# SUMMER PROCEEDINGS 2014

The Computing Research Center at Sandia National Laboratories

**Editors:**

Drew P. Kouri and Michael L. Parks
Sandia National Laboratories

December 18, 2014





SAND#: SAND2015-3829 O

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

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



Available to DOE and DOE contractors from

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

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

Available to the public from

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

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

## Preface

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

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

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

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

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

For the 2014 CR Proceedings, research articles have been organized into the following broad technical focus areas — *computational mathematics, applications, and software and high performance computing* — which are well aligned with Sandia's strategic thrusts in computer and information sciences.

We would like to thank all participants who have contributed to the outstanding technical accomplishments of CSRI and CERI in 2014 as documented by the high quality articles in this proceedings. The success of CSRI and CERI hinged on the hard work of 17 enthusiastic student collaborators and their dedicated Sandia technical staff mentors. It is truly impressive that the research described herein occurred primarily over a three month period of intensive collaboration.

CSRI and CERI benefited from the administrative help of Amy Levan, Phyllis Rutka, Darlene Aragon, Ashley Avallone, Sandra Portlock, Denise LaPorte, Lorena Martinez,
Bernadette Watts, Bill Goldman, John Perseo, and Bryan Trujillo. The success of CSRI and CERI is, in large part, due to their dedication and care, which are much appreciated. We would also like to thank those who reviewed articles for this proceedings — their feedback is an important part of the research training process and has significantly improved the quality of the papers herein.

Drew P. Kouri
Michael L. Parks

December 18, 2014

**Table of Contents**

**Preface**
*D.P. Kouri and M.L. Parks* . . . . . iii
**Computational Mathematics**
*D.P. Kouri and M.L. Parks* . . . . . 1
Risk-Averse Optimal Neumann Control
*X. Deng, D.P. Kouri, and D. Ridzal* . . . . . 3
A Multi-Timestepping Extension to the Peridynamic Theory
*P.E. Lindsay and M.L. Parks* . . . . . 15
Galerkin RBF Method: A Meshfree Method for the Discretization of Nonlocal Diffusion Equations
*S.T. Rowe and R.B. Lehoucq* . . . . . 26
A New Partitioned Algorithm for Explicit Elastodynamics Based on Variational Flux Recovery
*P. Kuberry and P. Bochev* . . . . . 38
A Conservative Semi-Lagrangian Spectal Element Method with Optimization Based Limiters
*S. A. Moe, P. B. Bochev, K. J. Peterson, D. Ridzal* . . . . . 54
Multilevel Methods for the Stochatic Galerkin Method for PDEs with Random Input Data
*S.V. Osborn and E.T. Phipps* . . . . . 67
LAMMPS Kokkos Package
*S.Y. Mashayak, C.R. Trott, and S.J. Plimpton* . . . . . 74
Force Convergence in Stopping Power from Molecular Dynamics
*P.M. Van Every and A.D. Baczewski and R.J. Magyar* . . . . . 89
**Applications**
*D.P. Kouri and M.L. Parks* . . . . . 103
Preliminary Residual Formulation of COBRA-TF
*C.A. Dances and V.A. Mousseau* . . . . . 105
ALEGRA and Equation of State Tables
*B.M. Kelley, S.V. Petney and D.M. Hensinger* . . . . . 125
Inversion Under Uncertainty for Trace Gases using Convection-Diffusion-Reaction
*H. Li, B.G. van Bloemen Waanders, T.M. Wildey* . . . . . 133
Angle Dependence of Oblique Impact Problems in Alegra
*J.M. Staten and A.C. Robinson* . . . . . 147
SQuadGen Grid Generation: A Tool to Further Improve the Atmospheric General Circulation Models
*S.M. Sullivan and M.A. Taylor* . . . . . 158
Optimization under uncertainty for the Shockley and the drift-diffusion models of a diode
*T.A. Takhtaganov, D.P. Kouri, D. Ridzal, E. Keiter* . . . . . 165
**Software and High Performance Computing**
*D.P. Kouri and M.L. Parks* . . . . . 175
Anasazi TraceMin User Manual
*A.M. Klinvex, M.A. Heroux, M.L. Parks, and K.D. Devine* . . . . . 177
Tensor Toolbox: Wrapping to Python using SWIG
*M.G. Peterson and D.M. Dunlavy* . . . . . 188

Implementing Parallel Algorithms Using the Dax Toolkit
*H. Schroots and K. Moreland* . . . . . 197
Strongly Connected Components on GPU
*G.M. Slota, S. Rajamanickam, and K. Madduri* . . . . . 208
A Function Shipping Layer for the Kitten Lightweight Kernel
*J.C. Cabrera and K.P. Pedretti* . . . . . 224
Balancing Power and Time of MPI Operations
*T. Groves and K.B. Ferreira* . . . . . 231
Delta: Data Reduction for Integrated Application Workflows
*G.J. Baptiste and G. Lofstead* . . . . . 241

## Computational Mathematics

The articles in this section discuss algorithms and discretizations for a physical application problems. This includes discretizations for local and nonlocal physics, algorithms for molecular dynamics, as well as implementation details on advanced computational architectures.

*Deng, Kouri, and Ridzal* discuss the use of Trilinos components, including the Rapid Optimization library (ROL), to solve optimization problems constrained by nonlinear parabolic partial differential equations. In particular, the authors focus on the optimal Neumann boundary control of nonlinear parabolic equations with uncertain coefficients and discuss the use of risk measures to mitigate this uncertainty. *Lindsay and Parks* introduce a modified peridynamics theory which allows for multi-timestepping. This modification accomodates differing time scales between physical subdomains and permits timestepping with different step sizes between subdomains. *Rowe and Lehoucq* discuss a meshfree discretization for nonlocal diffusion. Their meshfree approach is a Galerkin method based on novel localized radial basis functions. *Kuberry and Bochev* present a partitioned algorithm to handle interface problems arising in explicit elastodynamics. This algorithm is based on variational flux recovery and Kuberry and Bochev were able to prove second order accuracy in space. *Moe et al.* describe a conservative spectral element approach for hyperbolic physics which arise in atmospheric computations. Conservation and limiting are efficiently enforced through Optimization Based Remap. *Osborn and Phipps* describe a multilevel stochastic Galerkin approach to solving partial differential equations with uncertain coefficients. This approach is both multilevel in space and stochastic variables which permits low-order polynomial chaos for high-order spatial discretization. *Mashayak et al.* employ the Kokkos package to achieve performance improvements for the LAMMPS molecular dynamics package. Using Kokkos, the authors are able to port portions of LAMMPS to GPUs and other advanced architecures. *Van Every et al.* use time-dependent density functional theory to simulate a proton passing through several metals. The authors discuss discretization on a $k$-point grid and analyze the convergence of such an approximation.

D.P. Kouri
M.L. Parks

December 18, 2014

# RISK-AVERSE OPTIMAL NEUMANN CONTROL

XIAODI DENG*, DREW KOURI †, AND DENIS RIDZAL ‡

**Abstract.** In this work, we consider the optimal Neumann control of a nonlinear parabolic partial differential equation (PDE) with uncertain coefficients. PDE-constrained optimization problems arise in many engineering and scientific application problems. In particular, Neumann control problems arise when one can only influence the state of the physical system by modifying sources or sinks on the boundary. In addition, for real-world applications, the input data for the PDE model is often not known exactly, but rather measured from experimental data. For high-consequence applications, it is then essential to determine Neumann controls that are risk-averse or robust to the uncertainty in the input data. To handle this uncertainty, we employ risk measures. We conclude with some numerical results for both the deterministic and risk-averse problems.

**1. Introduction.** Simulation of physical systems is becoming an integral component of science and engineering research. As more weight is placed on computational simulation, advanced analyses such as optimization and uncertainty quantification of the physical system and computational simulation become increasingly important. When a simulation is used to influence a decision, it is essential to quantify and in some sense mitigate uncertainties in the simulation and physical model. In this work, we discuss optimization problems constrained by partial differential equations (PDEs) such as the optimal control or design of a physical system. When such control and design problems influence policy or decisions, it is critical to not just quantify uncertainties, but to mitigate uncertainty. To this end, we employ the concept of risk measures to determine optimal controls or designs that are, in some sense, insensitive to the uncertainty in model input data.

Optimization problems governed by PDEs with random and uncertain coefficients have been considered in [3, 4, 5, 10, 9, 13, 6, 14]. In [3, 4, 5, 10, 9, 6] the authors consider optimal control problems in which the controls are deterministic. For such problems, the authors consider minimizing the expected value of the random variable objective function. In contrast, the authors in [13, 14] consider stochastic controls and try to match different statistics associated with the random controls and objective functions such as moments or distribution functions.

In this work, we focus on the situation where the optimization variables (controls or designs) are deterministic. This models the realistic situation where one must determine a control action or design prior to observing the uncertainty. Moreover, we employ risk measures [11, 12] to mitigate the effects of uncertain PDE parameters.

This document is structured as follows. We first present the deterministic problem formulation. Second, we discuss how the problem changes when uncertainty is added. In this discussion, we present the notion of risk and discuss how risk is measured. Following the problem formulation, we discuss its implementation and conclude with numerical results.

**2. Deterministic Problem Formulation.** Let $\Omega \subset \mathbb{R}^3$ be a bounded domain with boundary $\partial\Omega$. Moreover, let $\Gamma_n \subset \partial\Omega$ denote the Neumann boundary and $\Gamma_r \subset \partial\Omega$ denote the Robin boundary. We assume that $\Gamma_n \cup \Gamma_r = \partial\Omega$ and $\Gamma_n \cap \Gamma_r = \emptyset$. Now, let $\mathcal{V} = \mathcal{V}(\Omega) \subset L^2(\Omega)$ be a reflexive Banach space of sufficiently regular functions on $\Omega$ such that $\mathcal{V} \hookrightarrow L^2(\Omega) \hookrightarrow \mathcal{V}^*$ is a Gelfand triple. Moreover, for any $T > 0$, we define

*Department of Computational And Applied Mathematics, Rice University, xiaodi.deng@rice.edu
†Sandia National Laboratories, dpkouri@sandia.gov
‡Sandia National Laboratories, dridzal@sandia.gov

the control space $\mathcal{Z} = L^2(0,T;L^2(\Gamma_n))$, and the state space

$$\mathcal{W} = W(0,T;L^2(\Omega),\mathcal{V}) = \{\, v : [0,T] \to \mathcal{V} \,:\, v \in L^2(0,T;\mathcal{V}),\, v_t \in L^2(0,T;\mathcal{V}^*) \,\}$$

where $v_t$ denotes the weak time derivative of $v$. We consider optimization problems governed by the following weak-form nonlinear parabolic PDE with time-independent coefficients

$$u_t + \mathbf{L}u + \mathbf{N}(u) = \mathbf{B}f + \mathbf{b} \tag{2.1a}$$

$$u(0) = u_0 \tag{2.1b}$$

where $\mathbf{L} \in \mathcal{L}(\mathcal{V},\mathcal{V}^*)$, $\mathbf{N} : \mathcal{V} \to \mathcal{V}^*$, $\mathbf{B} \in \mathcal{L}(\mathcal{Z},\mathcal{V}^*)$, $\mathbf{b} \in \mathcal{V}^*$, and $u_0 \in L^2(\Omega)$. For our examples, $\mathbf{L}$ is defined as

$$\langle \mathbf{L}u, v\rangle_{\mathcal{V}^*,\mathcal{V}} = \int_\Omega (\kappa(x)\nabla u(x)) \cdot \nabla v(x)\,\mathrm{d}x + \int_{\Gamma_r} \sigma(x)u(x)v(x)\,\mathrm{d}x \quad \forall\, u,\, v \in \mathcal{V} \tag{2.2}$$

for some $\kappa : \Omega \to \mathbb{R}^{3\times 3}$ satisfying $\kappa_{ij} \in L^\infty(\Omega)$ for $i,\, j = 1,2,3$ and $\kappa(x)$ is positive definite for almost all $x \in \Omega$, and $\sigma \in L^\infty(\Gamma_r)$. In addition, $\mathbf{B}$ is

$$\langle \mathbf{B}f, v\rangle_{\mathcal{V}^*,\mathcal{V}} = \int_{\Gamma_n} f(x)v(x)\,\mathrm{d}x \tag{2.3}$$

for any $f \in L^2(\Gamma_n)$ and $\mathbf{b}$ is

$$\langle \mathbf{b}, v\rangle_{\mathcal{V}^*,\mathcal{V}} = \int_\Omega l(x)v(x)\,\mathrm{d}x + \int_{\Gamma_r} g(x)v(x)\,\mathrm{d}x \tag{2.4}$$

for some functions $l : \Omega \to \mathbb{R}$ and $g : \Gamma_r \to \mathbb{R}$. Two common examples of the nonlinear term $\mathbf{N}$ are

$$\langle \mathbf{N}(u), v\rangle_{\mathcal{V}^*,\mathcal{V}} = \int_\Omega u^2(x)v(x)\,\mathrm{d}x \quad \text{and} \quad \langle \mathbf{N}(u), v\rangle_{\mathcal{V}^*,\mathcal{V}} = \int_{\Gamma_r} u^4(x)v(x)\,\mathrm{d}x.$$

The second nonlinearity is often referred to as a Stefan-Boltzman radiation boundary condition. We assume that $\mathcal{V}$ is chosen in such a way that (2.1) has a unique solution $u = u(f) \in \mathcal{W}$ for any $f \in \mathcal{Z}$.

Now, let $\bar{u} : L^2(0,T;L^2(\Omega))$, $w \in L^\infty(\Omega)$, and $\gamma_i > 0$ for $i = 1,2,3$. We consider the objective function $J : \mathcal{W} \times \mathcal{Z} \to \mathbb{R}$, defined as

$$\begin{aligned} J(u,f) =& \frac{\gamma_1}{2}\int_0^T \int_\Omega w(x)(u(x,t) - \bar{u}(x,t))^2\,\mathrm{d}x\mathrm{d}t \\ &+ \frac{\gamma_2}{2}\int_\Omega w(x)(u(x,T) - \bar{u}(x,T))^2\,\mathrm{d}x \\ &+ \frac{\gamma_3}{2}\int_0^T \int_{\Gamma_n} f^2(x,t)\,\mathrm{d}x\mathrm{d}t. \end{aligned} \tag{2.5}$$

Thus, the optimization problem of interest for this work is

$$\min_{f\in\mathcal{Z}} \widehat{J}(f) = J(u(f),f), \tag{2.6}$$

where $u(f) = u \in \mathcal{W}$ is the solution of (2.1).

PROPOSITION 2.1. *Suppose for each $f \in \mathcal{Z}$, there exists a unique solution to (2.1). Let the solution operator for (2.1) be $f \mapsto u(f) : \mathcal{Z} \to \mathcal{W}$ and assume $f \mapsto u(f)$ satisfies:*

$$\{f_n\} \subset \mathcal{Z} \quad \textit{such that} \quad f_n \rightharpoonup f \in \mathcal{Z} \quad \Longrightarrow \quad u(f_n) \rightharpoonup u(f) \in \mathcal{W}.$$

*Then, there exists $f^* \in \mathcal{Z}$ such that $\widehat{J}(f) \geq \widehat{J}(f^*)$ for all $f \in \mathcal{Z}$.*

*Proof.* First, note that $\widehat{J}$ is well-defined since we assumed existence and uniqueness of a solution to (2.1) for each $f \in \mathcal{Z}$. Clearly, $\widehat{J}$ is coercive since

$$\widehat{J}(f) \geq \frac{\gamma_3}{2} \int_0^T \int_{\Gamma_n} f^2(x,t)\,\mathrm{d}x\mathrm{d}t \quad \forall\, f \in \mathcal{Z}.$$

Thus, let $\{f_n\} \subset \mathcal{Z}$ be an infimizing sequence of $\widehat{J}$. Since $\widehat{J}$ is coercive, $\{f_n\}$ must be bounded. Moreover, since $\mathcal{Z}$ is a Hilbert space (i.e., it is reflexive), there exists a weaky converging subsequence of $\{f_n\}$ denoted by $\{f_{n_j}\}$ with weak limit $f^* \in \mathcal{Z}$. Now, since $\widehat{J}$ is a sum of weighted $L^2$-norms squared with weight $w \in L^\infty(\Omega)$ and $f \mapsto u(f)$ satisfies the assumptions of theorem, we have that $\widehat{J}$ is weakly sequentially lower semicontinuous. Therefore,

$$\widehat{J}(f) \geq \lim_{n\to\infty} \widehat{J}(f_n) \geq \liminf_{j\to\infty} \widehat{J}(f_{n_j}) \geq \widehat{J}(f^*)$$

for any $f \in \mathcal{Z}$, thus concluding the proof. □

**3. Uncertain Coefficients.** Let $(\Theta, \mathcal{F}, P)$ be a complete probability space. Here, $\Theta$ is the set of outcomes, $\mathcal{F} \subseteq 2^\Theta$ is a $\sigma$-algebra of events, and $P : \mathcal{F} \to [0,1]$ is a probability measure. We now consider the nonlinear parabolic problem with random coeffcients

$$u_t(\theta) + \widehat{\mathbf{L}}(\theta)u(\theta) + \widehat{\mathbf{N}}(u(\theta),\theta) = \widehat{\mathbf{B}}(\theta)f + \widehat{\mathbf{b}}(\theta) \tag{3.1a}$$

$$u(\theta,0) = \widehat{u}_0(\theta) \tag{3.1b}$$

where $\widehat{\mathbf{L}} : \Theta \to \mathcal{L}(\mathcal{V},\mathcal{V}^*)$, $\widehat{\mathbf{N}} : \mathcal{V} \times \Theta \to \mathbb{R}$, $\widehat{\mathbf{B}} : \Theta \to \mathcal{L}(L^2(\Gamma_n),\mathcal{V}^*)$, $\widehat{\mathbf{b}} : \Theta \to \mathcal{V}^*$, and $\widehat{u}_0 : \Theta \to L^2(\Omega)$. We require that (3.1) holds for $P$-almost every $\theta \in \Theta$. In addition the solution to (3.1) is a random field. We assume the random field solution, $\theta \mapsto u(\theta) : \Theta \to \mathcal{W}$ is a member of the Banach space $L^2_P(\Theta;\mathcal{W})$.

To facilitate numerical computation, we assume the Finite-Dimensional Noise Assumption [1] holds. That is, there exists a finite-dimensional random vector $\xi : \Theta \to \mathbb{R}^M$ with component random variables $\xi_i : \Theta \to \Xi_i$ where $\Xi_i$ is a connected interval of $\mathbb{R}$ such that for $P$-almost all $\theta \in \Theta$ we have $\widehat{\mathbf{L}}(\theta) = \mathbf{L}(\xi(\theta))$, $\widehat{u}_0(\theta) = u_0(\xi(\theta))$, $\widehat{\mathbf{N}}(u,\theta) = \mathbf{N}(u,\xi(\theta))$ for any $u \in \mathcal{V}$, $\widehat{\mathbf{B}}(\theta) = \mathbf{B}(\xi(\theta))$, and $\widehat{\mathbf{b}}(\theta) = \mathbf{b}(\xi(\theta))$ for some $\mathbf{L} : \Xi \to \mathcal{L}(\mathcal{V},\mathcal{V}^*)$, $\mathbf{N} : \mathcal{V} \times \Xi \to \mathbb{R}$, $\mathbf{B} : \Xi \to \mathcal{L}(L^2(\Gamma_n),\mathcal{V}^*)$, $\mathbf{b} : \Xi \to \mathcal{V}^*$, and $u_0 : \Xi \to L^2(\Omega)$ where $\Xi = \Xi_1 \times \cdot \times \Xi_M$. Moreover, we assume the component random variables $\xi_i$ have Lebesgue density $\rho_i$ and define the joint density $\rho = \rho_1 \otimes \cdots \otimes \rho_M$. Under this assumption, (3.1) becomes the parametric PDE

$$u_t(\xi) + \mathbf{L}(\xi)u(\xi) + \mathbf{N}(u(\xi),\xi) = \mathbf{B}(\xi)f + \mathbf{b}(\xi) \tag{3.2a}$$

$$u(\xi,0) = u_0(\xi) \tag{3.2b}$$

and the solution belongs to the Banach space $L^2_\rho(\Xi;\mathcal{W})$. The Finite-Dimension Noise Assumption is often satisfied by a truncated Karhunen-Loeve expansion and permits

the use of numerous discretizations based on sampling and projection onto orthogonal polynomials.

Now, if we assume that for each $f \in \mathcal{Z}$, there exists a unique $u(f) = u \in L^2_\rho(\Xi;\mathcal{W})$, then we have that $\widehat{J}(f) : \Xi \to \mathbb{R}$ is a random variable. If we further assume that $\widehat{J}(f) \in L^p_\rho(\Xi)$ for some $1 \leq p < \infty$ for all $f \in \mathcal{Z}$, then we can choose any $\mathcal{R} : L^p_\rho(\Xi) \to \mathbb{R} \cup \{\infty\}$ and solve the optimization problem

$$\min_{f \in \mathcal{Z}} \mathcal{R}(\widehat{J}(f)).$$

In financial mathematics and risk management, $\mathcal{R}$ is called a risk measure or risk function. Since $\mathcal{R}$ is a blank slate, the engineer, scientist, or decision maker can inject their own aversity to risk in the definition of $\mathcal{R}$. Common choices of $\mathcal{R}$ are the expected value, the mean-plus-deviation, and the conditional value-at-risk (which corresponds to a conditional expectation over a prescribed quantile). An in-depth discussion of risk measures is beyond the scope of this paper. We refer the interested reader to [11, 12] for a nice overview.

**4. Implementation and Numerical Results.** In this section, we describe the discretization of (2.1) in the presence of uncertain coefficients. There are numerous approaches to discretize a PDE with uncertain coefficients. Two prevelent classes of discretizations are projection methods such as stochastic Galerkin and polynomial chaos [2, 8], and sample-based methods such as Monte Carlo and stochastic collocation [1]. In this work, we focus on sample-based methods. Moreover, we discuss the use of Trilinos [7] components to aid in this discretization and the solution of the optimization problem.

**4.1. Implementation.** To discretize in physical space, we subdivided $\Omega$ with a quadrilateral mesh. We generate this mesh using the STK capabilities in Panzer. Moreover, we manage our degrees of freedom (both for the PDE solution and the control variables) using Panzers degree of freedom manager. We employ standard continuous piecewise polynomial finite elements to generate the spatial discretization. To compute the corresponding cell-local information such as cell stiffness and mass, we utilize Intrepid. Additionally, we use template-based automatic differentiation implemented in Sacado to compute cell-residual Jacobians. To facillitate parallel computation, we store finite element coefficients in Epetra multivectors. To discretize in time, we employ a uniform partition of the temporal domain $[0, T]$. We then timestep using backward Euler. Since backward Euler is an implicit scheme, we use AztecOO for linear solves and NOX for nonlinear equation solves at each backward Euler time step. We precondition these solves using the algebraic multigrid preconditioners in ML. Finally, to discretize the uncertain variables, we use the Monte Carlo and adaptive sparse grid techniques of the Rapid Optimization Library (ROL). To discretize these uncertain variables, we replace any expected values in the risk measure $\mathcal{R}$ with quadrature approximations. These quadrature approximations then induce a sample-based discretization of the parametrized PDE. Once the problem is discretized, we employ ROL's suite of optimization algorithms to solve the resulting large-scale nonlinear programming problem.

**4.2. Numerical Results.** Throughout this section, we let $\Omega = [0, 1]^3$ and adopt the notation $x_1, x_2, x_3$ to denote three coordinates of $x \in \Omega$. We begin with a few deterministic examples and conclude with an example with uncertain coefficients.

**4.2.1. A Linear Parabolic Optimal Control Problem.** For this example, we assume the temperature outside of $\Omega$ is 0 and the initial temperature inside of $\Omega$ is also 0. That is, $u_0 = 0$. We place a heat source around the center of the surface $x_3 = 0$ in the form of linear Robin boundary conditions. Moreover, we apply the control to the remaining five faces. Our objective for this example is to match the volumetric temperature at the final time to zero. The differential operator $\mathbf{L}$ for this problem is given by (2.2) with $\kappa(x) = \mathbf{I}$ where $\mathbf{I}$ denotes the $3 \times 3$ identity matrix and $\sigma \equiv 0.01$. The control operator $\mathbf{B}$ is given by (2.3) and the source term is given by (2.4) where $g(x) = 0$ and $l(x)$ is defined as

$$l(x) = 20[\cos(4\pi\sqrt{(x-0.5)^2 + (y-0.5)^2 + z^2}) + 1]$$

if $\sqrt{(x-0.5)^2 + (y-0.5)^2 + z^2} < 0.25$ and 0 otherwise. Finally, the nonlinearity is $\mathbf{N} \equiv 0$ and the objective function for this example is

$$J(u, f) = \frac{1}{2}\int_\Omega u^2(x, T)\,\mathrm{d}x + \frac{10^{-4}}{2}\int_0^T \int_{\Gamma_n} f^2(x, t)\,\mathrm{d}x\mathrm{d}t.$$

The results are displayed in Figure 4.1. These results display an oscillating control that first cools and then warms. The control repeats this pattern throughout the time interval and forms multiple ring-shaped temperature regions. These rings have the effect of first cooling the cube to below 0, then heating the cube to cancel the low temperature caused by the initial cooling. If viewed in more refined time steps, this cooling-warming pattern repeats in decreasing magnitude. In the final snapshot, temperature rings caused by the control are nearly cancelled and the heat on the face $x_3 = 0$ is negated by the first cooling ring produced.

