

International Conference on Computational Science, held in Iceland in 2015, is an important meeting attended by many Sandia researchers from the Center for Computing Research. The Aeras project was invited to participate in a conference workshop entitled "Numerical and Computational Developments to Advance Multi-scale Earth System Models," which we deemed highly appropriate.

Workshop on Uncertainty Quantification in Climate Modeling and Projection, held in Italy in 2015, was the first workshop we had been made aware of that focused on UQ specifically for climate modeling. This meeting ended up giving us important interactions with and insights from end users for UQ in climate.

References

[1] David Bader, William Collins, Robert Jacob, Philip Jones, Philip Rasch, Mark Taylor, Peter Thornton, and Dean Williams. Accelerated Climate Modeling For Energy (ACME) Project Strategy And Initial Implementation Plan. Technical report, Lawrence Livermore National Laboratory, 2014.

[2] P. Bochev, H. C. Edwards, R. C. Kirby, K. Peterson, and D. Ridzal. Solving PDEs with Intrepid. *Scientific Programming*, 20(2), 2012.

[3] I. Demeshko, O. Guba, R. Pawlowski, M. Heroux, A. Salinger, W. Spotz, and I. Tezaur. Towards performance portability of the Albany finite element analysis code using the Trilinos library Kokkos. *J. HPC Appll.*, submitted, 2016.

[4] H. Carter Edwards, Christian R. Trott, and Daniel Sunderland. Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. *Journal of Parallel and Distributed Computing*, 74(12):3202–3216, 2014.

[5] J. W. Hurrell and et al. The Community Earth System Model. *Bull. Am. Met. Soc.*, 2013.

[6] Mohamed Iskandarani, Dale B Haidvogel, and John P. Boyd. A staggered spectral element model with application to the oceanic shallow water equations. *International Journal for Numerical Methods in Fluids*, 20:393–414, 1995.

[7] C. Jablonowski and D. L. Williamson. A baroclinic instability test case for atmospheric dynamical cores. *Q. J. R. Meteorol. Soc.*, 132, 2006. doi:10.1256/qj.06.n.

[8] I. Kinnmark and W. Gray. One step integration methods of third-fourth order accuracy with hyperbolic stability limits. *Mathematics and Computers in Simulation*, 26:181–188, 1984.

[9] B. Lewis and D. J. Berg. *Multithreaded Programming with Pthreads*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1998.

[10] NVIDIA CUDA Programming Guide version 3.0. Technical report, NVIDIA Corporation, 2010.

[11] C. Ober, R. Bartlett, T. Coffey, and R. Pawlowski. Rythmos: Solution and Analysis Package for Differential-Algebraic and Ordinary-Differential Equations. Technical report, Sandia National Laboratories Report, 2013.

[12] OpenMP Application Program Interface. Technical report, OpenMP Architecture Review Board, 2013.

[13] R. P. Pawlowski, E. T. Phipps, and A. G. Salinger. Automating embedded analysis capabilities and managing software complexity in multiphysics simulation, Part I: Template-based generic programming. *Sci. Program.*, 20(2):197–219, April 2012.

[14] A.G. Salinger, R.A. Bartlett, Q. Chen, X. Gaoand G.A. Hansen, I. Kalashnikova, A. Mota, R.P. Muller, E. Nielsen, J.T. Ostien, R.P. Pawlowski, E.T. Phipps, and W. Sun. Albany: A component-based partial differential equation code built on Trilinos. Technical report, Sandia National Labs, SAND2013-8430J, 2013.

[15] A. J. Simmons and D. M. Burridge. An energy and angular-momentum conserving vertical finite-difference scheme and hybrid vertical coordinates. *Mon. Weather Rev.*, 109:758–766, 1981.

[16] L. Sirovich. Turbulence and the dynamics of coherent structures, Part III: dynamics and scaling. *Q. Appl. Math.*, 45(3):583–590, 1987.

[17] M. A. Taylor. Conservation of mass and energy for the moist atmospheric primitive variables. In P. H. Lauritzen, C. Jablonowski, M. A. Taylor, and R. D. Nair, editors, *Numerical Techniques for Global Atmospheric Models*, chapter 12. Springer, 2012.

[18] Mark Taylor, Joseph Tribbia, and Mohamed Iskandarani. The spectral element method for the shallow water equations on the sphere. *Journal of Computational Physics*, 130:92–108, 1997.

[19] P. A. Ullrich, C. Jablonowski, J. Kent, P. H. Lauritzen, R. D. Nair, and M. A. Taylor. Dynamical Core Model Intercomparison Project (DCMIP) test case document. Technical report, National Center for Atmospheric Research, 2012. Available at `https://earthsystemcog.org/projects/dcmip-2012/test_cases`.

[20] D. L. Williamson, J. B. Drake, J. J. Hack, R. Jakob, and P. N. Swarztrauber. A standard test set for numerical approximations of the shallow water equations in spherical geometry. *J. Comput. Phys.*, 102:211–224, 1992.

DISTRIBUTION:

1	Irina P. Demeshko Los Alamos National Laboratory P.O. Box 1663 Los Alamos, NM 87545	
1	MS 0750	Erika L. Roesler, 6913
1	MS 0825	Jeffrey A. Fike, 1515
1	MS 0836	Steven W. Bova, 1541
1	MS 1318	Andrew G. Salinger, 1442
1	MS 1320	S. Scott Collis, 1440
1	MS 1320	William F. Spotz, 1446
1	MS 1320	James R. Overfelt, 1443
1	MS 1320	Thomas M. Smith, 1446
1	MS 1320	Mark A. Taylor, 1446
1	MS 1321	Randall M. Summers, 1446
1	MS 1321	Peter A. Bosler, 1446
1	MS 1322	Oksana Guba, 1441
1	MS 9158	Jerry Watkins, 8959
1	MS 9159	Irina K. Tezaur, 8959
1	MS 0359	Donna L. Chavez, LDRD Office, 1911
1	MS 0899	Technical Library, 9536 (electronic copy)

Sandia National Laboratories