**4.2.2. A Semilinear Parabolic Optimal Control Problem.** In this example, we set the temperature outside of $\Omega$ to be 0 and initial temperature inside of $\Omega$ to also be 0. Our goal is to control the heat flux on opposite sides of the cube so that the cylinder connecting the centers of control faces (around $x_1 = x_2 = 0.5$) has temperature close to 10 throughout the time interval $[0, T]$. In the half of the cube defined by $x_1 < 0.5$, the conductivity in the $x_3$ direction is 100 times larger than in other directions. In the other half of the cube, $x_1 \geq 0.5$, conductivity is isotropic and identically 1. In addition to the discontinuous heat conductivity, we also consider a quadratic reaction term resulting in a semilinear parabolic PDE. The differential operator $\mathbf{L}$ is defined by (2.2) with $\kappa(x)$ satisfying

$$\kappa(x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 100 \end{bmatrix}$$

if $x_1 < 0.5$ and $\kappa(x) = \mathbf{I}$ otherwise, and $\sigma \equiv 0.01$. The control operator $\mathbf{B}$ is defined by (2.3) and source term is $\mathbf{b} \equiv 0$. The nonlinearity $\mathbf{N}$ for this problem is

$$\langle \mathbf{N}(u), v\rangle_{\mathcal{V}^*,\mathcal{V}} = \int_\Omega u^2(x)v(x)\,\mathrm{d}x.$$

Finally, the objective function for this example is

$$J(u, f) = \frac{1}{2}\int_0^T \int_\Omega w(x)(u(x, t) - \bar{u}(x, t))^2\,\mathrm{d}x\mathrm{d}t + \frac{10^{-4}}{2}\int_0^T \int_{\Gamma_n} f^2(x, t)\,\mathrm{d}x\mathrm{d}t$$

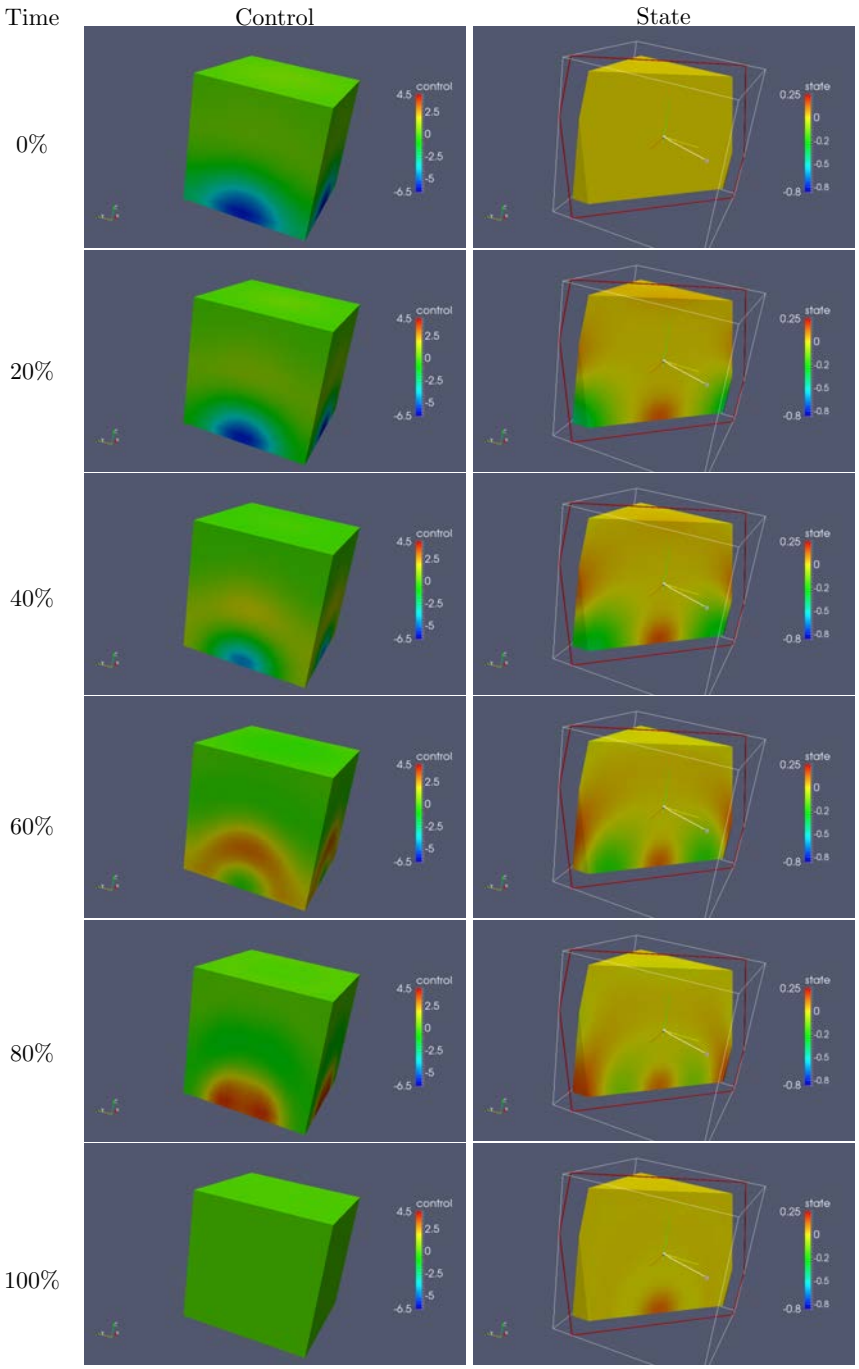



Fig. 4.1: Optimal control and optimal state for deterministic linear parabolic problem.

where $\bar{u}$ and $w$ are given by

$$\bar{u}(x,t) = \begin{cases} 10, & \text{if } |x_1 - 0.5| < 0.1 \text{ and } |x_2 - 0.5| < 0.1 \\ 0, & \text{otherwise} \end{cases}$$

$$w(x) = \begin{cases} 1, & \text{if } |x_1 - 0.5| < 0.1 \text{ and } |x_2 - 0.5| < 0.1 \\ 0, & \text{otherwise} \end{cases},$$

respectively.

The optimal heat flux and corresponding state are depicted in Figure 4.2. The optimal flux has larger magnitude on the side $x_1 < 0.5$ which corresponds to large heat conductivity in the $x_3$ direction. This result is due to the fact that the heat profile on the $x_1 \geq 0.5$ side diffuses at a much higher rate in all directions. Near $t = 0.5T$, the temperature at the center of $\Omega$ is approximately the target temperature, and as time advances, the magnitude of the heat flux decreases.

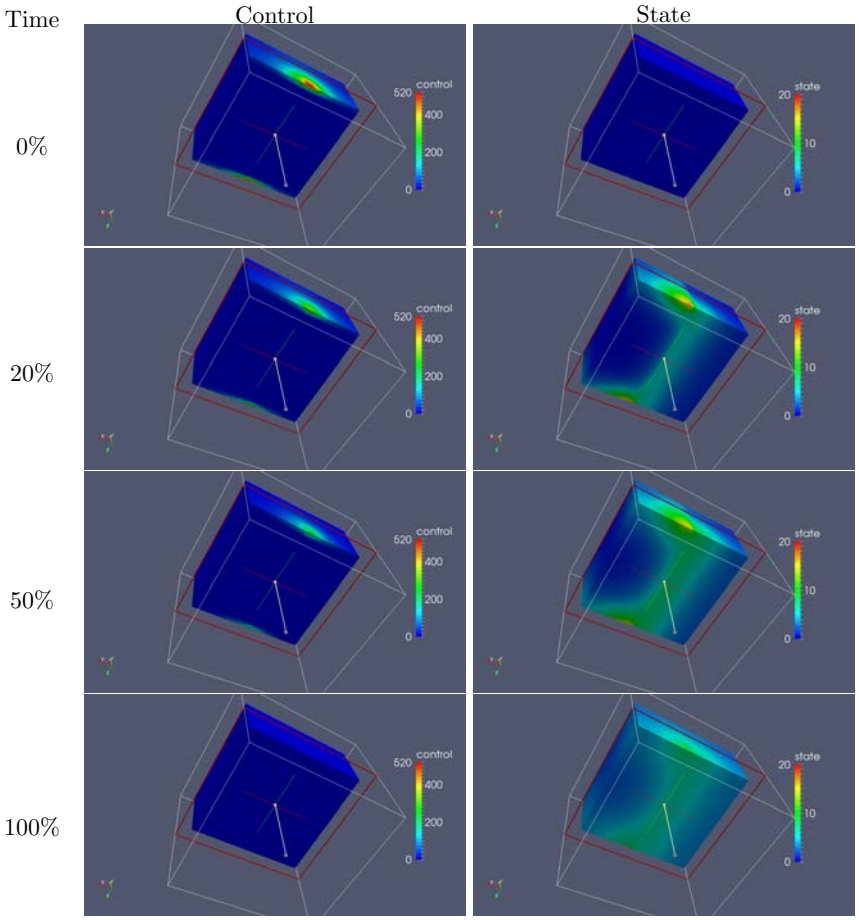



Fig. 4.2: Optimal control and optimal state for deterministic semilinear parabolic problem.

**4.2.3. A Linear Parabolic Optimal Control Problem with Time-Dependent Target.** For this example, we attempt to match a time-dependent target temperature profile by controlling the Neumann flux on all faces but $x_1 = 0$ and $x_1 = 1$. The target is moving an area of high temperature counter-clockwise with respect to the $x_3$ axis. The differential operator $\mathbf{L}$ is given by (2.2) with $\kappa(x) \equiv \mathbf{I}$ and $\sigma \equiv 0.01$. The control operator $\mathbf{B}$ is again given by (2.3) and the source is given by

$\mathbf{b} \equiv 0$. The nonlinearity is $\mathbf{N} \equiv 0$ as well. Finally, the objective function is given by

$$J(u,f) = \frac{1}{2}\int_0^T \int_\Omega w(x)(u(x,t) - \bar{u}(x,t))^2 \,\mathrm{d}x\mathrm{d}t + \frac{10^{-4}}{2}\int_0^T \int_{\Gamma_n} f^2(x,t)\,\mathrm{d}x\mathrm{d}t$$

where $w$ is given by

$$w(x) = \begin{cases} 1 & \text{if } \|x - 0.5\|_\infty < 0.25 \\ 0 & \text{otherwise.} \end{cases}$$

Moreover, the time dependent target $\bar{u}$ is implemented with the following code.

```
double state_objective_value (double x,double y,double z,double t){

  double T=0.03; //Half of full simulation time
  while(t>T)t-=T;
  double cx,cy,cz; //Center of heated area

  if(t < T * 0.25){
    double p = t / (T * 0.25);
    cx = 0.375 + p * 0.25;
    cy = 0.375;
    cz = 0.5;
  }

  if( T * 0.25 <= t && t< T * 0.5 ){
    double p = (t - T * 0.25) / (T/4);
    cx = 0.625;
    cy = 0.375 + p * 0.25;
    cz = 0.5;
  }

  if( T * 0.50 <= t && t< T * 0.75 ){
    double p = (t - T * 0.5) / (T/4);
    cx = 0.625 - p * 0.25;
    cy = 0.625;
    cz = 0.5;
  }

  if( T * 0.75 <= t ){
    double p = (t - T * 0.75) / (T/4);
    cx = 0.375;
    cy = 0.625 - p * 0.25;
    cz = 0.5;
  }

  if( abs(x-cx)<0.125 && abs(y-cy)<0.125 && abs(z-cz)<0.125)return 1;
  return -1;
}
```

The optimal controls and states are depicted in Figure 4.3. We observe that the control on the four sides causes the heated area to move around counter-clockwise to match the target temperature profile. Though we are able to create a counter-clockwise temperature profile, we do not exactly match the target function due to the nature of diffusion.

**4.2.4. A Semilinear Parabolic Optimal Control Problem with Uncertain Initial Conditions.** In this final example, we consider the initial condition to be uncertain. We assume the initial temperature inside of $\Omega$ is a random perturbation

of 0. That is $u_0 = \xi$. We implement two different distributions for this pertubation. In one case, we assume the perturbation $\xi$ is uniformly distributed in $[-1, 0.5]$ while in the second case we assume the perturbation is uniformly distributed in $[-0.5, 1]$. In addition, we attempt to match the heat flux on a single face of $\Omega$ by controlling the heat flux on the remaining faces. For this example, we consider the risk neutral setting where $\mathcal{R}$ is the expected value with respect to the distribution $\rho$. This describes the scenario in which we must maintain a fixed tempurature in some region of $\Omega$ in the face of uncertain initial conditions. The boundary conditions away from the control boundaries are the Stefan-Boltzman radiation condition. Thus, the PDE is semilinear. The differential operator $\mathbf{L}$ is given by (2.2) with $\kappa \equiv \mathbf{I}$ and $\sigma \equiv 0$. The control operator $\mathbf{B}$ is again given by (2.3) and the source is $\mathbf{b} \equiv 0$. The nonlinearity $\mathbf{N}$ is given by Stefan-Boltzman boundary conditions, i.e.,

$$\langle \mathbf{N}(u), v \rangle_{\mathcal{V}^*,\mathcal{V}} = \int_{\Gamma_r} u^4(x) v(x) \, \mathrm{d}x.$$

Finally, the parametric objective function for this example is

$$J(u(\xi), f) = \frac{1}{2} \int_\Omega u^2(\xi, x, T) \, \mathrm{d}x + \frac{10^{-2}}{2} \int_0^T \int_{\Gamma_n} f^2(x, t) \, \mathrm{d}x \mathrm{d}t.$$

The results for the two different distributions of $\xi$ are depicted in Figure 4.4. This figure displays the initial optimal Neumann control. The results for $\xi \in [-1, 0.5]$ demonstrate that heating is required during the entire time interval $[0, T]$. On the other hand, the results for $\xi \in [-0.5, 1]$ demonstrate that cooling is required during the entire interval $[0, T]$. Although there are scenarios in both cases that have positive and negative initial values, the optimal control tends to heat when there is a larger chance of negative initial temperature, and tends to cool when there is a larger chance of positive initial temperature.

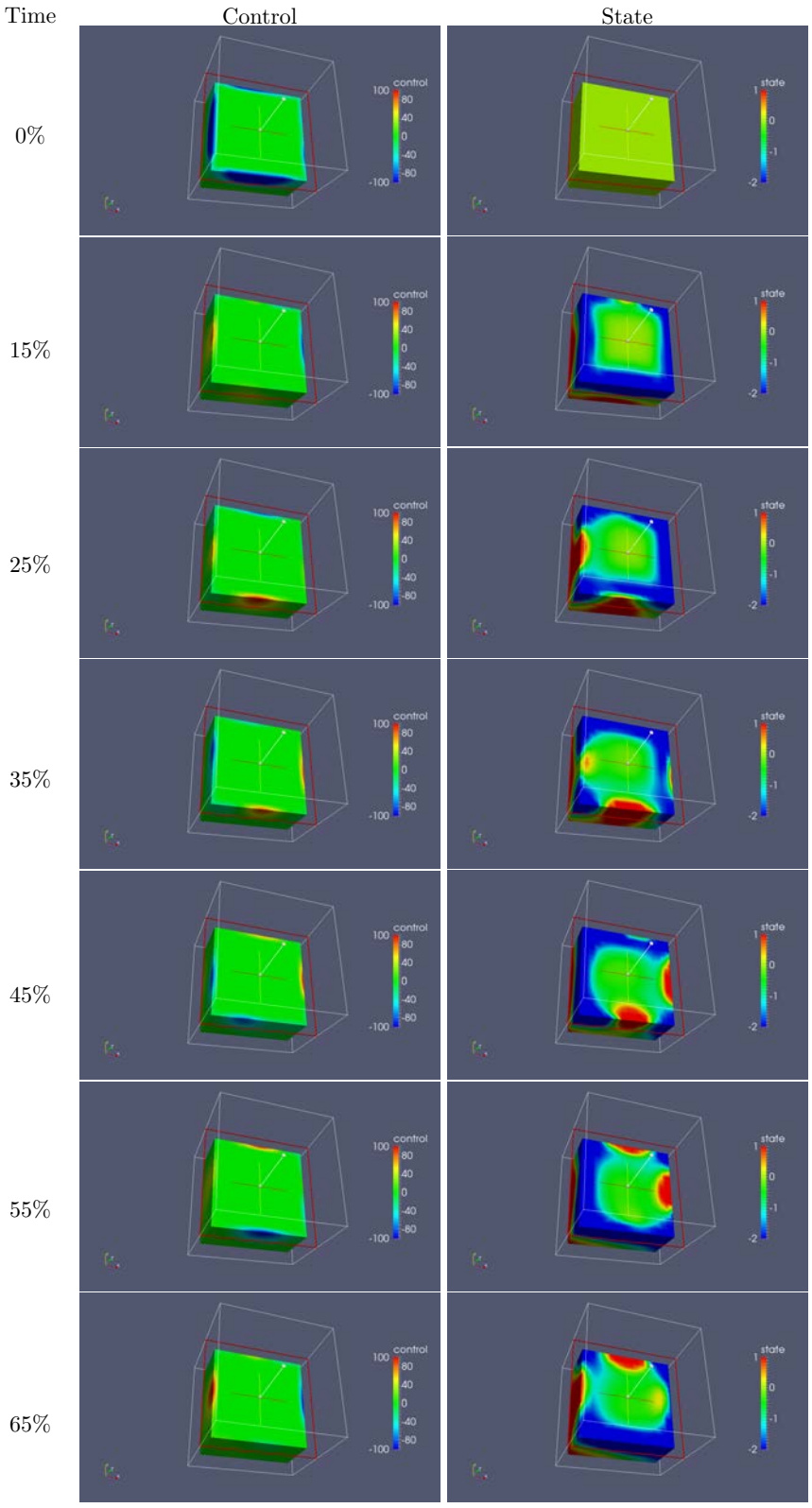



Fig. 4.3: Optimal control and optimal state for deterministic linear parabolic problem with a moving target.

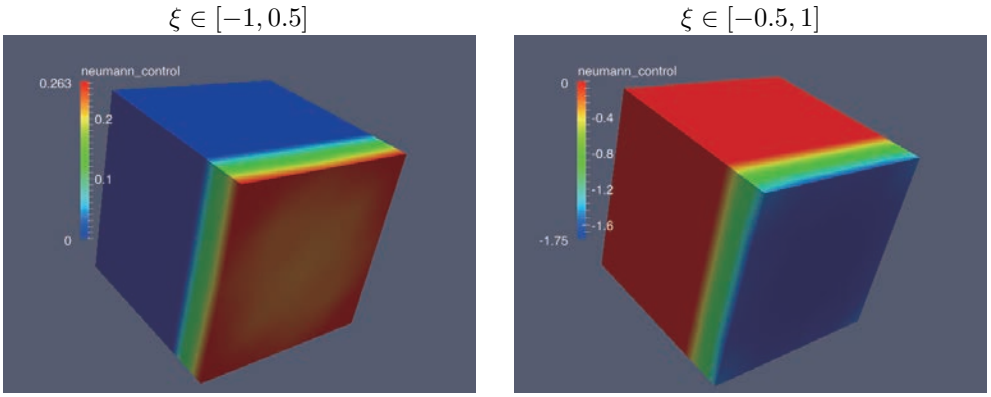



Fig. 4.4: Control at time zero for two different distributions of the random initial condition $\xi$.

**5. Conclusions.** In this paper, we discuss the optimal Neumann control of a nonlinear parabolic PDE and we extend the deterministic control problem to handle PDEs with uncertain coefficients. For this problem, we discuss the treatment of uncertainties in PDE input data. In particular, we employ risk measures to quantify and mitigate the risk associated with the parametric uncertainties. In addition, we discuss the implementation of such PDE-constrained optimization problems using Trilinos components and present numerical examples exercising this implementation.

REFERENCES


[1] I. Babuška, F. Nobile, and R. Tempone, *A stochastic collocation method for elliptic partial differential equations with random input data*, SIAM Rev., 52 (2010), pp. 317–355.

[2] I. Babuška, R. Tempone, and G. E. Zouraris, *Galerkin finite element approximations of stochastic elliptic partial differential equations*, SIAM J. Numer. Anal., 42 (2004), pp. 800–825 (electronic).

[3] A. Borzì, *Multigrid and sparse-grid schemes for elliptic control problems with random coefficients*, Comput. Vis. Sci., 13 (2010), pp. 153–160.

[4] A. Borzì, V. Schulz, C. Schillings, and G. von Winckel, *On the treatment of distributed uncertainties in PDE constrained optimization*, GAMM Mitteilungen, 33 (2010), pp. 230–246.

[5] A. Borzì and G. von Winckel, *A POD framework to determine robust controls in pde optimization*, Comput. Vis. Sci., 14 (2011), pp. 91–103.

[6] P. Chen, A. Quarteroni, and G. Rozza, *Stochastic optimal robin boundary control problems of advection-dominated elliptic equations*, SIAM Journal on Numerical Analysis, 51 (2013), pp. 2700–2722.

[7] M. Heroux, R. Bartlett, V. H. R. Hoekstra, J. Hu, T. Kolda, R. Lehoucq, K. Long, R. Pawlowski, E. Phipps, A. Salinger, H. Thornquist, R. Tuminaro, J. Willenbring, and A. Williams, *An Overview of Trilinos*, Tech. Rep. SAND2003-2927, Sandia National Laboratories, 2003.

[8] G. E. Karniadakis, C.-H. Su, D. Xiu, D. Lucor, C. Schwab, and R. A. Todor, *Generalized polynomial chaos solution for differential equations with random inputs*, Tech. Rep. 2005-01, Seminar for Applied Mathematics, ETH Zurich, Zurich, Switzerland, 2005.

[9] D. P. Kouri, *A multilevel stochastic collocation algorithm for optimization of pdes with uncertain coefficients*, SIAM/ASA J. Uncertainty Quantification, 2 (2014), pp. 55–81.

[10] D. P. Kouri, M. Heinkenschloss, D. Ridzal, and B. G. van Bloemen Waanders, *A trust-region algorithm with adaptive stochastic collocation for pde optimization under uncertainty*, SIAM J. Sci. Comput., 35 (2013), pp. A1847–A1879.

[11] A. Ruszczyński and A. Shapiro, *Optimization of risk measures*, in Probabilistic and Randomized Methods for Design Under Uncertainty, G. Calafiore and F. Dabbene, eds., London, 2006, Springer Verlag, pp. 119–157.

[12] A. Shapiro, D. Dentcheva, and A. Ruszczynski, *Lectures on Stochastic Programming: Modeling and Theory*, SIAM, Philadelphia, 2009.

[13] H. Tiesler, R. M. Kirby, D. Xiu, and T. Preusser, *Stochastic collocation for optimal control problems with stochastic PDE constraints*, SIAM J. Control Optim., (2012). accepted for publication.

[14] N. Zabaras and B. Ganapathysubramanian, *A scalable framework for the solution of stochastic inverse problems using a sparse grid collocation approach*, J. Comput. Phys., 227 (2008), pp. 4697–4735.

# A MULTI-TIMESTEPPING EXTENSION TO THE PERIDYNAMIC THEORY

PAYTON E. LINDSAY * AND MICHAEL L. PARKS †

**Abstract.** An extension to the basic theory of peridynamics is developed which enables the use of differing timescales in different regions of the problem domain. Peridynamics, which is a nonlocal extension of classical mechanics, is well-suited for solving many types of problems which would cause difficulties for most classical approaches such as modeling fracture. Multi-timestepping is a method developed for structural dynamics which enables the use of different spatial and temporal scales in different parts of a problem domain. This paper is an attempt to apply the method of multi-timestepping to peridynamics, and by extension to allow for larger and more complex problems to be modeled using the peridynamic approach. A proof of concept code was also developed, and numerical results using this code are included to illustrate the method.

**1. Introduction.** The classical theory of local solid mechanics is extremely versatile and has been used for many years to model a wide range of physical phenomena. However, there are many areas of interest for which classical solid mechanics ceases to give accurate results. One of the most frequently seen examples of this is spontaneous crack growth and propagation. Spontaneous crack growth is a common behavior in physical problems, and yet the classical theory of solid mechanics alone cannot capture this type of behavior. The reason for this is that the partial differential equations used in the classical theory are undefined along discontinuities. Because of this restriction, the classical theory must be augmented with some additional approach to capture cracking behavior. Peridynamics is a nonlocal extension of classical solid mechanics that attempts to circumvent this difficulty entirely by rewriting the basic equations of mechanics in terms of integral equations. In contrast with the differentiation used in the classical theory, integration is well defined along discontinuities, and thus modeling cracking behavior becomes trivial when using this approach.

Another limitation to classical solid mechanics is the computational burden associated with solving very large or complex problems. One approach that has been used with much success to reduce computational costs of classical problems is that of the multi-timestepping approach of Prakash and Hjelmstad [5, 4]. Multi-timestepping allows a problem domain to be divided into multiple subdomains, each with possibly varying timesteps and integration techniques, and then coupled together to accurately capture the behavior of the whole domain. Multi-timestepping has been used with great success to reduce the computational burden of solving large or complex problems while still retaining the accuracy and stability of the underlying methods.

In this paper we extend multi-timestepping to nonlocal models, namely peridynamics, using the same fundamental concepts as the multi-timestepping approach originally developed for classical methods. For this research, the theory for the simplest case of a 1D non-local spring system was first developed, and then then extended to accommodate full 3D systems. The goal of this research is to show that two peridynamic subdomains at different timesteps can be efficiently coupled, and furthermore that the coupled system accurately represents the response exhibited by the original model. A multi-timestep peridynamic code was also developed to demonstrate these results numerically. The results from the coupled system were compared with those from an unmodified peridynamics model to verify accuracy. A simple cost analysis is

*Purdue University, plindsay@purdue.edu
†Sandia National Laboratories, mlparks@sandia.gov

also included to demonstrate the efficiency of this method compared to unmodified peridynamics.

The remainder of this paper proceeds as follows. Section 2 gives a more detailed introduction to the multi-timestepping method, while section 3 presents a more thorough overview of peridynamics and introduces the basic peridynamic theory. Section 4 introduces the theory of multi-timestepping applied to full three-dimensional peridynamics and gives a simple cost analysis of the new method compared to unmodified peridynamics. Finally, section 5 provides numerical results and discussion, and section 6 offers some concluding remarks.

**2. Multi-Timestepping Background.** Fracture modeling has traditionally involved very large and costly computations. One of the main reasons for this is that the accuracy and stability requirements in a cracked region necessitates a much smaller mesh spacing and time step than would otherwise be required for the uncracked material. In many simulations, enforcing these tighter bounds over the entire region would seem to be a needlessly restrictive exercise, especially if the majority of the material remains uncracked. A method that allows for a relaxation of the temporal and spatial restrictions in regions that do not require these more stringent bounds while still retaining the accuracy of the original model in the critical region(s) would be ideal. Such a method would have the potential to significantly lower the computational costs of large simulations and increase the domain size that could be considered.

One particular method that has enjoyed great success is the multi-timestepping method of Prakash and Hjelmstad, first developed in [5] and [4], and extended for nonlinear dynamics by Prakash, Taciroglu, and Hjelmstad in [6]. This method retains an accurate description of material behavior while enabling the use of spatially varying time steps. The multi-timestepping method improves the CG method of Gravouil and Combescure [3], which is stable but dissipative. Unlike other similar methods, this method ensures both the accuracy and stability of the solution and is computationally efficient. The multi-timestepping method uses a Lagrange multiplier approach to couple the different subdomains across a non-overlapping interface and ensure continuity of the solutions across the subdomain boundaries. For this method, a combination of the subdomain equations of motion, Newmark-beta relations, and linear interpolation are used to calculate the kinematic variables. Lagrange multipliers, represented by nodal forces, are calculated using a combination of momentum and velocity conservation equations across the interface, the latter being chosen for stability.

This paper explores the practicality of applying an adapted version of the multi-timestepping method discussed above to peridynamics and the benefits of this adapted approach. Peridynamics is a nonlocal model, so several aspects of the existing multi-time-step method must necessarily be modified, but the basic theory is largely identical to the local case. The concept of an interface between subdomains is generalized to a volumetric interface region, with Lagrange multipliers acting over this region as before. The choice of an overlapping volumetric region for the interface was inspired by the approach used by Burak and Parks in [1] for domain decomposition of peridynamics. The kinematic variables are calculated similarly to the local method. The Lagrange multipliers, now represented by body forces, are calculated again by conservation of momentum and velocity across the volumetric interface region.

**3. Peridynamics Background and Theory.** As mentioned before, local solid mechanics alone is insufficient to capture many types of physical behavior. Classical-based models that are able to capture this behavior typically require some additional

equations to augment the basic equations. Two of the most commonly used methods for modeling fracture are the XFEM method by Belytschko and co-workers [2], and the associated GFEM method by Strouboulis et al. [10]. Both of these methods add enrichment functions to the standard finite element approximations to capture the behavior of cracks and discontinuities.

Peridynamics is a nonlocal analog of solid mechanics that was originally developed to address these concerns by reformulating the partial differential equations of solid mechanics with integro-differential equations. Because discontinuous displacements offer no added complexity for peridynamics, discontinuities can easily be modeled at realistic length scales. Physically, peridynamics can be represented by material points which interact with each other through nonlocal forces. The basic theory of peridynamics was first developed by Silling in [8], and later expanded by Silling et al. in [9] and [7].

In the general peridynamic approach, the equation of motion is given by

$$\rho(\mathbf{x})\ddot{\mathbf{u}}(\mathbf{x},t) = \int_{\mathcal{H}_\mathbf{x}} \{\underline{\mathbf{T}}[\mathbf{x},t]\langle\mathbf{x}'-\mathbf{x}\rangle - \underline{\mathbf{T}}[\mathbf{x}',t]\langle\mathbf{x}-\mathbf{x}'\rangle\}\, dV_{\mathbf{x}'} + \mathbf{b}(\mathbf{x},t). \tag{3.1}$$

Above, $\mathbf{x}$ represents the position of a particle and $\mathbf{u}$ is a displacement vector field. Additionally, $\mathbf{b}$ is an external body force density field and $\rho$ is the material density. $\underline{\mathbf{T}}$ is defined as the *force vector state*. It is a generalization of a second-order tensor such that $\underline{\mathbf{T}}[\mathbf{x},t]\langle\mathbf{x}'-\mathbf{x}\rangle$ maps the vector $\mathbf{x}'-\mathbf{x}$ to the force vector state field. $\mathcal{H}_\mathbf{x}$ is a neighborhood of $\mathbf{x}$ containing all of the points $\mathbf{x}'$ that $\mathbf{x}$ interacts with. In general, $\mathcal{H}_\mathbf{x}$ can be thought of as a sphere surrounding $\mathbf{x}$ of some radius $\delta$, called the horizon. In practice, a good choice for the horizon is usually 3. A more detailed description of the peridynamic model can be found in the Silling et al. paper [9].

One type of material which yields a useful simplification of the peridynamic equation of motion is that of so called *ordinary* materials. Ordinary materials are those which have the property that $\underline{\mathbf{T}} = \underline{t}\underline{\mathbf{M}}$, where $\underline{t}$ is a scalar state and $\underline{\mathbf{M}}$ is the *deformed direction vector state* such that $\underline{\mathbf{M}}\langle\mathbf{x}'-\mathbf{x}\rangle$ is a unit vector pointing from deformed $\mathbf{x}$ to deformed $\mathbf{x}'$. Ordinary materials are useful because it is now possible to define the equation of motion using a two-particle force function $\mathbf{f}$, defined by

$$\underline{t}[\mathbf{x},t]\langle\mathbf{x}'-\mathbf{x}\rangle = \frac{1}{2} f(\mathbf{u}'-\mathbf{u},\mathbf{x}'-\mathbf{x}). \tag{3.2}$$

This allows us to reformulate the equation of motion for ordinary materials as

$$\rho(\mathbf{x})\ddot{\mathbf{u}}(\mathbf{x},t) = \int_{\mathcal{H}_\mathbf{x}} \mathbf{f}(\mathbf{u}(\mathbf{x}',t)-\mathbf{u}(\mathbf{x},t),\mathbf{x}'-\mathbf{x})\, dV_{\mathbf{x}'} + \mathbf{b}(\mathbf{x},t). \tag{3.3}$$

Forces are applied over a volumetric region with a width proportional to the horizon. This is based on the constraints on boundary conditions established in previous research on peridynamic theory by Silling [8]. These forces can be constant or time-variant, and are representative of the body forces on the material which are seen in the equation of motion. Damage is calculated by comparing the current stretch, given by the ratio of the relative deformation between two nodes to their undeformed relative position, to some critical stretch $s_0$. The stretch must be checked for every bond formed with every node. If any bond stretches beyond the critical limit, that bond is broken and is taken as zero in future calculations.

For ordinary materials, the equation of motion can be stated more simply in a discretized form for each node $i$ as

$$\rho\mathbf{a}^i_{n+1} = \mathbf{L}^i_{n+1} + \mathbf{b}^i_{n+1}, \tag{3.4}$$

where $\mathbf{L}_{n+1}^i$ is a discretized form of the force integration given above. For homogeneous bodies the discretized form can be represented by a summation over all other nodes $p$ the current node $i$ interacts with as follows:

$$\mathbf{L}_{n+1}^i = \sum_p \mathbf{C}(\mathbf{x}^p - \mathbf{x}^i)(\mathbf{u}_n^p - \mathbf{u}_n^i)V^p, \tag{3.5}$$

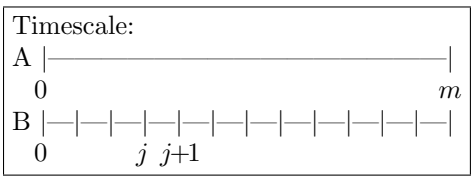
where $\mathbf{C}(\mathbf{x}^p - \mathbf{x}^i)$ is the micromodulus function of the material.

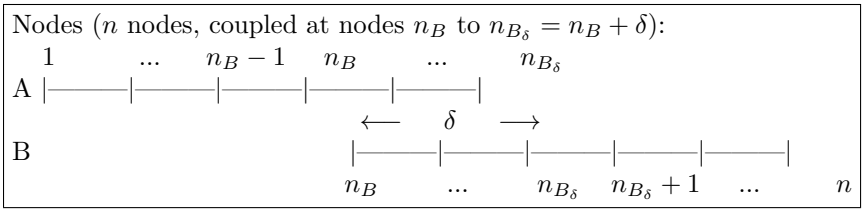
When solving discrete peridynamic systems as described above, a time-stepping scheme must also be used to advance the solution through time. For this application, the Newmark-Beta scheme was used.

**4. Basic Theory of Multi-Timestepping Applied to Peridynamics.** With the peridynamic approach, each discretized volume section is represented by a node in a peridynamic grid. We can further divide the grid into subdomains which will be integrated at different timesteps based on the accuracy and stability requirements for each specific region. In many cases, the accuracy and stability requirements in a relatively small region of the domain dictates the timestep for the entire domain. One common example of this is in the region surrounding a crack tip. Often, we have neither the need nor the resources to solve the entire domain at the level of fidelity that is required in that small region. When this occurs, the multi-timestepping method enables us to integrate only the small region at the more restrictive timestep. The rest of the domain is integrated at whatever time step is normally required, and the domains are coupled together so that they will accurately model the entirety of the domain.

For simplicity of derivation, we assume two subdomains, which we label subdomain A and subdomain B. However, it is useful to note that the multi-timestepping method is trivially applicable to a general number of subdomains using a subdomain tree approach. For a two domain case, subdomain B will refer to the collection of nodes at the smaller timestep, while subdomain A represents the remainder of the domain.

For multiple subdomains integrated at different time steps, there must be a volumetric interface region where the nodes of the subdomains overlap of width at least equal to the horizon $\delta$, similar to applied forces and boundary conditions. The following two graphics provide a visual representation of the time steps of two general subdomains and also an example of how a simple one dimensional, nonlocal spring-mass system can be divided into two subdomains.






Nodes in subdomain A can simply be integrated at a timestep chosen to adhere to the accuracy and stability requirements for that region, which we label $\Delta t_A$ Nodes in subdomain B are in a region that requires integration at some smaller timestep $\Delta t_B$. This means we need to calculate the kinematic values for every node at each time $t_j,\ j = 1, 2, \ldots, m$, where m=$\Delta t_A/\Delta t_B$. Conversely, for the nodes in subdomain A it becomes evident that we only need to calculate kinematic values for most nodes at the common time $t_m$.

Note that in the interface region (nodes $n_B - n_{B_\delta}$ above), each node is in both subdomains A and B. For each node in the interface region a Lagrange multiplier $\lambda$ must included in the equations of motion of each subdomain to couple the subdomains together. In this sense the Lagrange multipliers can be interpreted as interface reaction forces acting internally between the subdomains [5, 4].

The equations of motion for each node in the interface region thus become

$$\begin{aligned} \rho a^i_{A,m} &= L^i_{A,m} + b^i_{A,m} + \lambda^i_m \\ \rho a^i_{B,j} &= L^i_{B,j} + b^i_{B,j} - \lambda^i_j,\ \forall j = 1, 2, ..., m. \end{aligned} \tag{4.1}$$

Just as before, the Newmark algorithm is used to advance the solution forward in time. There are multiple subdomains advancing at various time steps with possibly different Newmark-Beta coefficients, which leads to a unique set of equations for each subdomain. The Newmark-Beta equations at each node $i$ for subdomains A and B as defined above are

$$\begin{aligned} v^i_{A,m} &= v^i_{A,0} + \Delta t_A[(1-\gamma_A)a^i_{A,0} + \gamma_A a^i_{A,m}] \\ d^i_{A,m} &= d^i_{A,0} + \Delta t_A v^i_{A,0} + \Delta t^2_A[(1/2-\beta_A)a^i_{A,0} + \beta_A a^i_{A,m}] \\ v^i_{B,j} &= v^i_{B,j-1} + \Delta t_B[(1-\gamma_B)a^i_{B,j-1} + \gamma_B a^i_{B,j}],\ \forall j = 1, 2, ..., m \\ d^i_{B,j} &= d^i_{B,j-1} + \Delta t_B v^i_{B,j-1} + \Delta t^2_B[(1/2-\beta_B)a^i_{B,j-1} + \beta_B a^i_{B,j}],\ \forall j = 1, 2, ..., m. \end{aligned} \tag{4.2}$$

The equations of motion and the Newmark equations correspond to the kinematic quantities of subdomain $B$ at all time steps $j = 1, 2, ..., m$, and of subdomain $A$ at the final step $m$. However, the nodes in subdomain A that nodes in subdomain B can "see" (those within its horizon) must also be defined at the finer-grained time step. This is necessary to satisfy the equation of motion for the nodes in subdomain B that reference them. To determine the kinematic values of those nodes in subdomain $A$ at the intermediate time steps $j = 1, 2, ..., m-1$, linear interpolation is used. For each corresponding node $i$ of subdomain A we have

$$\begin{aligned} d^i_{A,j} &= (1 - j/m) \cdot d^i_{A,0} + j/m \cdot d^i_{A,m} \\ v^i_{A,j} &= (1 - j/m) \cdot v^i_{A,0} + j/m \cdot v^i_{A,m} \\ a^i_{A,j} &= (1 - j/m) \cdot a^i_{A,0} + j/m \cdot a^i_{A,m}. \end{aligned} \tag{4.3}$$

To balance the interface reaction force at the common time step, $\lambda_m^i$, the continuity of a kinematic quantity must be enforced on the interface boundary. Previous research on the classical multi-timestepping method has shown that the continuity of velocity is the only choice that produces unconditionally stable results, see [3]. Numerical results of multi-timestepping applied to peridynamics also support this result, and continuity of velocity is the constraint that we elect to use as well. This leads to an additional set of velocity conservation equations for each interface node $i$,

$$v_{A,m}^i = v_{B,m}^i. \tag{4.4}$$

Additionally, to balance the interface reaction forces $\lambda_j^i$ at the intermediate time steps $j = 1, 2, ..., m-1$, we use the following conservation of momentum equations. For each interface node $i$ in subdomain A we have

$$\rho a_{A,j}^i = L_{A,j}^i + b_{A,j}^i + \lambda_j^i \ \forall j = 1, 2, ..., m-1. \tag{4.5}$$

For the equations of motion on the interface, a fraction of the stiffness and mass is distributed to each subdomain. This can be either a pre-determined value or calculated at runtime based on the properties of the system. Likewise, we distribute a fraction of the force acting on interface nodes to each subdomain. When considering nodes near the edge of the peridynamic region and nodes close to the horizon of other nodes, we enforce the restriction that only the portion of the nodal volume that the current node can "see" will contribute to the pairwise force function. To obtain the amount of the nodal volume that will contribute to the pairwise force, a simple approximation was made by taking the effective nodal volume to be the ratio of the number of corners of the cubic nodal volume our current node can see over the total number of corners (8 for cubic volumes). For one-dimensional problems, this simplifies to a factor of $1/2$ or 1.

To solve the coupled system, the full set of equations given above can be arranged into a linear system and solved using standard methods. Alternatively, the equations can be grouped in such a way that the equations corresponding to each subdomain can be decoupled and solved separately using a bordered solution procedure. The actual solution is then recovered through an update using the interface reaction forces. This procedure, which allows for a much more efficient solve, is discussed in detail in [4], and will not be covered here.

The results of this research correspond to a model representing the fracture of a glass plate. To capture this physical phenomenon, an initial crack was included in

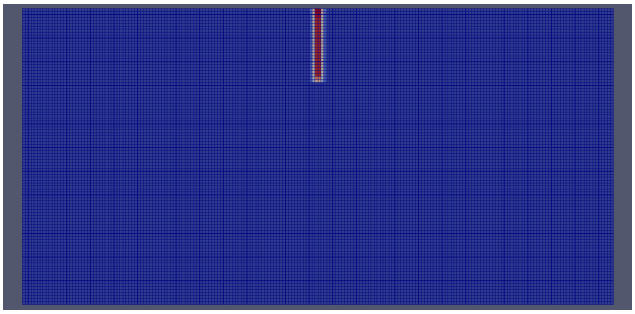


Fig. 4.1: Cracked Plate

the model which cuts through the depth of the slide. This crack was idealized as a pre-notch which was inserted in the model at the upper midpoint of the slide. For implementation, any bonds that would normally pass through this pre-crack are taken to be initially broken.

**4.1. Cost Analysis.** For any interior node, there are a given number of initially unbroken bonds connecting it to the surrounding nodes. This number is based on the horizon of the system and can be used to roughly approximate the cost of a simulation. For example, if the horizon is $\delta = 3$, there will be exactly 122 bonds connecting to an interior node. If $\delta = 4$, that number grows to 256, while for $\delta = 2$, there are only 32 bonds.

We can use this value to visualize the computational gain of utilizing the multi-time-step method with peridynamics. If we label the number of bonds at each interior node as $n_b$ and estimate costs for all nodes using this value, we can obtain a relatively simple value for comparison of the original system and the split system by approximating the total number of bonds to be evaluated in each. If we denote $n_n$ to be the number of nodes in the system, and $m = \Delta t_A/\Delta t_B$, the timestep ratio as before, the original peridynamic system will have approximately $n_b * n_n * m$ bonds to evaluate in one large timestep $\Delta t_A$. For the split system, we further define the number of nodes exclusively in each subdomain and in the overlap between the subdomains as $n_n^A$, $n_n^B$, and $n_n^O$ respectively, keeping in mind that $n_n^A + n_n^B + n_n^O = n_n$. Therefore, the approximate number of bonds in the split system we must evaluate in each timestep is $n_b * (n_n^A + (n_n^B + 2n_n^O) * m)$ This indicates that the splitting a peridynamic domain into two subdomains as outlined in this paper will start to be cost-effective approximately when the following condition is true: $n_n * m > n_n^A + (n_n^B + 2n_n^O) * m$, or more simply stated as $n_n^A(m-1) > 2m \cdot n_n^O$. It is also useful to note that the relative cost-savings between the multi-timestepping approach applied to classical mechanics and nonlocal peridynamics should be approximately equal. This is because an equivalent cost analysis for the classical method would yield the same results as peridynamics with a horizon of one.

**5. Numerical Results.** The following plot shows the final result from the simulation of fracture in a plate of Duran 50 glass of size $20 \times 10 \times .1$ mm and with discretized element size of 0.1 mm. Results were obtained for both peridynamics alone and also from the multi-timestepping peridynamic approach. For the multi-timestepping case, one subdomain was taken to include 20% of the total nodes and was located at the middle section of the plate (the region containing the crack). The remainder of the domain was not in a region requiring as high of fidelity as the region surrounding the crack and thus was evaluated with a timestep twice as large as the timestep in the middle region.

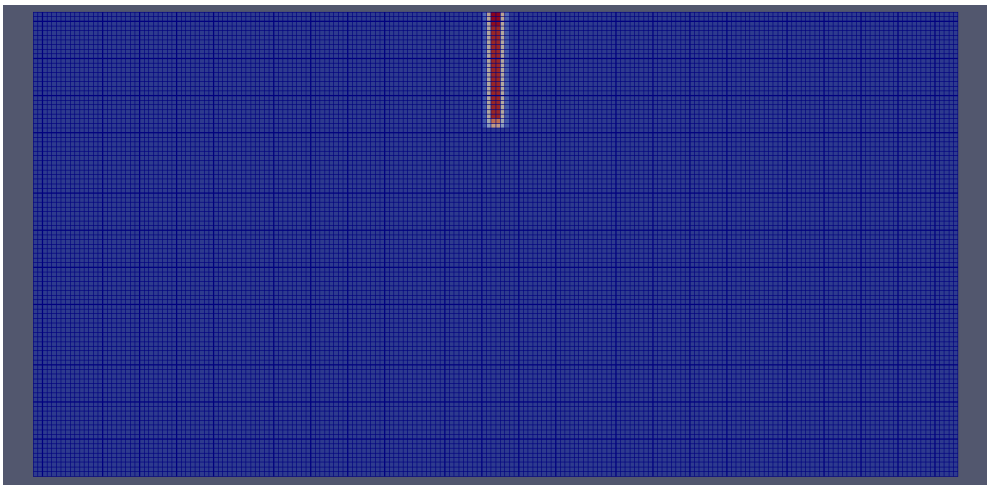
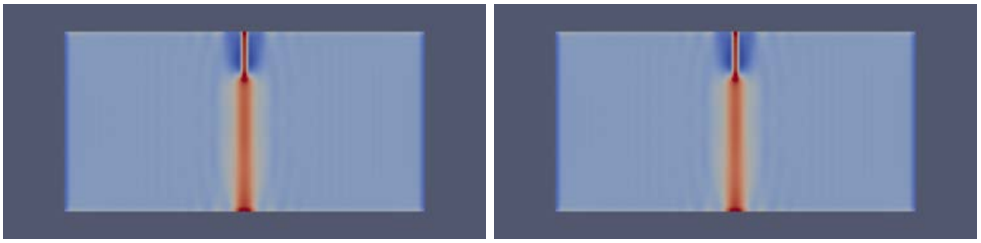


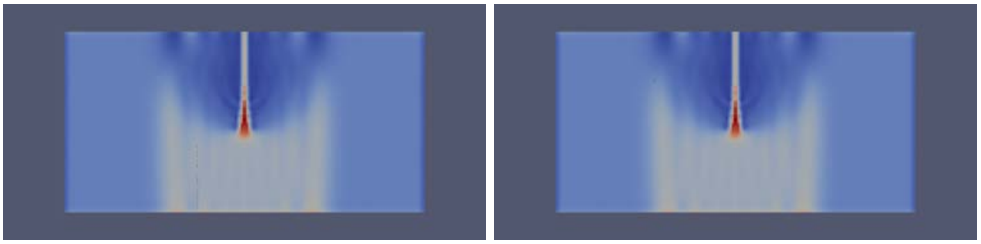
Fig. 5.1: Cracked Plate

Three results showing the displacement and damage in the nodes at a specific instant in time are shown below. The first is when the crack first starts propagating, the second is when branching behavior is first seen, and the third when the plate is almost completely cracked.



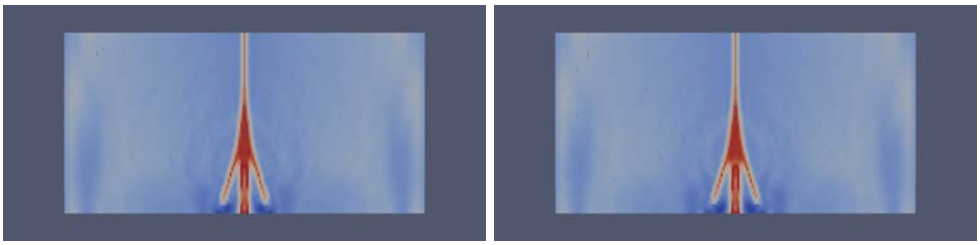
Original System Split System

Fig. 5.2: Plate when the crack first starts propagating



Original System Split System

Fig. 5.3: Plate when branching behavior is first seen



Original System Split System

Fig. 5.4: Plate when almost completely cracked

These results again show that the split system results match very well with the original system, and further the conclusion that the multi-time-stepping method applied to peridynamics is a valid solution approach.

Time-history results for a few selected nodes from this simulation are also included below. The nodes that were chosen correspond to the left edge of the plate, the middle of the plate, and an interface node. The results at interface nodes for subdomains A and B are nearly identical, so only subdomain A results are shown.

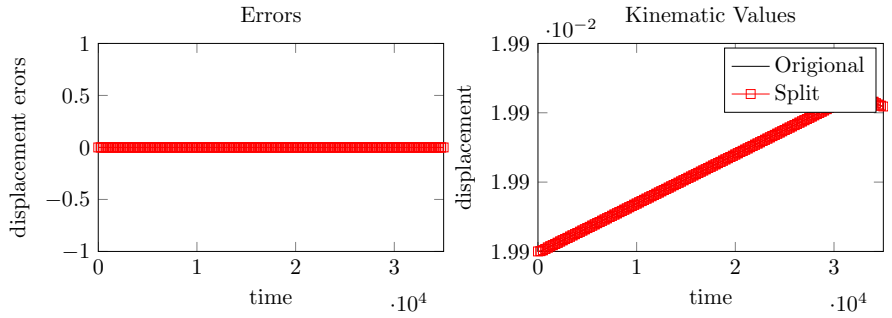



Fig. 5.5: Kinematic Values and Errors for Subdomain A, position (0,2.5e-03,0)

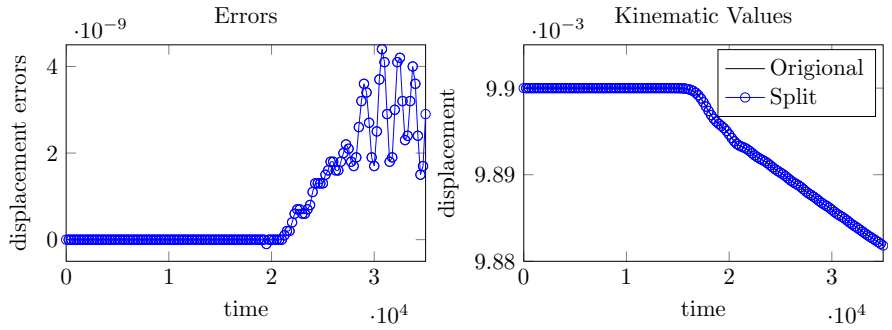



Fig. 5.6: Kinematic Values and Errors for Subdomain B, position (1e-02,9.9e-03,0)

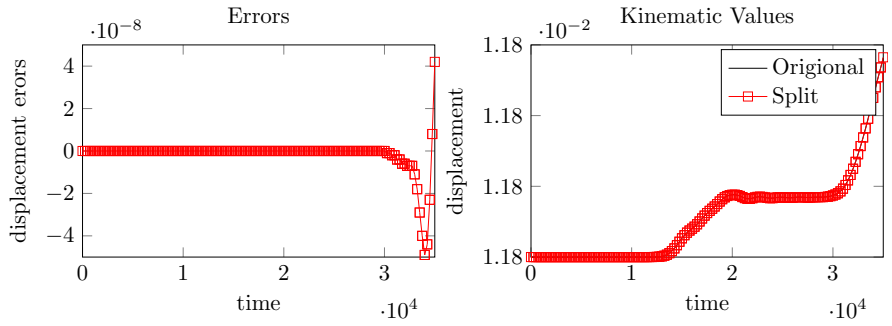



Fig. 5.7: Kinematic Values and Errors for Subdomain A, position (1.19e-02,0,0)

In the kinematic plots of figures (5.5-5.7), red lines represent data corresponding to subdomain A, blue lines to subdomain B, and black lines to the original system. In the error plots, red and blue lines correspond to the numerical errors for subdomains A and B respectively. These figures serve to show that the numerical and analytical errors track very well, while giving kinematic plots for a sense of scale. This, coupled with the significant cost advantage the multi-timestepping method can offer, gives a compelling motivation for the use of multi-timestepping in peridynamic models.

For a large system, the benefits of using multi-timestepping vs the undecomposed system become immediately clear. For a system 100 nodes wide and a timestep ratio of 2, with subdomain B being 10% of the total width, the simple error analysis shown earlier indicates that the split system will be over 60% faster than the original system. For 1000 nodes that number grows to 80% or much higher for smaller widths of subdomain B.

**6. Conclusions.** From the results of the simple 1D case to the full simulation of crack growth in a glass plate the advantages of multi-timestepping in peridynamic simulations is evident. When a problem involves obtaining accurate solutions to complex domains with widely varying time scales, it might well benefit from this type of approach. Not only does this approach gain the inherent benefits of domain decomposition by the splitting of a problem domain into subdomains, it also enables a completely novel development for peridynamic theory: the ability to relax temporal restrictions as needed. Additionally, error results not included in this paper seem to indicate that with a proper choice of integration parameters, multi-timestepping applied to peridynamics enjoys the same rate of convergence as the underlying Newmark method. The retention of the convergence of the underlying integration method is also seen in multi-timestepping applied to classical problems and is another attractive feature of this method. The combination of the ability to use restrictive timesteps only where it is needed most and the retention of convergence makes this approach very desirable in both cost and accuracy.

REFERENCES

[1] B. Aksoylu and M. L. Parks, *Variational theory and domain decomposition for nonlocal problems*, Applied Mathematics and Computation, 217 (2011), pp. 6498–6515.

[2] T. Belytschko and T. Black, *Elastic Crack Growth in Finite Elements with Minimal Remeshing*, International Journal for Numerical Methods in Engineering, 45 (1999),

pp. 601–620.

[3] A. Gravouil and A. Combescure, *Multi-time-step explicit–implicit method for non-linear structural dynamics*, International Journal for Numerical Methods in Engineering, 50 (2001), pp. 199–225.

[4] A. Prakash, *Multi-time-step Domain Decomposition and Coupling Methods for Non-linear Structural Dynamics*, ProQuest, 2007.

[5] A. Prakash and K. Hjelmstad, *A FETI-based multi-time-step coupling method for Newmark schemes in structural dynamics*, International Journal for Numerical Methods in Engineering, 61 (2004), pp. 2183–2204.

[6] A. Prakash, E. Taciroglu, and K. D. Hjelmstad, *Computationally efficient multi-time-step method for partitioned time integration of highly nonlinear structural dynamics*, Computers & Structures, 133 (2014), pp. 51–63.

[7] S. Silling and R. Lehoucq, *Peridynamic Theory of Solid Mechanics*, Advances in Applied Mechanics, 44 (2010), pp. 73–166.

[8] S. A. Silling, *Reformulation of elasticity theory for discontinuities and long-range forces*, Journal of the Mechanics and Physics of Solids, 48 (2000), pp. 175–209.

[9] S. A. Silling, M. Epton, O. Weckner, J. Xu, and E. Askari, *Peridynamic States and Constitutive Modeling*, Journal of Elasticity, 88 (2007), pp. 151–184.

[10] T. Strouboulis, I. Babuška, and K. Copps, *The design and analysis of the Generalized Finite Element Method*, Computer Methods in Applied Mechanics and Engineering, 181 (2000), pp. 43–69.

# GALERKIN RBF METHOD:A MESHFREE METHOD FOR THE DISCRETIZATION OF NONLOCAL DIFFUSION EQUATIONS

S.T. ROWE * AND R.B. LEHOUCQ †

**Abstract.** We discuss the computational aspects of a discretization for a nonlocal diffusion problem using a localized basis of radial basis functions. The stiffness matrix entries are assembled by a special quadrature routine unique to the localized basis. Combining the quadrature method with the localized basis produces a well-conditioned, sparse, symmetric positive definite stiffness matrix. We present convergence results for the method and discuss accelerating parts of the method with graphics processing units using CUDA and Kokkos.

**1. Introduction.** The purpose of this paper is to study the numerical properties of a meshfree discretization method for nonlocal diffusion equations. Nonlocal diffusion models are being explored due to the difficulty classical diffusion models have modeling anomalous diffusion, e.g., within heterogeneous material; see [2] a discussion.

We apply radial basis functions to develop a meshfree method for the discretization of a variational formulation of the steady state nonlocal diffusion model. Radial basis function methods use shifts and translates of a single univariate function to approximate and interpolate scattered data. Radial basis functions have found use in interpolation of scattered data, discretization methods for partial differential equations, and for surface reconstruction [6]. The discretization method we propose uses a novel, localized basis of radial basis functions rather than directly using the shifts and translates of a single univariate function. A quadrature method unique to the localized basis is constructed which provides fast assembly of a sparse stiffness matrix for the discretization of the nonlocal diffusion problem. The resulting stiffness matrix is well-conditioned and symmetric positive definite, and the linear system may be solved via conjugate gradient.

Previous work has explored the use of radial basis functions for the discretization of nonlocal diffusion problems [1]. We extend the previous work by introducing a method that offers the same benefits of the radial basis function method introduced in [1] while decreasing the computational difficulty associated with the previous method. The previous method required the solution of large, dense linear systems before the stiffness matrix for the nonlocal diffusion problem could even be assembled. The assembly of the quadrature weights, a necessary ingredient for the method, required the solution of another large, dense linear system. The method we propose and study similarly uses a localized basis without the need for the solution of any large, dense linear systems. Instead, small linear systems are solved for the construction of the localized basis we discuss in 3.1. The quadrature weights are assembled by directly integrating the basis functions. The integration of these basis functions is efficiently computed by exploiting parallelism. In 5.5, we discuss the assembly of the quadrature weights in more detail. In addition to the potential computational advantages of the method we propose, we extend the problems considered in [1] by exploring anisotropic nonlocal diffusion.

We study the acceleration of radial basis function methods by the use of graphics processing units (GPUs). Identifying and exploiting concurrency in a numerical method can potentially provide substntial acceleration to the method. We observe examples of concurrency within methods involving radial basis functions and consider

*Texas A&M University, srowe@math.tamu.edu
†Sandia National Laboratories, rblehou@sandia.gv

a parallel assembly of quadrature weights used in the RBF method used for the solution of the nonlocal diffusion equation. The methods we consider exhibit the parallel pattern of requiring execution of the same instruction on multiple data. Furthermore, a limited amount of memory transfers are required, which suggests the methods may benefit from acceleration of graphics processing units. We employ NVidia's Compute Unified Device Architecture (CUDA) to exploit massive parallelism for the speedup of computations involving radial basis functions. 5.5 discusses CUDA, GPUs, and the Trilinos package Kokkos in more detail.

**2. Nonlocal Diffusion.** In this section, we present the nonlocal diffusion operator and we introduce the problem we aim to solve. Let $\Omega \subset \mathbb{R}^n$ be an open subset and let $\gamma : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ be a kernel such that for each $x \in \Omega$, $\gamma(\boldsymbol{x}, \boldsymbol{y}) = 0$ for $\|\boldsymbol{x} - \boldsymbol{y}\| > \epsilon$, where $\epsilon$ is a positive number we refer to as the *horizon*. The *interaction domain* $\Omega_{\mathcal{I}}$ is defined as

$$\Omega_{\mathcal{I}} := \{\boldsymbol{y} \in \mathbb{R}^n \backslash \Omega : \gamma(\boldsymbol{x}, \boldsymbol{y}) \neq 0\} \tag{2.1}$$

We define the nonlocal diffusion operator $\mathcal{L}$ as

$$\mathcal{L}u(\boldsymbol{x}) := \int_{\Omega \cup \Omega_{\mathcal{I}}} \big(u(\boldsymbol{x}) - u(\boldsymbol{y})\big)\gamma(\boldsymbol{x}, \boldsymbol{y})\, d\boldsymbol{y} \tag{2.2}$$

Given $f : \Omega \to \mathbb{R}$ and $g : \Omega_{\mathcal{I}} \to \mathbb{R}$ the nonlocal diffusion problem seeks $u : \Omega \cup \Omega_{\mathcal{I}} \to \mathbb{R}$ such that

$$\begin{aligned} \mathcal{L}u(\boldsymbol{x}) &= f(\boldsymbol{x}) & \boldsymbol{x} &\in \Omega \\ u(\boldsymbol{x}) &= 0 & \boldsymbol{x} &\in \Omega_{\mathcal{I}} \end{aligned} \tag{2.3}$$

where $u : \Omega \cup \Omega_{\mathcal{I}} \to \mathbb{R}$.

It can be shown that the solution $u$ to (2.3) formally solves a variational problem. Let $a$ denote the bilinear form

$$a(u, v) := \int_{\Omega \cup \Omega_{\mathcal{I}}} \int_{\Omega \cup \Omega_{\mathcal{I}}} \big(u(\boldsymbol{x}) - u(\boldsymbol{y})\big)\big(v(\boldsymbol{x}) - v(\boldsymbol{y})\big)\gamma(\boldsymbol{x}, \boldsymbol{y})\, d\boldsymbol{y} d\boldsymbol{x}. \tag{2.4}$$

We define the *constrained energy space* $L^2_c(\Omega \cup \Omega_{\mathcal{I}})$ by

$$L^2_c(\Omega \cup \Omega_{\mathcal{I}}) := \{v \in L^2(\Omega \cup \Omega_{\mathcal{I}}) : v = 0 \text{ a.e. in } \Omega_{\mathcal{I}}\}.$$

The variational problem we solve is to find $u \in L^2_c(\Omega \cup \Omega_{\mathcal{I}})$ such that for all $v \in L^2_c(\Omega \cup \Omega_{\mathcal{I}})$,

$$a(u, v) = \int_{\Omega} f(\boldsymbol{x}) v(\boldsymbol{x})\, d\boldsymbol{x}. \tag{2.5}$$

The paper [2] provides sufficient conditions on the kernel $\gamma$ that the bilinear form (2.4) is coercive and bounded on the space $L^2_c(\Omega \cup \Omega_{\mathcal{I}})$. By the Lax-Milgram theorem, there exists a unique solution to (2.5). Our objective in this paper is to study a discretization of (2.5).

We assume that the kernel $\gamma$ is of the form

$$\gamma(\boldsymbol{x}, \boldsymbol{y}) = \big(\kappa(\boldsymbol{x}) + \kappa(\boldsymbol{y})\big)\Phi(\|\boldsymbol{x} - \boldsymbol{y}\|)$$

where $\Phi : \mathbb{R} \to \mathbb{R}$ is assumed to be a compactly supported function with horizon $\epsilon$ and $\kappa : \mathbb{R}^n \to \mathbb{R}$ is a smooth function. The spatial variation in the kernel $\gamma$ is generated by what we refer to as the *anisotropy function* $\kappa$. By properly rescaling the operator $\mathcal{L}$ with respect to the horizon $\epsilon$, the nonlocal operator converges to a classical differential operator; see [2].

**3. Radial Basis Functions.** We present necessary background on radial basis functions, introduce the localized basis we use for our proposed discretization, and discuss the quadrature weights we employ for our method. Let $\Omega \subset \mathbb{R}^n$ and let $\varphi : [0, \infty) \to \mathbb{R}$ be a continuous function. Let $X \subset \Omega$ be a collection of scattered points, referred to as *centers*. A set of *radial basis functions* is defined to be the collection $\{\varphi(\|\boldsymbol{x} - \boldsymbol{x}_j\|) : \boldsymbol{x}_j \in X\}$. Given a continuous function $f : \Omega \to \mathbb{R}$, an interpolant $I_X f = \sum_{i=1}^N c_i \varphi(\|\boldsymbol{x} - \boldsymbol{x}_i\|)$ is constructed by enforcing the interpolation conditions

$$f(\boldsymbol{x}_j) = \sum_{i=1}^{N} c_i \varphi(\|\boldsymbol{x} - \boldsymbol{x}_i\|) \qquad j = 1, \ldots, N,$$

provided that their exists coefficients $c_i$ which satisfy the interpolation conditions.

Let $\pi_m$ denote the space of degree $m-1$ polynomials in $\mathbb{R}^n$ and let $\{p_l\}_{l=1}^{m_n}$ be a basis for $\pi_m$. We say that a function is *conditionally positive definite* of order $m$ if for any collection of scattered centers $X$

$$\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j \varphi(\|\boldsymbol{x}_i - \boldsymbol{x}_j\|) > 0$$

for any $\boldsymbol{\alpha} \in \mathbb{R}^n$ that satisfies

$$\sum_{i=1}^{n} \alpha_i p_l(\boldsymbol{x}_i) = 0 \qquad l = 1, \ldots, m_n.$$

We focus on the *thin plate spline* (or *surface spline*)

$$\varphi(r) = \begin{cases} r^{2m-n} & n \text{ is odd} \\ r^{2m-n} \log(r) & n \text{ is even} \end{cases}$$

which is conditionally positive definite of order $m$ on $\mathbb{R}^n$. The interpolation conditions for conditionally positive functions of order $m$ requires the addition of a degree $m-1$ polynomial to the linear combination of radial basis functions. Given centers $\{\boldsymbol{x}_i\}_{i=1}^N$ and data $\{f(\boldsymbol{x}_i)\}\_i = 1^N$, the interpolant

$$I_X f(\boldsymbol{x}) = \sum_{i=1}^{N} c_i \varphi(\|\boldsymbol{x} - \boldsymbol{x}_i\|) + \sum_{l=1}^{m_n} d_l p_l(\boldsymbol{x})$$

is constructed by enforcing the conditions

$$\begin{aligned} f(\boldsymbol{x}_j) &= \sum_{i=1}^{N} c_i \varphi(\|\boldsymbol{x}_i - \boldsymbol{x}_j\|) + \sum_{l=1}^{m_n} d_l p_l(\boldsymbol{x}_j) \qquad \text{for } j = 1, \ldots, N, \\ 0 &= \sum_{i=1}^{N} c_i p_l(\boldsymbol{x}_i) \qquad \text{for } l = 1, \ldots, m_n \end{aligned} \tag{3.1}$$

The geometry of the centers $X$ controls the approximation quality of the interpolant $I_X f$. The *mesh norm* $h$ is defined to be the radius of the largest ball in $\Omega$ which does not contain any centers $X$. The separation radius $q$ is defined to be the

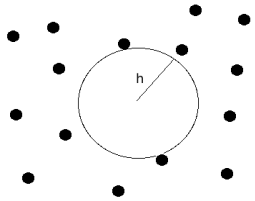



Fig. 3.1: A ball of radius $h$ is illustrated. The ball does not contain any of the centers.

minimal distance between any two centers. 3.1 illustrates an example of a mesh norm for a collection of scattered points. The formulae for the mesh norm $h$ and separation radius $q$ are given, respectively, by

$$h := \sup_{\boldsymbol{x}\in\Omega} \min_{\boldsymbol{x}_j\in X} \|\boldsymbol{x}-\boldsymbol{x}_j\| \qquad q := \min_{\boldsymbol{x}_j,\boldsymbol{x}_i\in X} \|\boldsymbol{x}_j-\boldsymbol{x}_j\|. \tag{3.2}$$

**3.1. Lagrange Functions and Local Lagrange Functions.** Given a collection of scattered centers $X \subset \Omega \subset \mathbb{R}^n$ and a conditionally positive definite function $\varphi$ of order $m$, there exists a collection of unique functions $\{\chi_i\}_{i=1}^N$ that satisfy $\chi_i(\boldsymbol{x}_j) = \delta_{i,j}$ for each center $\boldsymbol{x}_j \in X$. The condition $\delta_{i,j} = \chi_i(\boldsymbol{x}_j)$ is enforced by employing the conditions in (3.1). We call $\chi_i$ the *Lagrange function* centered at $\boldsymbol{x}_i$. The construction of $\chi_i$ necessitates the solution of an $(N + m_n) \times (N + m_n)$ size dense, linear system. The collection of Lagrange functions $\{\chi_i\}_{i=1}^N$ is referred to as the Lagrange basis and interpolation of data samples $\{f(\boldsymbol{x}_i)\}_{i=1}^N$ is accomplished by

$$I_X f(\boldsymbol{x}) = \sum_{i=1}^N f(\boldsymbol{x}_i)\chi_i(\boldsymbol{x}).$$

The *local Lagrange function* $\hat{\chi}_i$ centered at $\boldsymbol{x}_i$ is a function designed to approximate $\chi_i$ by using only centers nearby $\boldsymbol{x}_i$. Let $r > 0$ be a fixed number and let $\Upsilon_i = \{\boldsymbol{y} \in X : \|\boldsymbol{y} - \boldsymbol{x}_i\| < r\}$. We construct $\hat{\chi}_i$ by enforcing the interpolation conditions (3.1) by $\hat{\chi}_i(\boldsymbol{x}_j) = \delta_{i,j}$ for $\boldsymbol{x}_j \in \Upsilon_i$. A detailed discussion of the construction of local Lagrange functions on domains in $\mathbb{R}^n$ and on manifolds may be found in [5] and [4], respectively. It can be shown that the choice $r = Kh|\log(h)|$ allows for optimal order error between $\chi_i$ and $\hat{\chi}_i$ in the supremum norm.

**3.2. Local Lagrange Quadrature.** We introduce a quadrature method for compactly supported functions in a set $\Omega$ that is imperative for the implementation of the discretization method we propose in 4. Let $X \subset \Omega \subset \mathbb{R}^n$ be a collection of scattered functions and let $\{\hat{\chi}_i\}_{i=1}^N$ be a collection of local Lagrange functions constructed from a conditionally positive definite of order $m$ function $\varphi$. For $f \in W_2^k(\Omega)$ where $\frac{n}{2} < k \leq m$, we define the local Lagrange quadrature rule

$$Q_X f := \sum_{i=1}^N f(\boldsymbol{x}_i)\hat{w}_i \qquad \hat{w}_i := \int_\Omega \hat{\chi}_i(\boldsymbol{x})\, d\boldsymbol{x}.$$

**4. Discretization.** We propose a discretization of (2.5) using local Lagrange functions. Let $X \subset \Omega \cup \Omega_{\mathcal{I}}$ be a collection of centers and let $X_\Omega$. We define the

approximation space $V_h = \text{span}\{\hat{\chi}_i \mathbb{1}_\Omega : x_i \in X_\Omega\}$ where

$$\mathbb{1}_A(\boldsymbol{x}) = \begin{cases} 1 & \boldsymbol{x} \in A \\ 0 & \boldsymbol{x} \notin A. \end{cases}$$

The $\mathbb{1}_\Omega$ is multiplied onto the local Lagrange functions so that $V_h \subset L^2_c(\Omega \cup \Omega_\mathcal{I})$. The local Lagrange functions are not contained within the constrained energy space since they have nonzero values in the interaction domain. We seek $u_h \in V_h$ such that for all $v_h \in V_h$,

$$a(u_h, v_h) = \int_\Omega f(\boldsymbol{x}) v_h(\boldsymbol{x})\, d\boldsymbol{x}. \tag{4.1}$$

Since $V_h \subset L^2_c(\Omega \cup \Omega_\mathcal{I})$, there exists a unique solution to (4.1). The resulting stiffness $\boldsymbol{A}$ and source vector $\boldsymbol{b}$ have entries given by

$$\boldsymbol{A}_{i,j} = a(\hat{\chi}_i, \hat{\chi}_j) \qquad \boldsymbol{b}_i = \int_\Omega f(\boldsymbol{x}) \hat{\chi}_i(\boldsymbol{x})\, d\boldsymbol{x}.$$

We apply the local Lagrange quadrature method of 3.2 to evaluate the entries of the stiffness matrix and the source vector. We arrive at

$$\begin{aligned} \hat{\boldsymbol{A}}_{i,j} &= 2\delta_{i,j}\hat{w}_i \int_{\Omega \cup \Omega_\mathcal{I}} \gamma(\boldsymbol{x}, \boldsymbol{x}_i)\, d\boldsymbol{x} - 2\hat{w}_i \hat{w}_j \gamma(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ \hat{\boldsymbol{b}_i} &= f(\boldsymbol{x}_i)\hat{w}_i. \end{aligned} \tag{4.2}$$

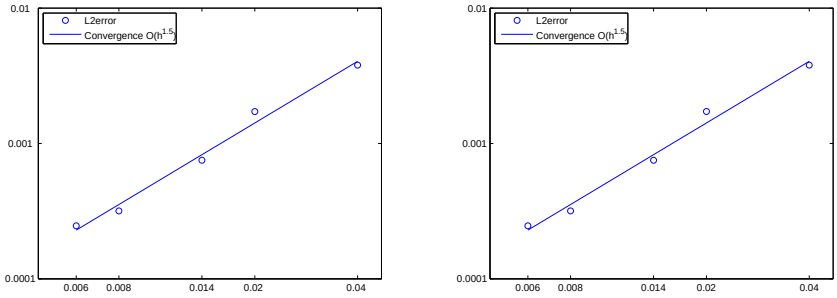
Applying the local Lagrange quadrature rule results in a sparse stiffness matrix. For $\|\boldsymbol{x}_i - \boldsymbol{x}_j\| > \epsilon$, $\hat{\boldsymbol{A}}_{i,j} = 0$ since $\gamma(\boldsymbol{x}_i, \boldsymbol{x}_j) = 0$.

PROPOSITION 4.1. . *The condition number of the discrete stiffness matrix is bounded independent of $h$ or $q$.*

**5. Numerical Results.** We present numerical results for experiments using the discretization described in 4. We discuss local Lagrange function construction, $L^2$ error computations, and condition number computations. We compare the theoretical prediction for $L^2$ convergence and condition numbers with observed results from numerical experiments. We consider solving two dimensional problems of the form (2.4) with two radial kernels $\Phi$ and two anisotropy functions $\kappa$; see 5.1 and 5.2. For all tests we consider zero Dirichlet volume constraints. The tests are computed on the set $\Omega \cup \Omega_\mathcal{I}$ where $\Omega = (0,1) \times (0,1)$ and $\Omega_\mathcal{I} = [-\frac{1}{4}, \frac{5}{4}] \times [-\frac{1}{4}, \frac{5}{4}] \backslash \Omega$. All computations are done in MATLAB and the condition numbers of the sparse stiffness matrices are approximated by the condest function. The sparse linear system is solved with either MATLAB's backslash operator or by conjugate gradient with a specified tolerance of $10^{-9}$. The number of iterations required for convergence for conjugate gradient did not vary as $h$ decreased.

The local Lagrange functions are constructed with linear combinations of the surface spline $\varphi(r) = r^2 \log(r)$. Each local Lagrange function is constructed using approximately $11 \log(N)^2$ nearest neighbor centers, where $N$ is the total number of centers in $\Omega \cup \Omega_\mathcal{I}$. The stiffness matrix for the nonlocal problem only requires Lagrange functions centered in $\Omega$, although thin plate splines centered in $\Omega_\mathcal{I}$ are required for the construction of the local Lagrange functions.

For numerically solving (4.1), a kernel $\gamma(\boldsymbol{x}, \boldsymbol{y}) = \big(\kappa(\boldsymbol{x}) + \kappa(\boldsymbol{y})\big)\Phi(\|\boldsymbol{x} - \boldsymbol{y}\|)$ is chosen with fixed horizon $\epsilon$ and a solution $u \in L^2_c(\Omega \cup \Omega_\mathcal{I})$ is chosen for each numerical




(a) Uniformly Spaced Centers (b) Scattered Centers

Fig. 5.1: The log of $h$ versus the log of the $L^2$ error for the linear anisotropic experiment with functions given by (5.1) is displayed.

experiment. The right hand side source function $f$ is manufactured by computing $\mathcal{L}u(\boldsymbol{x}_i) = f(\boldsymbol{x}_i)$ for each center $\boldsymbol{x}_i$, The values of $f$ are computed by using tensor products of Gauss-Legendre nodes to approximate the integral (2.2).

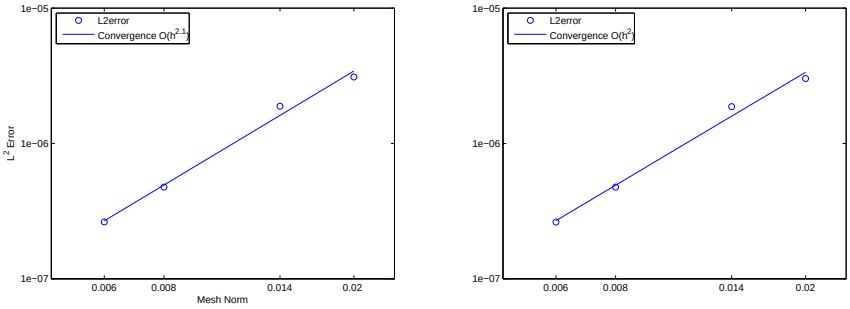
We study $L^2$ convergence of the discrete solution by constructing sets of uniformly spaced centers and sets of scattered centers with various mesh norms. Uniformly spaced collections of centers $X_h$ are constructed using grid spacing $h = .04, .02, .014, .008,$ and $.006$. Collections of scattered centers $\tilde{X}_h$ are constructed by modifying centers in $X_h$ by a random perturbation of magnitude at most $\frac{2h}{15}$. Local Lagrange functions for each collection of centers are constructed to build the discretization space. The convergence of the discrete solution $u_h$ to the solution $u$ is measured by plotting the $L^2$ norm of the error $\|u_h - u\|_{L^2(\Omega\cup\Omega_{\mathcal{I}})}$ against the mesh norm $h$. We expect for $u \in W_2^k(\Omega \cup \Omega_{\mathcal{I}})$ that $\|u - u_h\|_{L^2(\Omega)} \leq Ch^k \|u\|_{W_2^k(\Omega)}$.

**5.1. Linear Anisotropy Experiment.** We choose solution a $u$ and a kernel $\gamma$ with anisotropy function $\kappa$ and radial function $\Phi$ given by

$$\begin{cases} u(\boldsymbol{x}) = \sin(2\pi x_1)\sin(2\pi x_2)\mathbb{1}_{\Omega}(\boldsymbol{x}) \\ \kappa(\boldsymbol{x}) = 1 + x_1 + x_2 \\ \Phi(\|\boldsymbol{x} - \boldsymbol{y}\|) = \exp\big(-(1 - \epsilon^{-2}\|\boldsymbol{x} - \boldsymbol{y}\|^2)^{-1}\big) \end{cases} \tag{5.1}$$

and we discretize (4.1) with local Lagrange functions.

5.1 displays the observed $L^2$ convergence rates with respect to the mesh norm $h$ for the uniformly spaced and scattered centers experiments. The log of the computed $L^2$ error versus the log of the mesh norm is presented along with a best fit line to estimate the convergence order of the observed data. 5.1 displays the condition numbers of the discrete stiffness matrices. The observed condition numbers of the stiffness matrices do not increase as the mesh norm decreases, which matches the result of 4.1.




(a) Uniformly Spaced Centers (b) Scattered Centers

Fig. 5.2: The log of $h$ versus the log of the $L^2$ error for the exponential anisotropy experiment with functions given by (5.2) is displayed.

Table 5.1: The mesh norm $h$, number of rows $n$ of the stiffness matrix, and the estimated condition number for the stiffness matrix with the linear anisotropy (5.1) and the exponential anisotropy (5.2). The condition number of the stiffness matrices does not increase as $h$ decreases.

| | | Approximate Condition Number | |
|---|---|---|---|
| $h$ | $n$ | Linear | Exponential |
| 2.83e-2 | 625 | 58 | 89 |
| 1.41e-2 | 2500 | 59 | 90 |
| 9.9e-3 | 5041 | 59 | 90 |
| 5.7e-3 | 15625 | 60 | 92 |
| 4.2e-3 | 27889 | 60 | 92 |

**5.2. Exponential Anisotropy Experiment.** We choose solution a $u$ and a kernel $\gamma$ with anisotropy function $\kappa$ and radial function $\Phi$ given by

$$\begin{cases} u(\boldsymbol{x}) = \big(x_1(x_1-1)\big)^{\frac{3}{2}}\big(x_2(x_2-1)\big)^{\frac{3}{2}}\mathbb{1}_{\Omega}(\boldsymbol{x}) \\ \kappa_2(\boldsymbol{x}) = \exp(x_1+x_2) \\ \Phi(\|\boldsymbol{x}-\boldsymbol{y}\|) = \exp\big(-(1-\epsilon^{-2}\|\boldsymbol{x}-\boldsymbol{y}\|^2)^{-1}\big) \end{cases} \tag{5.2}$$

and we discretize (4.1) with local Lagrange functions.

5.2 displays the $L^2$ convergence plots for the experiments involving $u_2$ and $\kappa_2$. 5.1 displays the condition numbers for the discrete stiffness matrices of various values for $h$. The expected $h^2$ order convergence is observed in both the uniformly spaced centers and the scattered centers experiment. The condition number of the discrete stiffness matrices does not grow as the mesh norm decreases, which matches the prediction in 4.1.

**5.3. Vanishing Nonlocality.** We present numerical results for experiments that investigate the effects of shrinking the horizon $\epsilon$. The nonlocal operator $\mathcal{L}$ may converge to a classical differential operator of the form $-\nabla\cdot\boldsymbol{C}\cdot\nabla$ [2]. We consider

anisotropic kernels of the form

$$\gamma_\epsilon(\boldsymbol{x},\boldsymbol{y}) = \frac{1}{\epsilon^3}\big(\kappa(\boldsymbol{x})+\kappa(\boldsymbol{y})\big)\Phi(\frac{1}{\epsilon}\|\boldsymbol{x}-\boldsymbol{y}\|), \tag{5.3}$$

where $\Phi(\frac{1}{\epsilon}\|\boldsymbol{x}\|)$ is a compactly supported radial function with support radius $\epsilon$. We investigate approximating the solution to an anisotropic differential equation by solving an anisotropic nonlocal problem with sufficiently small horizon $\epsilon$. Numerical experiments demonstrate that the discrete solution to the anisotropic nonlocal problem converges to the solution of the anisotropic differential equation.

A Taylor series expansion argument can be used to find the differential operator $D$ that the nonlocal operator $\mathcal{L}_\epsilon$ approximates in the small horizon limit. We assume that $\kappa, u : \mathbb{R} \to \mathbb{R}$ are smooth functions. Then, for fixed $x \in \Omega$, we apply a Taylor series expansion in a ball $B_\epsilon(x)$ to obtain for some $\zeta, \eta \in B_\epsilon(x)$

$$\begin{aligned} u(y)-u(x) &= u'(x)(y-x)+\frac{1}{2}u''(x)(y-x)^2+\frac{1}{6}u''(\zeta)(y-x)^3 \\ \kappa(x)+\kappa(y) &= 2\kappa(x)+\kappa'(x)(y-x)+\kappa''(x)(y-x)^2+\frac{1}{6}\kappa''(\eta)(y-x)^3. \end{aligned}$$

For smooth $u$, it follows that

$$\begin{aligned} \mathcal{L}_\epsilon u(x) = \frac{1}{\epsilon^3}\big(&2u'(x)\kappa(x)\int_{-\epsilon}^{\epsilon} z\Phi(\frac{1}{\epsilon}|z|)\,dz + u'(x)\kappa'(x)\int_{-\epsilon}^{\epsilon} z\Phi(\frac{1}{\epsilon}|z|)dz\big) \\ &+\frac{1}{\epsilon^3}\big(2u''(x)\kappa(x)\int_{-\epsilon}^{\epsilon} z^2\Phi(\frac{1}{\epsilon}|z|)\,dz + u''(x)\kappa'(x)\int_{-\epsilon}^{\epsilon} z^3\Phi(\frac{1}{\epsilon}|z|)\,dz + ...\big) \end{aligned}$$

where we have truncated the expression to exclude any of the $(y-x)^3$ terms. The $z\Phi(\frac{1}{\epsilon}|z|)$ integrals vanish since $z\Phi(\frac{1}{\epsilon}|z|)$ is an odd function. We exclude the $z^3$ integrals since

$$\frac{1}{\epsilon^3}\int_{-\epsilon}^{\epsilon} z^3\Phi(\frac{1}{\epsilon}|z|) \le \frac{1}{2}\epsilon\|\Phi\|_{L^\infty(\Omega)},$$

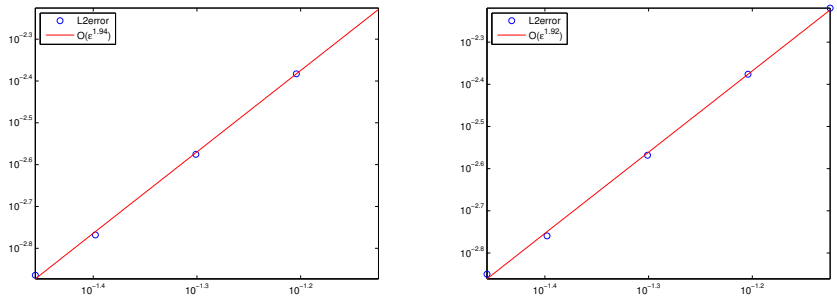
which is $O(\epsilon)$. Eliminating these terms, we compute

$$\begin{aligned} \mathcal{L}_\epsilon u(x) &\approx 2\big(u'(x)k'(x)+u''(x)k(x)\big)\int_{-\epsilon}^{\epsilon} z^2\Phi(\frac{|z|}{\epsilon})\,dz \\ &= 2\big(u'(x)k'(x)+u''(x)k(x)\big)\int_{-1}^{1}\tau^2\Phi(|\tau|)d\tau. \end{aligned}$$

Therefore, as $\epsilon$ decreases to zero,

$$\begin{aligned} \mathcal{L}_\epsilon u(x) &\to \rho\big(u'(x)k'(x)+u''(x)k(x)\big) \\ \rho &:= 2\int_{-1}^{1}\tau^2\Phi(|\tau|)\,d\tau \end{aligned}$$

We numerically experiment with a Lagrange function discretization to solve the problem $\mathcal{L}_\epsilon u_\epsilon = f$ for anisotropic nonlocal operators. Let $u$ denote the solution to $Du = f$ and let $h$ be a given mesh norm. We solve $\mathcal{L}_\epsilon u_\epsilon = f$ by discretizing the problem with Lagrange functions to construct an approximate solution $u_{\epsilon,h}$. We numerically demonstrate that as $\epsilon \to 0$, $\|u-u_{\epsilon,h}\|_{L^2(\Omega\cup\Omega_{\mathcal{I}})} \sim O(\epsilon^2)$.




(a) $\kappa(x,y) = \exp(x+y)$ (b) $\kappa(x,y) = 1+x+y$

Fig. 5.3: The log of $\epsilon$ vs. the log of the $L^2$ error of the discrete solution $u_{\epsilon,h}$ is plotted. As $\epsilon$ goes to zero, we observe $\epsilon^2$ convergence.

We let $\Phi(\frac{1}{\epsilon}\|x\|) = \big(1-\frac{1}{\epsilon^2}\|x\|^2\big)\mathbb{1}_{\|x\|<\epsilon}(x)$ and we consider two separate anisotropy functions $\kappa(x,y)$. We first consider a case of a linear anisotropic function of the form $\kappa_1(x,y) = 1+x+y$ and $\kappa_2(x,y) = \exp(x+y)$. We set $\gamma_\epsilon$ as in (5.3) with the two choices for $\kappa$. The mesh norm $h = .000075$ is fixed for the experiments and we consider a range of $\epsilon$ values from $.075, .0625, .05, .04$, and $.035$. We discretize the problem $\mathcal{L}_\epsilon u_\epsilon = f$ with Lagrange functions and a discrete solution $u_{\epsilon,h}$ is computed as described in 4.

We choose

$$u(x) = \big(1-\cos(2\pi x)\big)\cdot \mathbb{1}_{[0,1]}(x) \tag{5.4}$$

and we analytically compute $Du = f$, where $D$ is the differential operator that $\mathcal{L}_\epsilon$ converges to. We compute,

$$f(x) = \begin{cases} -2\pi\big(\sin(2\pi x) + 2\pi(1+x)\cos(2\pi x)\big) & \text{for } \kappa_1 \\ -\exp(x)\big(2\pi\sin(2\pi x) + 4\pi^2\cos(2\pi x)\big) & \text{for } \kappa_2 \end{cases}$$

In contrast to the experiments in Section 5, the right hand side is fixed, $h$ is fixed, and $\epsilon$ changes. The function $f$ is chosen to be the solution to the problem $Du = f$, where $u$ is the fixed function (5.4). As $\epsilon$ goes to zero, we expect the solution $u_\epsilon$ should converge to $u$. We numerically investigate how the discrete solution to the nonlocal problem, $u_{\epsilon,h}$ converges to the solution to the differential equation $Du = f$. As can be seen in Figure 5.3, for both $\kappa_1$ and $\kappa_2$, the $L^2$ error $\|u-u_{\epsilon,h}\|_{L^2[0,1]}$ converges at about $O(\epsilon^2)$. The numerical results suggest it is possible to approximate the solution to an anisotropic differential equation by discretizing and solving an anisotropic nonlocal volume constrained equation.

**5.4. Quadrature Experiment.** We present a numerical experiment for the proposed local Lagrange quadrature method introduced in 3.2. The tests are computed on the set $\Omega\cup\Omega_{\mathcal{I}}$ where $\Omega = (0,1)\times(0,1)$ and $\Omega_{\mathcal{I}} = [-\frac{1}{4},\frac{5}{4}]\times[-\frac{1}{4},\frac{5}{4}]\backslash\Omega$. Let $X \subset \Omega\cup\Omega_{\mathcal{I}}$ be a collection of scattered centers and let $\hat{w}_i$ denote the quadrature

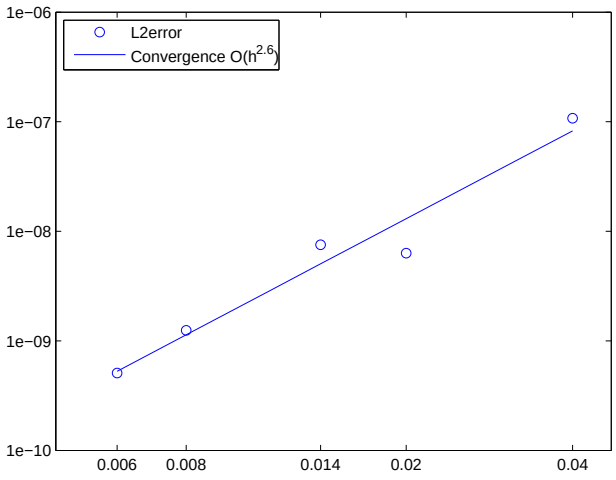



Fig. 5.4: The log of the quadrature error (5.5) versus the log of the mesh norm $h$ is displayed for the function (5.6).

weight centered at the point $\boldsymbol{x}_i \in X$. We choose a function $u : \Omega \cup \Omega_{\mathcal{I}} \to \mathbb{R}$ and we investigate how the quadrature error, given by

$$\left| \int_{\Omega \cup \Omega_{\mathcal{I}}} u(\boldsymbol{x})\, d\boldsymbol{x} - \sum_{\boldsymbol{x}_i \in X} u(\boldsymbol{x}_i)\hat{w}_i \right|, \tag{5.5}$$

varies as we decrease the mesh norm. We expect the quadrature error to decrease as $O(h^k)$ for $u \in W_2^k(\Omega)$. We choose the function

$$u(\boldsymbol{x}) = \big(x_1(1-x_1)\big)^2 \big(x_2(1-x_2)\big)^2 \mathbb{1}_{\Omega} \tag{5.6}$$

and we consider various sets of centers with mesh norms $h = .04, .02, .014, .008$, and $.006$. 5.4 shows the results for an experiment using the polynomial function (5.6), which exhibits a convergence rate of $h^{2.5}$.

**5.5. Graphics Processing Units.** We consider the acceleration of radial basis function computations using graphics processing units (GPUs). GPUs may provide substantial acceleration for algorithms that require the same instructions to be executed on multiple data simultaneously. Many computations involving radial basis functions exhibit parallelism that may be exploited. We consider the example of computing the local Lagrange quadrature weights introduced in 3.2.

For a collection of centers $X \subset \Omega$, the local Lagrange quadrature weight may be constructed by directly integrating

$$\hat{w}_i := \sum_{\boldsymbol{x}_j \in \Upsilon_i} \alpha_{i,j} \int_{\Omega} \varphi(\|\boldsymbol{x} - \boldsymbol{x}_j\|) + \sum_{l=1}^{n} \beta_{l,j} \int_{\Omega} p_l(\boldsymbol{x})\, d\boldsymbol{x}.$$

This requires the integration of the functions $\varphi(\|\boldsymbol{x} - \boldsymbol{x}_j\|)$ and subsequent multiplication by the local Lagrange coefficients $\alpha_{i,j}$. Each quadrature weight $\hat{w}_i$ has few

coefficients as described in 3.1. The computation time is primarily the computation of the integrals of the shifts of the radial basis function $\varphi(\|\boldsymbol{x} - \boldsymbol{x}_j\|)$. To numerically evaluate these integrals, we use tensor product Gauss-Legendre quadrature nodes and weights in $\Omega$. We approximate

$$\int_\Omega \varphi(\|\boldsymbol{x} - \boldsymbol{x}_j\|)\, d\boldsymbol{x} \approx \sum_{g=1}^{G} \varphi(\|\boldsymbol{y}_g - \boldsymbol{x}_j\|) q_g \tag{5.7}$$

where $\boldsymbol{y}_g$ is the quadrature node and $q_g$ is the quadrature weight. We note that with this fixed collection of quadrature weights, the summation in (5.7) requires the same operation repeatedly, with different input data $\boldsymbol{y}_g - \boldsymbol{x}_j$.

After the integrals of the shifts of $\varphi$ are computed, each weight is assembled by multiplying by the corresponding $\alpha_{i,j}$ coefficients for $\boldsymbol{x}_j \in \Upsilon_i$. The multiplications require the same operation repeatedly with different input data, which may be executed quickly on a GPU. Applying this method on a 96 CUDA core Nvidia Geforce GT430 yielded at least a four times speed-up vs. an OpenMP implementation. For example, the construction of 5776 quadrature weights required 550 miliseconds on the 96 CUDA core device versus 2 seconds on the CPU.

We discuss Kokkos, a Trilinos package that enables code portability for parallel algorithms to various architectures. Kokkos supports multicore CPU parallelism such as OpenMP and pthreads, GPU parallelism with CUDA, and parallelism on Intel Xeon Phi co-processors. One objective of Kokkos is to minimize the amount of architecture specific knowledge the programmer is required to know. The Kokkos API reduces the need for the user to focus on device specific programming models without sacrificing the performance of writing code directly in terms of the device's native language. Kokkos handles various architectures by correctly choosing the memory layout of arrays depending on the device being used. Kokkos automatically chooses the optimal layout of the arrays. The interested reader should consult [3] for details.

Kokkos provided our work with a way to rapidly implement parallel numerical algorithms without focusing on the details of the parallel method. The same code we wrote for OpenMP parallelism in Kokkos could be used for CUDA with no changes necessary. With an easy to use API, the computational kernels

**6. Acknowledgments.** The authors would like to thank Carter Edwards and Eric Phipps for their invaluable assistance on this project. Eric Phipps provided access to CUDA capable computers. Without this, the GPU acceleration portion of this project would likely not have been pursued. Carter Edwards provided essential support to the project and introduced the second author to the Kokkos library and supported the second author's summer work.

REFERENCES

[1] S. D. Bond, R. B. Lehoucq, and S. T. Rowe, *A Galerkin radial basis function method for nonlocal diffusion*, Tech. Rep. SAND 2013-10673P, 2014. To Appear in the Proceedings of the Seventh International Workshop on Meshfree Methods for Partial Differential Equations.

[2] Q. Du, M. Gunzburger, R. B. Lehoucq, and K. Zhou, *Analysis and approximation of nonlocal diffusion problems with volume constraints*, SIAM Review, 54 (2012), pp. 667–696.

[3] H. C. Edwards, C. R. Trott, and D. Sunderland, *Kokkos: Enabling manycore performance portability through polymorphic memory access patterns*, Journal of Parallel and Distributed Computing, 74 (2014), pp. 3202 – 3216. Domain-Specific Languages and High-Level Frameworks for High-Performance Computing.

[4] E. J. Fuselier, T. Hangelbroek, F. J. Narcowich, J. D. Ward, and G. B. Wright, *Localized bases for kernel spaces on the unit sphere*, SIAM J. Numer. Anal., 51 (2013).
[5] T. Hangelbroek, F. J. Narcowich, C. Rieger, and J. D. Ward, *An inverse theorem on bounded domains for meshless methods using localized bases*, ArXiv e-prints, (2014).
[6] H. Wendland, *Scattered Data Approximation*, Cambridge University Press, 2005.

# A NEW PARTITIONED ALGORITHM FOR EXPLICIT ELASTODYNAMICS BASED ON VARIATIONAL FLUX RECOVERY

PAUL KUBERRY* AND PAVEL BOCHEV†

**Abstract.** Problems involving an interface arise frequently, whether as a surface between two distinct physical systems or in decomposing one system's domain into two or more domains. It is generally the case that the geometries for two domains differ greatly. When using Galerkin finite elements to solve partial differential equations that have boundary conditions over an interface, often finite elements of differing polynomial degree are used. From this arises the need to accurately and stably communicate solution information across interfaces having mismatched vertices. We present a new partitioned algorithm for explicit elastodynamics, based on variational flux recovery. This method works by the exchange of tractions which depend on both the known state and the unknown state at the future time step. The exchange of recovered tractions between the bodies formally involves modification of both the forcing term and the mass matrix on each subdomain. Reliance on the future time step distinguishes our approach from traditional partitioned solution algorithms, which use interface conditions between subdomains and the current time step solution to define boundary conditions for the models. Our approach offers some distinct numerical and theoretical advantages. It passes a linear patch test and is second order accurate in space. Furthermore, if interface grids match, our new partitioned method recovers the solution of a monolithic coupling scheme for the solid-solid interaction problem.

**Acknowledgment.** This work has been funded by LDRD. We thank our colleagues M. Aguilo, E. Love, D. Ridzal, A. Robinson, and M. Wong for stimulating discussions about the research problem. Special thanks are owed to D. Ridzal for help with the Intrelab package.

**Nomenclature.**

$\Omega_i$ – subdomain occupied by $i^{\text{th}}$, $i = 1, 2$ elastic solid;
$\sigma$ – interface between subdomains $\Omega_1$ and $\Omega_2$;
$\Gamma_i$ – Dirichlet boundary of $\Omega_i$; $\Gamma_i = \partial\Omega_i/\sigma$;
$\mathbf{H}(\Omega_i)$ – Sobolev space for the displacements on $\Omega_i$;
$\mathbf{H}_{\Gamma_i}(\Omega_i)$ – subspace of all functions in $\mathbf{H}(\Omega_i)$, which vanish on $\Gamma_i$;
$\Omega_i^h$ – finite element partition of $\Omega$;
$\sigma_i^h$ – finite element partition of $\sigma$ induced by subdomain mesh $\Omega_i^h$;
$V(\sigma_i^h)$ – set of all interface vertices in finite element partition $\Omega_i^h$;
$V(\breve{\Omega}_i^h)$ – set of all interior vertices in finite element partition $\Omega_i^h$;
$S_i^h$ – conforming finite element subspace of $\mathbf{H}(\Omega_i)$ defined on $\Omega_i^h$;
$N_{i,k}$ – nodal basis of $S_i^h$;
$S_{i,\Gamma}^h$ – conforming finite element subspace of $\mathbf{H}_{\Gamma_i}(\Omega_i)$;
$S_{i,\sigma}^h$ – interface part of $S_i^h$: span of all basis functions associated with vertices in $\sigma_i^h$;
$S_{i,0}^h$ – interior part of $S_i^h$: all displacements vanishing on $\partial\Omega_i$;
$\Pi_1$ interface transfer map $S_{2,\sigma}^h \mapsto S_{1,\sigma}^h$;
$\Pi_2$ interface transfer map: $S_{1,\sigma}^h \mapsto S_{2,\sigma}^h$;
$\Pi_{(\boldsymbol{u})}$ interface transfer map, which depends on the exact solution
$\boldsymbol{u}_i^h$ – finite element function in $S_i^h$;
$\boldsymbol{u}_{i,\sigma}^h$ – interface part of $\boldsymbol{u}_i^h$; $\boldsymbol{u}_{i,\sigma}^h \in S_{i,\sigma}^h$;
$\boldsymbol{u}_{i,0}^h$ – interior part of $\boldsymbol{u}_i^h$; $\boldsymbol{u}_{i,0}^h \in S_{i,0}^h$;
$\vec{\boldsymbol{u}}_i$ – coefficient vector of $\boldsymbol{u}_i^h$;

*Clemson University, pkuberr@clemson.edu
†Sandia National Laboratories, pbboche@sandia.gov

$\vec{\boldsymbol{u}}_{i,\sigma}$ – coefficient vector of interface part $\boldsymbol{u}^h_{i,\sigma}$;
$\vec{\boldsymbol{u}}_{i,0}$ – coefficient vector of interior part $\boldsymbol{u}^h_{i,0}$;
$M_i$ – mass matrix associated with $S(\Omega^h_i)$;
$M_{i,\sigma}$ – volume mass matrix associated with interface trace space $S^h_{i,\sigma}$;
$\overline{M}_{i,\sigma}$ – area mass matrix associated with interface trace space $S^h_{i,\sigma}$;
$M_{i,0}$ – volumetric mass matrix associated with subspace $S^h_{i,0}$;

**1. Introduction.** In this report we present a new partitioned method for solid-solid interaction (SSI) problems, based on variational flux recovery. Simulation of two interacting solids appears in a variety of science and engineering problems. Our principal driver is simulation of electromechanical systems comprising a piezoelectric material mounted on some structure. However, since our main focus is on the coupling procedure, it suffices to consider the case of small displacements and linear elastic models without incorporating a piezoelectric constitutive relation into one of them. Finally, we restrict attention to the case of spatially coincident interfaces, in which the interfacial grids induced by subdomain partitioning may not be matching, but are spatially coincident.

Numerical methods for such SSI problems follow two general paths. Monolithic schemes view the governing equations together with the appropriate interface conditions as a single, fully coupled system of partial differential equations. Its discretization yields a coupled system of algebraic equations for the approximate numerical solution of the SI problem.

Traditional partitioned methods on the other hand discretize and solve the governing equations for each one of the bodies independently and use the interface conditions to couple the two problems. Typically this is accomplished by using the interface conditions to form boundary conditions, which pass displacements and/or tractions between the bodies.

In this paper we develop a new coupling scheme for SSI, which shares some common elements with both monolithic and partitioned methods, but is distinct from both. Specifically, in our approach the two solid models are discretized independently on their respective subdomains and the fully discrete equations are advanced in time by an explicit scheme. This part of the scheme resembles a traditional partitioned approach. However, unlike the latter, at the beginning of each time step the two models exchange fluxes given in terms of both the *known* solution at the *current time step* and the *unknown* solution at the *future time step*. Exchange of subdomain fluxes results in an update of both the forcing term on each subdomain *and* the subdomain mass matrix. The latter represents modification of an operator that is independent of the solution and so it needs to be performed once at the beginning of the simulation and every time the mesh defining the mass operator is changed. This part of our scheme resembles the assembly process of a monolithic problem, and in fact can be shown to yield the same system as the latter for matching interface grids. However, unlike monolithic schemes, we discretize the two solids completely independently from each other without any assumptions on interface grids.

Our approach offers some distinct numerical and theoretical advantages. It passes a linear patch test and is second order accurate in space. Furthermore, if interface grids match, the new partitioned method recovers the solution of a monolithic coupling scheme for the solid-solid interaction problem.

**1.1. Notations.** We consider two elastic bodies occupying non-overlapping bounded regions ("subdomains") $\Omega_1$ and $\Omega_2$ such that $\Omega_1 \cap \Omega_2 = \emptyset$ and $\overline{\Omega}_1 \cap \overline{\Omega}_2 = \sigma$. We refer

to $\sigma$ as the interface between the bodies, $\Gamma_i = \partial\Omega_i/\sigma$ are the Dirichlet boundaries of the subdomains and $\Omega = \Omega_1 \cup \Omega_2$ is the problem domain.

Let $\Omega_i^h$ denote a partition of $\Omega_i$, $i = 1, 2$ into finite elements. The subdomain partitions induce finite element partitions $\sigma_1^h$ and $\sigma_2^h$ of the interface $\sigma$. The interface partitions are not required to match but are assumed to be spatially coincident, i.e., there are no voids or overlaps between $\Omega_1^h$ and $\Omega_2^h$. The mesh vertices are $\boldsymbol{x}_i$, where $i$ is the vertex ordinal.

A minimal requirement for any mesh-tying method is a consistency condition called *patch test*. A method passes a patch test of order $k$ if it can recover any solution of (2.1) that is a polynomial of degree $k$.

**1.1.1. Finite element operators.** We define the interface transfer operator $\Pi_1 : S_{2,\sigma}^h \mapsto S_{1,\sigma}^h$ as the interpolant of a function $\boldsymbol{u}_{2,\sigma} \in S_{2,\sigma}^h$ in $S_{2,\sigma}^h$. Such functions have the form

$$\boldsymbol{u}_{2,\sigma} = \sum_{k \in V(\sigma_2^h)} (\vec{\boldsymbol{u}}_{2,\sigma})_k N_{2,k}(\boldsymbol{x}) \,,$$

and so it is straightforward to see that

$$\Pi_1(\boldsymbol{u}_{2,\sigma}) = \sum_{i \in V(\sigma_1^h)} \boldsymbol{u}_{2,\sigma}(\boldsymbol{x}_{1,i}) N_{1,i}(\boldsymbol{x}) = \sum_{i \in V(\sigma_1^h)} \left( \sum_{k \in V(\sigma_2^h)} (\vec{\boldsymbol{u}}_{2,\sigma})_k N_{2,k}(\boldsymbol{x}_{1,i}) \right) N_{1,i}(\boldsymbol{x}) \,.$$

Because basis functions $N_{2,k}$ have local support, the interior sum above reduces to

$$\sum_{k \in V(K_2 \ni \boldsymbol{x}_{1,i})} (\vec{\boldsymbol{u}}_{2,\sigma})_k N_{2,k}(\boldsymbol{x}_{1,i}) \tag{1.1}$$

where $V(K_2 \ni \boldsymbol{x}_{1,i})$ are the vertices of element $K_2 \in \sigma_2^h$ containing vertex $\boldsymbol{x}_{1,i}$ from $\sigma_1^h$. It follows that the coefficient vector of the function $\Pi_1(\boldsymbol{u}_{2,\sigma})$ is given by $P_1 \vec{\boldsymbol{u}}_{2,\sigma}$ where $P_1$ is $|V(\sigma_1^h)| \times |V(\sigma_2^h)|$ sparse matrix. The row of this matrix corresponding to vertex $\boldsymbol{x}_{1,i}$ contains the values $N_{2,k}(\boldsymbol{x}_{1,i})$ for $k \in V(K_2 \ni \boldsymbol{x}_{1,i})$.

**2. Governing equations.** Consider two linear elastic bodies occupying bounded regions $\Omega_1$ and $\Omega_2$, respectively. We assume that $\Omega_1 \cap \Omega_2 = \emptyset$ and denote $\Omega = \Omega_1 \cup \Omega_2$, $\Gamma_1 = \partial\Omega_1/\sigma$, and $\Gamma_2 = \partial\Omega_2/\sigma$; see Figure 2.1. The *interface* between the two bodies, $\sigma = \overline{\Omega}_1 \cap \overline{\Omega}_2$, is a connected, non empty set. A choice of a unit normal $\boldsymbol{n}_\sigma$ on $\sigma$ orients the interface. Without loss of generality we assume that $\boldsymbol{n}_\sigma$ coincides with the outer unit normal to $\partial\Omega_1$.

The coupled solid-solid interaction problem comprises a pair of governing equations

$$\left\{ \begin{array}{rcll} \ddot{\boldsymbol{u}}_i - \nabla \cdot \sigma(\boldsymbol{u}_i) &=& \mathbf{f} & \text{in } \Omega_i \times [0,T] \\ \boldsymbol{u}_i(0,\boldsymbol{x}) &=& \boldsymbol{u}_0(\boldsymbol{x}) & \text{in } \Omega_i \\ \dot{\boldsymbol{u}}_i(0,\boldsymbol{x}) &=& \dot{\boldsymbol{u}}_0(\boldsymbol{x}) & \text{in } \Omega_i \\ \boldsymbol{u}_i &=& \boldsymbol{g} & \text{on } \Gamma_i \times [0,T] \end{array} \right. \qquad i = 1, 2 \tag{2.1}$$

for displacements $\boldsymbol{u}_i(t,\boldsymbol{x})$, $i = 1, 2$ of the elastic bodies, and a pair of interface conditions

$$\boldsymbol{u}_1(\boldsymbol{x},t) = \boldsymbol{u}_1(\boldsymbol{x},t) \quad \text{and} \quad \sigma_1(\boldsymbol{x},t) \cdot \boldsymbol{n}_\sigma = \sigma_2(\boldsymbol{x},t) \cdot \boldsymbol{n}_\sigma \quad \text{on } \sigma \times [0,T]. \tag{2.2}$$

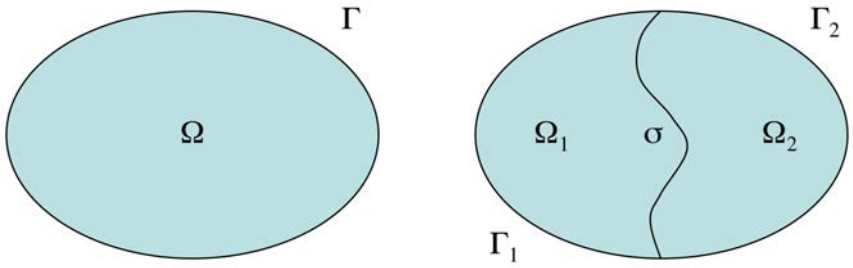



Fig. 2.1: Two elastic bodies occupy subdomains $\Omega_1$ and $\Omega_2$ forming a bounded region $\Omega$.

expressing continuity of the displacement and the traction across the interface. We restrict attention to linear elastodynamic problems for which the stress

$$\sigma(\boldsymbol{u}_i) = \lambda_i(\nabla \cdot \boldsymbol{u}_i)I + 2\mu_i\varepsilon(\boldsymbol{u}_i)$$

and the strain

$$\varepsilon(\boldsymbol{u}_i) = \frac{1}{2}(\nabla \boldsymbol{u}_i + \nabla \boldsymbol{u}_i^T)$$

The Lame coefficients $\lambda_i$ and $\mu_i$ are allowed to have a discontinuity along $\sigma$.

**3. Partitioned solution algorithm.** In this report we develop a new partitioned solution algorithm for (2.1) based on variational flux recovery. A formal splitting of the problem into two "independent" mixed boundary value subdomain equations is the starting point in the formulation. This partitioning is formal because it imposes the unknown traction value on the interface as a Neumann boundary condition and resulting solutions satisfy a weak continuity relation in terms of an operator that is not available in closed form. Using variational flux recovery ideas we eliminate the unknown traction from the subdomain equations. In so doing we obtain two fully decoupled subdomain equations which implicitly incorporate appropriate discrete notions of the interface conditions (2.2).

**3.1. Formal partitioning of the governing equations.** Let $\boldsymbol{u}_i$, $i = 1, 2$ denote the exact solutions of (2.1)–(2.2) and

$$\gamma = \sigma_1(\boldsymbol{x}, t) \cdot \boldsymbol{n}_\sigma = \sigma_2(\boldsymbol{x}, t) \cdot \boldsymbol{n}_\sigma$$

be the corresponding exact interface traction. Suppose that $\gamma$ is known exactly. Then, displacement $\boldsymbol{u}_i$ on $\Omega_i$ can be determined by solving the following mixed boundary value problem:

$$\begin{cases} \ddot{\boldsymbol{u}}_i - \nabla \cdot \sigma_i = \mathbf{f} & \text{in } \Omega_i \times [0, T] \\ \boldsymbol{u}_i(0, \boldsymbol{x}) = \boldsymbol{u}_0(\boldsymbol{x}) & \text{in } \Omega_i \\ \dot{\boldsymbol{u}}_i(0, \boldsymbol{x}) = \dot{\boldsymbol{u}}_0(\boldsymbol{x}) & \text{in } \Omega_i \\ \boldsymbol{u}_i = \boldsymbol{g} & \text{on } \Gamma_i \\ \sigma_i(\boldsymbol{x}, t) \cdot \boldsymbol{n}_\sigma = \gamma & \text{on } \Gamma_i \times [0, T] \end{cases} \tag{3.1}$$

The exact traction function $\gamma$ provides a Neumann boundary condition on the interface, which closes the subdomain problems and makes it possible to solve them independently from each other. Note that by the uniqueness of the solutions of the original formulation (2.1) and (3.1) it follows that the solutions of the latter necessarily satisfy the first interface condition in (2.2), i.e., $\boldsymbol{u}_1 = \boldsymbol{u}_2$ on $\sigma$.

The corresponding weak forms of the subdomain equations (3.1) are to seek $\boldsymbol{u}_1 \in \mathbf{H}(\Omega_1)$ and $\boldsymbol{u}_2 \in \mathbf{H}(\Omega_2)$ such that

$$\begin{aligned} (\ddot{\boldsymbol{u}}_1, \boldsymbol{v}_1)_{\Omega_1} + (\sigma_1, \varepsilon(\boldsymbol{v}_1))_{\Omega_1} &= (\mathbf{f}, \boldsymbol{v}_1)_{\Omega_1} + \langle \gamma, \boldsymbol{v}_1 \rangle_\sigma \quad \forall \boldsymbol{v}_1 \in \mathbf{H}_{\Gamma_1}(\Omega_1) \\ (\ddot{\boldsymbol{u}}_2, \boldsymbol{v}_2)_{\Omega_2} + (\sigma_2, \varepsilon(\boldsymbol{v}_2))_{\Omega_2} &= (\mathbf{f}, \boldsymbol{v}_2)_{\Omega_2} - \langle \gamma, \boldsymbol{v}_2 \rangle_\sigma \quad \forall \boldsymbol{v}_2 \in \mathbf{H}_{\Gamma_2}(\Omega_2) \end{aligned} \tag{3.2}$$

Again, solutions of (3.2) necessarily satisfy the first interface condition in (2.2), i.e., $\boldsymbol{u}_1 = \boldsymbol{u}_2$ on $\sigma$. We refer to (3.1) and (3.2) as formal partitionings of the SSI problem, because in practice the exact traction $\gamma$ is not known on the interface.

**3.2. Spatial discretization.** To discretize (3.2) in space we restrict the test functions $\boldsymbol{v}_i$ to a finite element space $S^h_{i,\Gamma}$ and seek $\boldsymbol{u}_i \in S^h_i \times [0, T]$ which satisfies the initial and boundary conditions prescribed by (3.1) and such that

$$\begin{aligned} (\ddot{\boldsymbol{u}}_1, \boldsymbol{v}_1)_{\Omega_1} + (\sigma(\boldsymbol{u}_1), \varepsilon(\boldsymbol{v}_1))_{\Omega_1} &= (\mathbf{f}, \boldsymbol{v}_1)_{\Omega_1} + \langle \gamma, \boldsymbol{v}_1 \rangle_\sigma \quad \forall \boldsymbol{v}_1 \in S^h_{1,\Gamma} \\ (\ddot{\boldsymbol{u}}_2, \boldsymbol{v}_2)_{\Omega_2} + (\sigma(\boldsymbol{u}_1), \varepsilon(\boldsymbol{v}_2))_{\Omega_2} &= (\mathbf{f}, \boldsymbol{v}_2)_{\Omega_2} - \langle \gamma, \boldsymbol{v}_2 \rangle_\sigma \quad \forall \boldsymbol{v}_2 \in S^h_{2,\Gamma} \end{aligned} \tag{3.3}$$

In general the subdomain grids $\Omega^h_1$ and $\Omega^h_2$ induce non-matching finite element partitions $\sigma^h_1$ and $\sigma^h_2$ of interface $\sigma$. As a result, solutions of (3.2) cannot satisfy a pointwise version of the first interface condition in (2.2). Instead they satisfy some "weaker" notion of displacement continuity, which we formally express as

$$\boldsymbol{u}_{1,\sigma} = \Pi_1(\boldsymbol{u})\boldsymbol{u}_{2,\sigma} \quad \text{and} \quad \boldsymbol{u}_{2,\sigma} = \Pi_2(\boldsymbol{u})\boldsymbol{u}_{1,\sigma} \tag{3.4}$$

where $\Pi_1(\boldsymbol{u}) : S^h_1 \mapsto S^h_2$ and $\Pi_2(\boldsymbol{u}) : S^h_2 \mapsto S^h_1$ are some unknown operators.

**3.3. Elimination of the surface traction.** The discrete subdomain equations (3.3) remain coupled by virtue of the interface traction $\gamma$ and the weak continuity condition (3.4). Because $\gamma$ and $\Pi_i(\boldsymbol{u})$ are unknown, this also means that (3.3) are not solvable in their current form. In this section we focus on the elimination of $\gamma$ from subdomain equations and the algebraic form of the resulting equations. We discuss the handling of the displacement continuity condition and the complete separation of the discrete problems in Section 3.4.

To explain elimination of surface traction we rewrite (3.3) in a block form corresponding to the partitioning of $S^h_i$ into an interfacial part $S^h_{i,\sigma}$ and a zero trace part $S^h_{i,0}$, along with the appropriate weak continuity equation. This form is given by

$$\begin{cases} (\ddot{\boldsymbol{u}}_{1,\sigma}, N_{1,i})_{\Omega_1} + (\sigma(\boldsymbol{u}_1), \varepsilon(N_{1,i}))_{\Omega_1} = (\mathbf{f}, N_{1,i})_{\Omega_1} + \langle \gamma, N_{1,i} \rangle_\sigma & \forall i \in V(\sigma^h_1) \\ (\ddot{\boldsymbol{u}}_{1,0}, N_{1,i})_{\Omega_1} + (\sigma(\boldsymbol{u}_1), \varepsilon(N_{1,i}))_{\Omega_1} = (\mathbf{f}, N_{1,i})_{\Omega_1} & \forall i \in V(\breve{\Omega}^h_1) \\ \boldsymbol{u}_{1,\sigma} = \Pi_1(\boldsymbol{u})\boldsymbol{u}_{2,\sigma} \end{cases} \tag{3.5}$$

on the first subdomain and

$$\begin{cases} (\ddot{\boldsymbol{u}}_{2,\sigma}, N_{2,i})_{\Omega_2} + (\sigma(\boldsymbol{u}_2), \varepsilon(N_{2,i}))_{\Omega_2} = (\mathbf{f}, N_{2,i})_{\Omega_2} - \langle \gamma, N_{2,i} \rangle_\sigma & \forall i \in V(\sigma_2^h) \\ (\ddot{\boldsymbol{u}}_{2,0}, N_{2,i})_{\Omega_2} + (\sigma(\boldsymbol{u}_2), \varepsilon(N_{2,i}))_{\Omega_2} = (\mathbf{f}, N_{2,i})_{\Omega_2} & \forall i \in V(\breve{\Omega}_2^h) \\ \boldsymbol{u}_{2,\sigma} = \Pi_2(\boldsymbol{u}) \boldsymbol{u}_{1,\sigma} \end{cases} \tag{3.6}$$

is the analogous form on the second subdomain.

We now proceed to eliminate the unknown traction $\gamma$ from (3.5) by using (3.6) and vice versa. Solving the interface equations in (3.6) for the traction yields

$$\langle \gamma, N_{2,i} \rangle_\sigma = (\mathbf{f}, N_{2,i})_{\Omega_2} - (\sigma(\boldsymbol{u}_2), \varepsilon(N_{2,i}))_{\Omega_2} - (\ddot{\boldsymbol{u}}_{2,\sigma}, N_{2,i})_{\Omega_2} \quad \forall i \in V(\sigma_2^h) \tag{3.7}$$

Equation (3.7) defines a finite element approximation $\gamma_2 \in S^h_{2,\sigma}$ of the interface traction in terms of $\ddot{\boldsymbol{u}}_{2,\sigma}$ and $\boldsymbol{u}_2$. It can be interpreted as variational recovery [1] of $\gamma$ from a finite element solution. To emphasize dependence of the recovered traction on finite element solution we write $\gamma_2(\ddot{\boldsymbol{u}}_{2,\sigma}, \boldsymbol{u}_2)$.

To complete the elimination process we approximate the unknown traction $\gamma$ in (3.5) by the interpolant $\Pi_1 \gamma_2 \in S^h_{1,\sigma}$ of the recovered traction. This yields the following system of equations on the first subdomain:

$$\begin{cases} (\ddot{\boldsymbol{u}}_{1,\sigma}, N_{1,i})_{\Omega_1} + (\sigma(\boldsymbol{u}_1), \varepsilon(N_{1,i}))_{\Omega_1} = (\mathbf{f}, N_{1,i})_{\Omega_1} + \langle \Pi_1 \gamma_2(\ddot{\boldsymbol{u}}_{2,\sigma}, \boldsymbol{u}_2), N_{1,i} \rangle_\sigma & \forall i \in V(\sigma_1^h) \\ (\ddot{\boldsymbol{u}}_{1,0}, N_{1,i})_{\Omega_1} + (\sigma(\boldsymbol{u}_1), \varepsilon(N_{1,i}))_{\Omega_1} = (\mathbf{f}, N_{1,i})_{\Omega_1} & \forall i \in V(\breve{\Omega}_1^h) \\ \boldsymbol{u}_{1,\sigma} = \Pi_1(\boldsymbol{u}) \boldsymbol{u}_{2,\sigma} \end{cases} \tag{3.8}$$

Conversely, using (3.5) to eliminate $\gamma$ from (3.6) we obtain an analogous equation on $\Omega_2$:

$$\begin{cases} (\ddot{\boldsymbol{u}}_{2,\sigma}, N_{2,i})_{\Omega_2} + (\sigma(\boldsymbol{u}_2), \varepsilon(N_{2,i}))_{\Omega_2} = (\mathbf{f}, N_{2,i})_{\Omega_2} - \langle \Pi_2 \gamma_1(\ddot{\boldsymbol{u}}_{1,\sigma}, \boldsymbol{u}_1), N_{2,i} \rangle_\sigma & \forall i \in V(\sigma_2^h) \\ (\ddot{\boldsymbol{u}}_{2,0}, N_{2,i})_{\Omega_2} + (\sigma(\boldsymbol{u}_2), \varepsilon(N_{2,i}))_{\Omega_2} = (\mathbf{f}, N_{2,i})_{\Omega_2} & \forall i \in V(\breve{\Omega}_2^h) \\ \boldsymbol{u}_{2,\sigma} = \Pi_2(\boldsymbol{u}) \boldsymbol{u}_{1,\sigma} \end{cases} \tag{3.9}$$

The semi-discrete in space problems (3.8)–(3.9) have convenient matrix forms. Let $\vec{F}_i$ denote a vector with interface and interior parts $\vec{F}_{i,\sigma}$ and $\vec{F}_{i,0}$, respectively, and element

$$\vec{F}_i^k = (\mathbf{f}, N_{i,k})_{\Omega_i} - (\sigma(\boldsymbol{u}_i), \varepsilon(N_{i,k}))_{\Omega_i} \quad \forall k \in V(\Omega_i^h)\,. \tag{3.10}$$

Then, the interface equation in (3.8) can be written as

$$M_{1,\sigma} \ddot{\boldsymbol{u}}_{1,\sigma} = \vec{F}_{1,\sigma} + \overline{M}_{1,\sigma} P_1 \gamma_2(\ddot{\boldsymbol{u}}_{2,\sigma}, \boldsymbol{u}_2)\,, \tag{3.11}$$

whereas the matrix form of equation (3.7), which defines $\gamma_2$, is given by

$$\overline{M}_{2,\sigma} \gamma_2 = \vec{F}_{2,\sigma} - M_{2,\sigma} \ddot{\boldsymbol{u}}_{2,\sigma}\,.$$

Solving the latter for $\gamma_2$ yields

$$\gamma_2(\ddot{\boldsymbol{u}}_{2,\sigma}, \boldsymbol{u}_2) = \overline{M}_{2,\sigma}^{-1} \vec{F}_{2,\sigma} - \overline{M}_{2,\sigma}^{-1} M_{2,\sigma} \ddot{\boldsymbol{u}}_{2,\sigma}\,.$$

The algebraic form of (3.8) follows by substituting this result into (3.11):

$$\begin{cases} M_{1,\sigma}\, \ddot{\boldsymbol{u}}_{1,\sigma} + \overline{M}_{1,\sigma} P_1 \overline{M}_{2,\sigma}^{-1} M_{2,\sigma}\, \ddot{\boldsymbol{u}}_{2,\sigma} &=& \vec{F}_{1,\sigma} + \overline{M}_{1,\sigma} P_1 \overline{M}_{2,\sigma}^{-1} \vec{F}_{2,\sigma} \\ M_{1,0} \ddot{\boldsymbol{u}}_{1,0} &=& \vec{F}_{1,0} \\ \boldsymbol{u}_{1,\sigma} &=& P_1(\boldsymbol{u}) \boldsymbol{u}_{2,\sigma} \end{cases} \tag{3.12}$$

Proceeding along the same lines we obtain an analogous algebraic form for (3.8):

$$\left\{\begin{array}{rcl} M_{2,\sigma}\,\ddot{\boldsymbol{u}}_{2,\sigma} + \overline{M}_{2,\sigma}P_2\overline{M}_{1,\sigma}^{-1}M_{1,\sigma}\,\ddot{\boldsymbol{u}}_{1,\sigma} &=& \vec{F}_{2,\sigma} + \overline{M}_{2,\sigma}P_2\overline{M}_{1,\sigma}^{-1}\vec{F}_{1,\sigma} \\ M_{2,0}\ddot{\boldsymbol{u}}_{2,0} &=& \vec{F}_{2,0} \\ \boldsymbol{u}_{2,\sigma} &=& P_2(\boldsymbol{u})\boldsymbol{u}_{1,\sigma} \end{array}\right. \tag{3.13}$$

**3.4. Elimination of displacement continuity equations.** Equations (3.12)–(3.13) remain coupled through their dependence on interface states from both subdomains. One possible decoupling strategy is to approximate $P_i(\boldsymbol{u}_i)$ by a computable operator $\widetilde{P}_i$ and then use displacement continuity equations to eliminate states from the adjacent domain, resultinging in subdomain equations that can be solved independently from each other. Because the exact nature of $P_i(\boldsymbol{u}_i)$ is not generally known, construction of $\widetilde{P}_i$ may not be straightforward. However, under some additional assumptions on the matrix structure it turns out that $P_i(\boldsymbol{u}_i)$ can be effectively approximated by the interface interpolant $P_i$ in which case the weak continuity equations in (3.12)–(3.13) are replaced by

$$\boldsymbol{u}_{1,\sigma} = P_1\boldsymbol{u}_{2,\sigma} \quad \text{and} \quad \boldsymbol{u}_{2,\sigma} = P_2\boldsymbol{u}_{1,\sigma}\,, \tag{3.14}$$

respectively. The key factor that enables such an approximation is to work with diagonal mass matrices. Thus, from now on we assume that (i) assembly is performed using node-based quadrature rules, which result in

$$M_{i,\sigma} = diag(m_{i,\sigma}^k) \quad \text{and} \quad \overline{M}_{i,\sigma} = diag(\overline{m}_{i,\sigma}^k); \quad i = 1, 2\,,$$

and (ii) displacement continuity conditions are given by (3.14). For clarity we explain elimination of interface states from the opposite side of the interface in a two-dimensional seting. In this case matrix forms of interface transfer operators $\Pi_i$ assume a particularly simple form with at most two non-zero elements per row. We explain the structure of $P_1$. Let $\boldsymbol{x}_{1,i} \in \sigma_1^h$ be an arbitrary vertex on the interface of $\Omega_1$ and $K_{2,k_i} \in \sigma_2^h$ be the element from the interface of $\Omega_2$, which contains[1] $\boldsymbol{x}_{1,i}$.

Since $\sigma$ is one-dimensional, element $K_{2,k_i}$ is an interval with endpoints $\boldsymbol{x}_{2,k_i-1}$ and $\boldsymbol{x}_{2,k_i}$, respectively. As a result, (1.1) reduces to the following sum

$$\sum_{k\in V(K_{2,k_i})} \vec{\boldsymbol{u}}_{2,\sigma}^k N_{2,k}(\boldsymbol{x}_{1,i}) = \vec{\boldsymbol{u}}_{2,\sigma}^{k_i-1} N_{2,k_i-1}(\boldsymbol{x}_{1,i}) + \vec{\boldsymbol{u}}_{2,\sigma}^{k_i} N_{2,k_i}(\boldsymbol{x}_{1,i}) \tag{3.15}$$

Since basis functions form a partition of unity on every element,

$$N_{2,k_i-1}(\boldsymbol{x}_{1,i}) + N_{2,k_i}(\boldsymbol{x}_{1,i}) = 1$$

and so, there exists $0 \le \alpha_i \le 1$ such that

$$N_{2,k_i-1}(\boldsymbol{x}_{1,i}) = \alpha_{1,i} \quad \text{and} \quad N_{2,k_i}(\boldsymbol{x}_{1,i}) = 1 - \alpha_{1,i}$$

It follows that the matrix $P_1$ is given by

$$(P_1)_{ij} = \left\{\begin{array}{rl} \alpha_{1,i} & \text{if } j = k_i - 1 \\ 1 - \alpha_{1,i} & \text{if } j = k_i \\ 0 & \text{otherwise} \end{array}\right. \tag{3.16}$$

[1] If $\boldsymbol{x}_{1,i}$ is also a vertex in $\sigma_2^h$, then it is shared by two elements in $\sigma_h^2$. In this case we can take $K_{2,k_i}$ to be either one of these two elements.

where $K_{2,k_i} = [\boldsymbol{x}_{2,k_i-1}, \boldsymbol{x}_{2,k_i}]$ is the element from the interface on $\Omega_2$ containing vertex $\boldsymbol{x}_{1,i}$ from the interface on $\Omega_1$. Repeating the same arguments for $\Pi_2$ shows that

$$(P_2)_{ij} = \begin{cases} \alpha_{2,i} & \text{if } j = k_i - 1 \\ 1 - \alpha_{2,i} & \text{if } j = k_i \\ 0 & \text{otherwise} \end{cases} \tag{3.17}$$

where $K_{1,k_i} = [\boldsymbol{x}_{1,k_i-1}, \boldsymbol{x}_{1,k_i}]$ is the element from the interface on $\Omega_1$ containing vertex $\boldsymbol{x}_{2,i}$ from the interface on $\Omega_2$ and

$$\alpha_{2,i} = N_{1,k_i-1}(\boldsymbol{x}_{2,i}) \quad \text{and} \quad 1 - \alpha_{2,i} = N_{1,k_i}(\boldsymbol{x}_{2,i})\,.$$

Since interior equations are fully decoupled from the interface equations we focus solely on the structure of the latter. Their right hand sides are given by

$$\left(\vec{F}_{1,\sigma} + \overline{M}_{1,\sigma} P_1 \overline{M}_{2,\sigma}^{-1} \vec{F}_{2,\sigma}\right)_j = F_{1,\sigma}^j + \overline{m}_{1,\sigma}^j \left[\alpha_{1,j} \frac{F_{2,\sigma}^{k_j-1}}{\overline{m}_{2,\sigma}^{k_j-1}} + (1-\alpha_{1,j}) \frac{F_{2,\sigma}^{k_j}}{\overline{m}_{2,\sigma}^{k_j}}\right]$$

for the interface equation on $\Omega_1$ and

$$\left(\vec{F}_{2,\sigma} + \overline{M}_{2,\sigma} P_2 \overline{M}_{1,\sigma}^{-1} \vec{F}_{1,\sigma}\right)_j = F_{2,\sigma}^j + \overline{m}_{2,\sigma}^j \left[\alpha_{2,j} \frac{F_{1,\sigma}^{k_j-1}}{\overline{m}_{1,\sigma}^{k_j-1}} + (1-\alpha_{2,j}) \frac{F_{1,\sigma}^{k_j}}{\overline{m}_{1,\sigma}^{k_j}}\right],$$

for the interface equation on $\Omega_2$.

Consider next the terms involving displacements from the opposite sides of the interface, that is, $\overline{M}_{1,\sigma} P_1 \overline{M}_{2,\sigma}^{-1} M_{2,\sigma} \ddot{\boldsymbol{u}}_{2,\sigma}$ in (3.12) and $\overline{M}_{2,\sigma} P_2 \overline{M}_{1,\sigma}^{-1} M_{1,\sigma} \ddot{\boldsymbol{u}}_{1,\sigma}$ in (3.13). We have that

$$\left(\overline{M}_{1,\sigma} P_1 \overline{M}_{2,\sigma}^{-1} M_{2,\sigma} \ddot{\boldsymbol{u}}_{2,\sigma}\right)_j = \overline{m}_{1,\sigma}^j \left[\alpha_{1,j} \frac{m_{2,\sigma}^{k_j-1}}{\overline{m}_{2,\sigma}^{k_j-1}} \ddot{\boldsymbol{u}}_{2,\sigma}^{k_j-1} + (1-\alpha_{1,j}) \frac{m_{2,\sigma}^{k_j}}{\overline{m}_{2,\sigma}^{k_j}} \ddot{\boldsymbol{u}}_{2,\sigma}^{k_j}\right] \tag{3.18}$$

and

$$\left(\overline{M}_{2,\sigma} P_2 \overline{M}_{1,\sigma}^{-1} M_{1,\sigma} \ddot{\boldsymbol{u}}_{1,\sigma}\right)_j = \overline{m}_{2,\sigma}^j \left[\alpha_{2,j} \frac{m_{1,\sigma}^{k_j-1}}{\overline{m}_{1,\sigma}^{k_j-1}} \ddot{\boldsymbol{u}}_{1,\sigma}^{k_j-1} + (1-\alpha_{2,j}) \frac{m_{1,\sigma}^{k_j}}{\overline{m}_{1,\sigma}^{k_j}} \ddot{\boldsymbol{u}}_{1,\sigma}^{k_j}\right]. \tag{3.19}$$

For shape-regular grids it is not unreasonable to expect that

$$\frac{m_{2,\sigma}^{k_j-1}}{\overline{m}_{2,\sigma}^{k_j-1}} \approx \frac{m_{2,\sigma}^{k_j}}{\overline{m}_{2,\sigma}^{k_j}} := \mu_{2,\sigma}^j \quad \text{and} \quad \frac{m_{1,\sigma}^{k_j-1}}{\overline{m}_{1,\sigma}^{k_j-1}} \approx \frac{m_{1,\sigma}^{k_j}}{\overline{m}_{1,\sigma}^{k_j}} := \mu_{1,\sigma}^j\,. \tag{3.20}$$

This assumption allows us to exchange the order of interpolation and matrix multiplication in (3.18) to obtain

$$\left(\overline{M}_{1,\sigma} P_1 \overline{M}_{2,\sigma}^{-1} M_{2,\sigma} \ddot{\boldsymbol{u}}_{2,\sigma}\right)_j = \overline{m}_{1,\sigma}^j \mu_{2,\sigma}^j \left[\alpha_{1,j} \ddot{\boldsymbol{u}}_{2,\sigma}^{k_j-1} + (1-\alpha_{1,j}) \ddot{\boldsymbol{u}}_{2,\sigma}^{k_j}\right] = \overline{m}_{1,\sigma}^j \mu_{2,\sigma}^j (P_1 \ddot{\boldsymbol{u}}_{2,\sigma})_j$$

Likewise, exchanging the order of operators in (3.19) gives

$$\left(\overline{M}_{2,\sigma} P_2 \overline{M}_{1,\sigma}^{-1} M_{1,\sigma} \ddot{\boldsymbol{u}}_{1,\sigma}\right)_j = \overline{m}_{2,\sigma}^j \mu_{1,\sigma}^j \left[\alpha_{2,j} \ddot{\boldsymbol{u}}_{1,\sigma}^{k_j-1} + (1-\alpha_{2,j}) \ddot{\boldsymbol{u}}_{1,\sigma}^{k_j}\right] = \overline{m}_{2,\sigma}^j \mu_{1,\sigma}^j (P_2 \ddot{\boldsymbol{u}}_{1,\sigma})_j\,.$$

From the weak continuity equations (3.14) it follows that

$$\ddot{\boldsymbol{u}}_{1,\sigma}^j = (P_1 \ddot{\boldsymbol{u}}_{2,\sigma})_j \quad \text{and} \quad \ddot{\boldsymbol{u}}_{2,\sigma}^j = (P_2 \ddot{\boldsymbol{u}}_{1,\sigma})_j\,.$$

Using these identities we can eliminate $\ddot{\boldsymbol{u}}_{2,\sigma}$ from (3.18) and $\ddot{\boldsymbol{u}}_{1,\sigma}$ from (3.19) to obtain

$$\left(\overline{M}_{1,\sigma}P_1\overline{M}_{2,\sigma}^{-1}M_{2,\sigma}\right)\ddot{\boldsymbol{u}}_{2,\sigma} \approx \left(\overline{M}_{1,\sigma}\mu_{2,\sigma}\right)\ddot{\boldsymbol{u}}_{1,\sigma}$$

and

$$\left(\overline{M}_{2,\sigma}P_2\overline{M}_{1,\sigma}^{-1}M_{1,\sigma}\ddot{\boldsymbol{u}}_{1,\sigma}\right) \approx \left(\overline{M}_{2,\sigma}\mu_{1,\sigma}\right)\ddot{\boldsymbol{u}}_{2,\sigma}\,,$$

respectively. This completes the decoupling of (3.12)–(3.13) into an independent subdomain equation

$$\left\{\begin{array}{rcl}\left(M_{1,\sigma}+\overline{M}_{1,\sigma}\mu_{2,\sigma}\right)\ddot{\boldsymbol{u}}_{1,\sigma} &=& \vec{F}_{1,\sigma}+\overline{M}_{1,\sigma}P_1\overline{M}_{2,\sigma}^{-1}\vec{F}_{2,\sigma}\\ M_{1,0}\,\ddot{\boldsymbol{u}}_{1,0} &=& \vec{F}_{1,0}\end{array}\right. \tag{3.21}$$

for $\Omega_1$, and another independent subdomain equation

$$\left\{\begin{array}{rcl}\left(M_{2,\sigma}+\overline{M}_{2,\sigma}\mu_{1,\sigma}\right)\ddot{\boldsymbol{u}}_{2,\sigma} &=& \vec{F}_{2,\sigma}+\overline{M}_{2,\sigma}P_2\overline{M}_{1,\sigma}^{-1}\vec{F}_{1,\sigma}\\ M_{2,0}\ddot{\boldsymbol{u}}_{2,0} &=& \vec{F}_{2,0}\end{array}\right. \tag{3.22}$$

on $\Omega_2$. The interface equations in each subdomain have the following component form:

$$\left(m_{1,\sigma}^j+\overline{m}_{1,\sigma}^j\mu_{2,\sigma}^j\right)\ddot{\boldsymbol{u}}_{1,\sigma}^j = F_{1,\sigma}^j+\overline{m}_{1,\sigma}^j\left[\alpha_{1,j}\frac{F_{2,\sigma}^{k_j-1}}{\overline{m}_{2,\sigma}^{k_j-1}}+(1-\alpha_{1,j})\frac{F_{2,\sigma}^{k_j}}{\overline{m}_{2,\sigma}^{k_j}}\right];\quad j\in V(\sigma_1^h) \tag{3.23}$$

and

$$\left(m_{2,\sigma}^j+\overline{m}_{2,\sigma}^j\mu_{1,\sigma}^j\right)\ddot{\boldsymbol{u}}_{2,\sigma}^j = F_{2,\sigma}^j+\overline{m}_{2,\sigma}^j\left[\alpha_{2,j}\frac{F_{1,\sigma}^{k_j-1}}{\overline{m}_{1,\sigma}^{k_j-1}}+(1-\alpha_{2,j})\frac{F_{1,\sigma}^{k_j}}{\overline{m}_{1,\sigma}^{k_j}}\right];\quad j\in V(\sigma_2^h) \tag{3.24}$$

Modification of subdomain mass matrices in (3.23)–(3.24) can be interpreted as their completion to bulk mass matrices on $S_{1,\sigma}^h\cup S_{2,\sigma}^h$. Section 4.1 examines further this connection for matching interface grids.

**3.5. Fully discrete partitioned equations.** For brevity, we restrict attention to explicit time discretization of (3.23)–(3.24) using second central difference in time, i.e., we approximate $\ddot{\boldsymbol{u}}_i$ by

$$\ddot{\boldsymbol{u}}_i(t,\boldsymbol{x}) \approx \frac{\boldsymbol{u}_i(t+\Delta t,\boldsymbol{x})-2\boldsymbol{u}_i(t,\boldsymbol{x})+\boldsymbol{u}_i(t-\Delta t,\boldsymbol{x})}{\Delta t^2}\,.$$

Let $\boldsymbol{u}_i^{n+1}\in S_i^h$, $\boldsymbol{u}_i^n\in S_i^h$ and $\boldsymbol{u}_i^{n-1}\in S_i^h$ denote finite element approximations of $\boldsymbol{u}_i$ at $t_n+\Delta t$, $t_n$ and $t_{n-1}=t_n-\Delta t$, respectively, $\ddot{D}^{n+1}(\boldsymbol{u}_i)=(\boldsymbol{u}_i^{n+1}-2\boldsymbol{u}_i^n+\boldsymbol{u}_i^{n-1})/\Delta t^2$, and $(\vec{F}_i)^n$ be the force vector (3.10) evaluated at $\boldsymbol{u}_i^n$. Then, for given $\boldsymbol{u}_i^n$ and $\boldsymbol{u}_i^{n-1}$, the fully discrete partitioned formulation is to find $\boldsymbol{u}_1^{n+1}$ such that

$$\left\{\begin{array}{rcl}\left(M_{1,\sigma}+\overline{M}_{1,\sigma}\mu_{2,\sigma}\right)\ddot{D}^{n+1}(\boldsymbol{u}_{1,\sigma}) &=& (\vec{F}_{1,\sigma})^n+\overline{M}_{1,\sigma}P_1\overline{M}_{2,\sigma}^{-1}(\vec{F}_{2,\sigma})^n\\ M_{1,0}\ddot{D}^{n+1}(\boldsymbol{u}_{1,0}) &=& (\vec{F}_{1,0})^n\end{array}\right. \tag{3.25}$$

and $\boldsymbol{u}_2^{n+1}$ such that

$$\begin{cases} \left(M_{2,\sigma} + \overline{M}_{2,\sigma}\mu_{1,\sigma}\right)\ddot{D}^{n+1}(\boldsymbol{u}_{2,\sigma}) &= (\vec{F}_{2,\sigma})^n + \overline{M}_{2,\sigma}P_2\overline{M}_{1,\sigma}^{-1}(\vec{F}_{1,\sigma})^n \\ M_{2,0}\ddot{D}^{n+1}(\boldsymbol{u}_{2,0}) &= (\vec{F}_{2,0})^n \end{cases} \tag{3.26}$$

for the finite element approximations $\boldsymbol{u}_i^{n+1}$, $i = 1, 2$ of the subdomain solutions at $t_{n+1}$.

**4. Properties of the VFR partitioned formulation.** In this section we show that for matching interface grids the partitioned formulation is equivalent to a monolithic explicit elastodynamics formulation. We also show that our formulation passes a linear patch test for arbitrary interface grids.

**4.1. Equivalence to a monolithic discretization on matching interface grids.** Suppose that finite element partitions of $\Omega_1$ and $\Omega_2$ are such that interface grids match, i.e, there is a common interface partition $\sigma^h$ such that $\sigma^h = \sigma_1^h = \sigma_2^h$ Then, the combined grid on the two subdomains forms a conforming partition of $\Omega_1 \cup \Omega_2$ and $S^h = S_1^h \cup S_2^h$ is a conforming finite element subspace of $\mathbf{H}^1(\Omega)$. This space gives rise to a fully discrete monolithic formulation of (2.1)

$$M\ddot{D}^{n+1}\boldsymbol{v} = (\vec{F})^n\,.$$

where $M$ and $\vec{F}$ are a diagonal mass matrix and force vector assembled using $S^h$. Partitioning of mesh nodes into interface and subdomain nodes induces partitioning of the solution vector $\boldsymbol{v}$ into coefficient vectors $\boldsymbol{v}_\sigma$, $\boldsymbol{v}_{1,0}$ and $\boldsymbol{v}_{2,0}$ corresponding to interface and interior subdomain nodes, respectively. As a result, we can write the monolithic problem in the following block diagonal form:

$$\begin{cases} M_\sigma\ddot{D}^{n+1}(\boldsymbol{v}_\sigma) &= (\vec{F}_\sigma)^n \\ M_{1,0}\ddot{D}^{n+1}(\boldsymbol{v}_{1,0}) &= (\vec{F}_{1,0})^n \\ M_{2,0}\ddot{D}^{n+1}(\boldsymbol{v}_{2,0}) &= (\vec{F}_{2,0})^n \end{cases} \tag{4.1}$$

We now examine the relationship between the solution of (4.1) and the partitioned SSI problem.

THEOREM 4.1. *Assume that interface grids $\sigma_1^h$ and $\sigma_2^h$ are matching and interface displacements at all previous time steps coincide:*

$$(\vec{\boldsymbol{u}}_{1,\sigma})^\nu = (\vec{\boldsymbol{u}}_{2,\sigma})^\nu \quad \nu = 1, 2, \ldots, n-1, n\,. \tag{4.2}$$

*Then the partitioned solution $(\vec{\boldsymbol{u}}_{1,\sigma}, \vec{\boldsymbol{u}}_{1,0})$, $(\vec{\boldsymbol{u}}_{2,\sigma}, \vec{\boldsymbol{u}}_{2,0})$ coincides with the solution $\vec{\boldsymbol{v}} = (\vec{\boldsymbol{v}}_\sigma, \vec{\boldsymbol{v}}_{1,0}, \vec{\boldsymbol{v}}_{2,0})$ of the monolithic problem:*

$$\vec{\boldsymbol{v}}_\sigma = \vec{\boldsymbol{u}}_{1,\sigma} = \vec{\boldsymbol{u}}_{2,\sigma}; \quad \vec{\boldsymbol{u}}_{1,0} = \vec{\boldsymbol{v}}_{1,0} \quad \textit{and} \quad \vec{\boldsymbol{u}}_{2,0} =, \vec{\boldsymbol{v}}_{2,0}\,.$$

*Proof.* For clarity we present the proof for the two-dimensional formulation (3.21)–(3.22). Owing to the assumption that interface grids on $\Omega_1$ and $\Omega_2$ match, it follows that the area mass matrices $\overline{M}_{1,\sigma}$ and $\overline{M}_{2,\sigma}$ are identical, i.e., they have the same dimension and with proper renumbering of their elements we can write

$$\overline{m}_{1,\sigma}^j = \overline{m}_{2,\sigma}^j \qquad \forall j \in V(\sigma_1^h) \equiv V(\sigma_2^h)$$

For matching interface grids we also have that $P_1 = P_1 = I$. As a result, the interface equations assume the form

$$\left(m^j_{1,\sigma} + m^j_{2,\sigma}\right) \boldsymbol{u}^j_{1,\sigma} = F^j_{1,\sigma} + F^j_{2,\sigma} \qquad \forall j \in V(\sigma^h_1)$$

and

$$\left(m^j_{2,\sigma} + m^j_{1,\sigma}\right) \boldsymbol{u}^j_{2,\sigma} = F^j_{2,\sigma} + F^j_{1,\sigma} \qquad \forall j \in V(\sigma^h_2),$$

respectively. Thus, for matching interface grids our partitioned SSI formulation has the following matrix form

$$\left\{ \begin{array}{rcl} (M_{1,\sigma} + M_{2,\sigma})\vec{\boldsymbol{u}}_{1,\sigma} & = & \vec{F}_{1,\sigma} + \vec{F}_{2,\sigma} \\ M_{1,0}\vec{\boldsymbol{u}}_{1,0} & = & \vec{F}_{1,0} \end{array} \right. \tag{4.3}$$

and

$$\left\{ \begin{array}{rcl} (M_{2\sigma} + M_{1\sigma})\vec{\boldsymbol{u}}_{2,\sigma} & = & \vec{F}_{2,\sigma} + \vec{F}_{1,\sigma} \\ M_{2,0}\vec{\boldsymbol{u}}_{2,0} & = & \vec{F}_{2,0} \end{array} \right. \tag{4.4}$$

It immediately follows that

$$\vec{\boldsymbol{u}}_{1,0} = \vec{\boldsymbol{v}}_{1,0} \quad \text{and} \quad \vec{\boldsymbol{u}}_{2,0} = \vec{\boldsymbol{v}}_{2,0}\,.$$

On the other hand, it is easy to see that for matching interface partitions, the monolithic volume interface mass matrix is sum of the volume interface mass matrices on $\Omega_1$ and $\Omega_2$:

$$M_\sigma = M_{1,\sigma} + M_{2,\sigma}\,.$$

Furthermore, if (4.2) holds, a direct calculation shows that the monolithic interface force vector is sum of the interface force vectors on $\Omega_1$ and $\Omega_2$:

$$\vec{F}_\sigma = \vec{F}_{1,\sigma} + \vec{F}_{2,\sigma}\,.$$

Therefore, $\vec{\boldsymbol{u}}_{1,\sigma}$ and $\vec{\boldsymbol{u}}_{2,\sigma}$ solve an identical equation, which coincides with the monolithic interface equation and so,

$$\vec{\boldsymbol{u}}_{1,\sigma} = \vec{\boldsymbol{u}}_{2,\sigma} = \vec{\boldsymbol{v}}_\sigma\,.$$

□

**4.2. Recovery of linear solutions.** In this section we show that the partitioned formulation recovers steady state linear solutions.

THEOREM 4.2. *Assume that $\boldsymbol{u}^n_i$ and $\boldsymbol{u}^{n-1}_i$ correspond to a constant in time, globally linear in space function defined on $\Omega_1 \cup \Omega_2$. Then, the partitioned solution $\boldsymbol{u}^{n+1}_i$ has the property that*

$$\boldsymbol{u}^{n+1}_i = \boldsymbol{u}^n_i\,. \tag{4.5}$$

*Proof.* Consider the equation on $\Omega_1$

$$\begin{cases} \left(M_{1,\sigma} + \overline{M}_{1,\sigma}\mu_{2,\sigma}\right)\ddot{D}^{n+1}(\boldsymbol{u}_{1,\sigma}) &= (\vec{F}_{1,\sigma})^n + \overline{M}_{1,\sigma}P_1\overline{M}_{2,\sigma}^{-1}(\vec{F}_{2,\sigma})^n \\ M_{1,0}\ddot{D}^{n+1}(\boldsymbol{u}_{1,0}) &= (\vec{F}_{1,0})^n \end{cases}$$

For linear functions $(\vec{F}_1)^n = 0$ and so the equations reduce to

$$M_1\ddot{D}^{n+1}(\boldsymbol{u}_1) = 0\,.$$

By assumption $\boldsymbol{u}_i^n = \boldsymbol{u}_i^{n-1}$, and so

$$\ddot{D}^{n+1}(\boldsymbol{u}_1) = \frac{\boldsymbol{u}_1^{n+1} - \boldsymbol{u}_1^n}{\Delta t^2}\,.$$

This implies $\boldsymbol{u}_1^{n+1} = \boldsymbol{u}_1^n$, which proves the theorem. □

## 5. Computational studies.

**5.1. Convergence rates.** As our first test, we consider a problem with a known solution in order to verify that the rate of convergence using the algorithm based on variational flux recovery is second order accurate in space.

The monolithic domain, $\Omega$, is the unit square $[0,1] \times [0,1]$. We investigate two configurations for the interface between the two subdomains $\Omega_1$ and $\Omega_2$, one vertical and the other diagonal. The parameters we use for the linear elasticity of a homogenous isotropic solid are $\mu = 0.01$ and $\lambda = 0.02$ $dyne/cm^2$. Throughout this paper, the assumption is made that the density of the solid is 1 $g/cm^3$ or that the equations have been scaled appropriately.

The known solution for the displacement of the solid in the vertical and horizontal directions is given by:

$$\boldsymbol{u} = \begin{pmatrix} \sin(5\pi x)\cos(3\pi y)\log(1+t) \\ 4x^4\cos(4\pi y)\sqrt{t+2} \end{pmatrix} \tag{5.1}$$

with Dirichlet boundary conditions enforced on the boundaries of $\Omega$, initial conditions, and right hand side forcing terms determined by the known solution.

For the vertical interface scenario in Figure 5.1, $\Omega_1 = [0, 0.6] \times [0,1]$, $\Omega_2 = [0.6, 1] \times [0,1]$, and $\Omega = \Omega_1 \cup \Omega_2 = [0,1] \times [0,1]$. $\Omega_1$ and $\Omega_2$ are both spatially discretized on a uniform mesh.

| | | Error/Rate | | |
|---|---|---|---|---|
| | Mesh 1 | $12 \times 20$ | $24 \times 40$ | $48 \times 80$ |
| | Mesh 2 | $20 \times 20$ | $40 \times 40$ | $80 \times 80$ |
| $\|\mathbf{u} - \mathbf{u}^h\|_0$ | $\Omega_1$ | 7.97e-03/- | 2.06e-03/1.95 | 5.12e-04/2.01 |
| | $\Omega_2$ | 2.58e-02/- | 6.42e-03/2.01 | 1.59e-03/2.01 |
| $\|\mathbf{u} - \mathbf{u}^h\|_1$ | $\Omega_1$ | 5.61e-01/- | 2.59e-01/1.11 | 1.30e-01/1.00 |
| | $\Omega_2$ | 2.11e+00/- | 1.06e+00/1.00 | 5.29e-01/1.00 |

Table 5.1: Errors and convergence rate results using a vertical interface and uniform meshes at time $t = 0.25$ $s$.

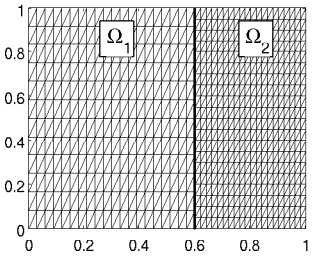



Fig. 5.1: Uniform spatial discretization of $\Omega_1$ and $\Omega_2$ with a vertical interface at $x = 0.6\ cm$.

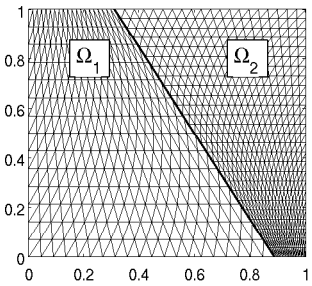



Fig. 5.2: Non shape regular spatial discretization of $\Omega$ into $\Omega_1$ and $\Omega_2$ with an interface having a slope of tan(110°) and passing through (0.6,0.5).

For the slanted interface scenario in Figure 5.2, $\Omega_1$ and $\Omega_2$ share an interface of a line with a slope of $\tan(110°) \approx -2.7475$ passing through the point (0.6,0.5) and $\Omega = \Omega_1 \cup \Omega_2 = [0,1] \times [0,1]$. $\Omega_1$ and $\Omega_2$ are both spatially discretized on meshes that are not shape regular.

| | | Error/Rate | | | |
|---|---|---|---|---|---|
| | Mesh 1 | $14 \times 20$ | $28 \times 40$ | $56 \times 80$ | $48 \times 80$ |
| | Mesh 2 | $26 \times 20$ | $52 \times 40$ | $104 \times 80$ | $48 \times 80$ |
| $\|\mathbf{u} - \mathbf{u}^h\|_0$ | $\Omega_1$ | 8.19e-03/- | 2.07e-03/1.98 | 5.16e-04/2.01 | 1.37e-04/1.92 |
| | $\Omega_2$ | 1.52e-02/- | 3.79e-03/2.00 | 9.58e-04/1.99 | 2.48e-04/1.95 |
| $\|\mathbf{u} - \mathbf{u}^h\|_1$ | $\Omega_1$ | 5.64e-01/- | 2.78e-01/1.02 | 1.39e-01/1.00 | 6.94e-02/1.00 |
| | $\Omega_2$ | 1.63e+00/- | 8.22e-01/0.99 | 4.12e-01/1.00 | 2.06e-01/1.00 |

Table 5.2: Convergence rate results using a slanted interface and non shape regular meshes at time $t = 0.25\ s$.

We observe in Tables 5.1 and 5.2 that by using a coincidental interface with nonmatching vertices and temporal step sizes on the order of $h$, the rate of convergence

is second order regardless of the interface orientation.

**5.2. Union of decomposed solutions are equivalent to monolithic solution.** Using the same problem formulation and domain previously described, consider decomposing $\Omega$ into $\Omega_1$ and $\Omega_2$ sharing an interface $\sigma$ such that the vertices on the interface are matching and coincidental. The vertical interface is at $x = 0.6\ cm$ as before, and the slanted interface has a slope of $\tan(110^\circ)$ and passes through $(0.6, 0.5)$.

| Error | | Vertical Interface | Slanted Interface |
|---|---|---|---|
| | Mesh 1 | $24 \times 20$ | $24 \times 20$ |
| | Mesh 2 | $24 \times 20$ | $24 \times 20$ |
| $\|\mathbf{u_1}^h - \mathbf{u}^h\|_0$ | $\Omega_1$ | 3.38e-17 | 9.43e-17 |
| $\|\mathbf{u_2}^h - \mathbf{u}^h\|_0$ | $\Omega_2$ | 1.07e-15 | 1.05e-15 |
| $\|\mathbf{u_1}^h - \mathbf{u}^h\|_1$ | $\Omega_1$ | 2.49e-15 | 7.74e-15 |
| $\|\mathbf{u_2}^h - \mathbf{u}^h\|_1$ | $\Omega_2$ | 9.92e-14 | 1.23e-13 |

Table 5.3: Difference between solution computed on whole domain $\mathbf{u^h}$ and solutions computed on subdomains $\Omega_1$ and $\Omega_2$, $\mathbf{u_1^h}$ and $\mathbf{u_2^h}$.

The difference in solutions, shown in Table 5.3, are equivalent up to roundoff whether computed through the monolithic formulation or using the algorithm based on variational flux recovery.

**5.3. Preservation of linear displacements.** The next test we consider is to recover the solution to a problem determined by a steady state linear solution.

The monolithic domain, $\Omega$, is the unit square $[-1, 1] \times [-1, 1]$. We investigate two configurations for the interface between the two subdomains $\Omega_1$ and $\Omega_2$, one vertical and the other diagonal. The parameters we use for the linear elasticity of a homogenous isotropic solid are $\mu = 1.5$ and $\lambda = 7\ dyne/cm^2$. Throughout this paper, the assumption is made that the density of the solid is $1\ g/cm^3$ or that the equations have been scaled appropriately.

The known solution for the displacement of the solid in the vertical and horizontal directions is given by:

$$\boldsymbol{u} = \begin{pmatrix} -5x + 50y \\ 33x - 22y \end{pmatrix} \tag{5.2}$$

with Dirichlet boundary conditions enforced on the boundaries of $\Omega$, initial conditions, and right hand side forcing terms determined by the known solution.

In the vertical interface scenario, $\Omega_1 = [-1, -0.1] \times [-1, 1]$, $\Omega_2 = [-0.1, 1] \times [-1, 1]$, and $\Omega = \Omega_1 \cup \Omega_2 = [-1, 1] \times [-1, 1]$. $\Omega_1$ and $\Omega_2$ are both spatially discretized on a uniform mesh. In the slanted interface scenario, $\Omega_1$ and $\Omega_2$ share an interface of a line with a slope of $\tan(65^\circ) \approx 2.1445$ passing through the point (-0.1,0) and $\Omega = \Omega_1 \cup \Omega_2 = [-1, 1] \times [-1, 1]$. $\Omega_1$ and $\Omega_2$ are both spatially discretized on non shape regular meshes. For both interface cases, shown in Figure 5.3, random perturbations were made to the vertices on the interface for each domain, i.e., the vertices on the interface for both $\Omega_1$ and $\Omega_2$ were moved a random amount tangent to the interface,

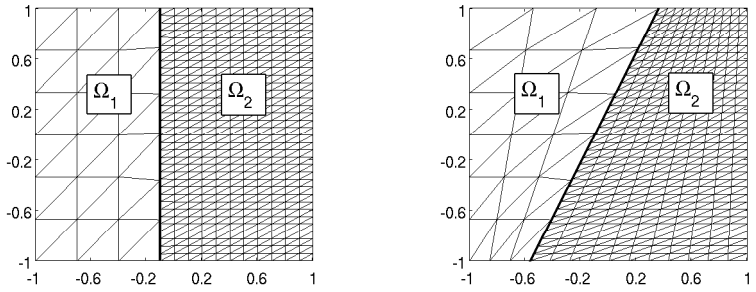



Fig. 5.3: Uniform and non uniform domain discretizations on which to recover a linear solution.

where a signed direction and magnitude were generated by a uniformly distributed random number. Vertices on the interface were allowed to slide up to 20% of the distance to the nearest vertex on the interface.

| Error | | Vertical Interface | Slanted Interface |
|---|---|---|---|
| | Mesh 1 | $6 \times 3$ | $6 \times 3$ |
| | Mesh 2 | $34 \times 11$ | $34 \times 11$ |
| $\|\mathbf{u} - \mathbf{u}^h\|_0$ | $\Omega_1$ | 3.45e-15 | 3.54e-15 |
| | $\Omega_2$ | 4.00e-15 | 4.11e-15 |
| $\|\mathbf{u} - \mathbf{u}^h\|_1$ | $\Omega_1$ | 1.16e-14 | 1.59e-14 |
| | $\Omega_2$ | 6.42e-14 | 7.85e-14 |

Table 5.4: Patch test errors at time $t = 0.05$ $s$.

In Table 5.4 we see that even with random perturbations, ensuring mismatched vertices on the interface between $\Omega_1$ and $\Omega_2$, the algorithm still recovers the linear solution up to errors introduced by roundoff, regardless of the interface orientation.

**6. Conclusions.** We have presented a new partioned method, based on variational flux recovery, for solid-solid interaction (SSI) problems. This scheme is unique from existing methods in that it exchanges fluxes that are given in terms of both the *known* solution at the *current time step* and the *unknown* solution at the *future time step*. Focusing on the case of linear elastic models based on the assumption of small displacements, we have proven and demonstrated numerically that our algorithm, when applied to spatially coincident interfaces, passes any patch test of order one. The method is second order accurate for shape regular meshes. Also, it recovers the monolithic solution when the vertices on the interface are coincidental and matched. Future work will include augmenting the method in order to conserve linear momentum.

REFERENCES

[1] G. Carey, S. Chow, and M. Seager, *Approximate boundary-flux calculations*, Computer Methods in Applied Mechanics and Engineering, 50 (1985), pp. 107–120.

# A CONSERVATIVE SEMI-LAGRANGIAN SPECTRAL ELEMENT METHOD WITH OPTIMIZATION BASED LIMITERS

SCOTT A. MOE[†], PAVEL B. BOCHEV[‡], KARA J. PETERSON[§], AND DENIS RIDZAL[¶]

**Abstract.** A Spectral Element Method with Semi-Lagrangian time-stepping for the scalar advection equation is analyzed. The method is designed to efficiently advect large numbers of passive tracers as part of a larger atmospheric model. The Semi-Lagrangian time-stepping has many advantageous properties for this application because of the need to only evolve gridpoints instead of tracer densities. The Spectral Element space discretization allows efficient coupling with established atmospheric models. The connection between the SLSEM and Characteristic Galerkin schemes is commented on and exploratory work shows that the SL scheme can be modified using ideas inspired by this connection to stabilize the method for discontinuous initial conditions. The base numerical method is inserted as part of an Optimization Based Remap framework allowing the efficient enforcement of physical constraints such as conservation. In addition this Optimization Based framework is utilized to implement limiters. The limiters are shown numerically to not greatly disturb the high-order convergence rate of the solution of smooth density profiles in the $L_1$ norm, though they do affect pointwise convergence. The optimization based limiters are generalized to equations involving zeroth order source terms (including divergent flows) using a "fictitious density" approach to define a quantity that should display monotonicity even when the conserved quantity itself will not.

## 1. Introduction.

$$\frac{\partial q}{\partial t} + \nu \cdot \nabla(q) = 0 \tag{1.1}$$

Equation (1.1) describes a passive tracer advected by velocity field $\nu$. This equation is relatively simple, however atmospheric models often are required to solve it for arbitrarily many separate tracers. Thus despite its simplicity, this equation is responsible for a very large portion of the computational cost of many atmospheric models. This being the case, it is important to solve this equation in a robust and efficient manner. In addition the method used to solve equation (1.1) should use the information already present in the main atmospheric model (ideally it would use the same spatial discretization as the core scheme).

The Spectral Element Method (SEM) is a spatial discretization that has proven effective for atmospheric modeling [13] because of its diagonal mass matrix and arbitrary order accuracy. However the SEM, like other high order Finite Element Methods, suffers from a severe stability imposed time-step restriction for Hyperbolic PDEs. This can be avoided by using an implicit or IMEX method, but these are not the most attractive methods for advecting hundreds of passive tracers computationally. Schemes based on the Method of Characteristics have long been used as extremely efficient methods for solving scalar hyperbolic (or parabolic but nearly hyperbolic) equations [4]. These methods take the Lagrangian viewpoint, directly approximating the analytically known movements of "fluid particles" under the influence of the velocity field.

An important complaint, though, about Semi-Lagrangian schemes is that they do not necessarily preserve some properties that physically must hold. Most importantly they are not, in general, mass conservative. There has been work on combining the Semi-Lagrangian method with a Spectral Element Space discretization [9, 15, 8].

[†]University of Washington Department of Applied Mathematics, smoe@uw.edu
[‡]Sandia National Laboratories, pbboche@sandia.gov
[§]Sandia National Laboratories, kjpeter@sandia.gov
[¶]Sandia National Laboratories, dridzal@sandia.gov

However in order to use a Semi-Lagrangian SEM in an actual atmospheric model it will be necessary to have a method to enforce physical constraints. For hyperbolic PDEs generally the weak form of the equation is more fundamental, physically, than the differential form. So it is possible to have even strong discontinuities in solutions that are physically observed. Approximation theory states that any high order method predicated on an orthogonal function expansion must illustrate large oscillations (known as Runge or Gibbs phenomena) in this situation. Limiting is necessary to remove these non-physical oscillations introduced by the discretization.

The basic approach that will be employed is to think of the high order numerical scheme as the source of a very accurate approximation to the exact solution, this will be referred to as the target. The actual solution provided by the total numerical method will then be the solution that best approximates this target in some norm while still satisfying a host of physical constraints. This is essentially the approach referred to as an Optimization Based Remap (OBR) by Bochev et. al. [3]. This framework has previously been successfully applied to an Eulerian Spectral Element Method [10].

## 2. Mathematical Preliminaries.

**2.1. The Scalar Advection Equation.** The basic problem considered in this work is the multi-dimensional scalar advection equation. Simple as it is, in many atmospheric models advection of arbitrarily many tracers is a major source of computational cost. A Semi-Lagrangian method would be ideal for this for two reasons. First it would allow much larger time steps than an Eulerian method. Secondly in Lagrangian methods the quantities that are actually advected are the points of the grid themselves. This saves having to advect every single tracer using an integration scheme as you would in a Eulerian scheme.

The equation of primary concern in this work is the advection equation for a passive tracer

$$\frac{\partial q}{\partial t} + \nu \cdot \nabla q = 0 \tag{2.1}$$

In the atmosphere passive tracer concentrations will be augmented by an equation describing the motion of the atmospheric fluid density.

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\nu \rho) = 0 \tag{2.2}$$

The quantity $\rho$ is a conserved quantity as is $(q\rho)$ for any passive tracer $q$.

**2.2. Specifics of the Method.** The method that will be described is very general with respect to geometry in the sense that it can work for any type of grid. However, boundary conditions could complicate things. Since the focus is on the atmosphere work has been restricted to a doubly periodic domain, or to the surface of a sphere.

The key idea of the Semi-Lagrangian method is to define $\tilde{\mathbf{x}}(t)$ such that

$$\frac{d\tilde{\mathbf{x}}}{dt} = \nu \tag{2.3}$$

then equation (2.1) says

$$\frac{dq}{dt}(\tilde{\mathbf{x}}(t)) = \frac{\partial q}{\partial t} + \frac{d\tilde{\mathbf{x}}}{dt} \cdot \nabla q = 0 \tag{2.4}$$

So along contours $\tilde{\mathbf{x}}(t)$ passive tracer $q$ is constant.

**2.3. High-Order Space discretization.** As mentioned in section 1 a Spectral Element spatial discretization will be implemented as it matches the spatial discretization of some established atmospheric codes. The Gauss-Legendre-Lobatto nodes are also ideal for this type of Semi-Lagrangian method for several other reasons.

- They display asymptotically optimal Lebesgue Constant growth [14]
- They have enough degrees of freedom to maintain inter-element continuity. This fact is key for the stability of this method. Trying to repeat the method that will be described below using Gauss quadrature nodes, for example, will not be stable because continuity is not enforced at cell boundaries and so large oscillations will be introduced when interpolating.
- They have an associated quadrature rule that is only slightly less accurate than a Gaussian quadrature rule (an important part of their popularity in Finite Element Methods).

Let us define $\tau$ as the collection of (quadrilateral) cells that tessellate our domain $\Omega$. Each cell $\tau_k$ will posses a mapping, call it $N_k(\mathbf{x})$, from $\tau_k$ to the reference quadrilateral. Assume that each cell posses an $(n+1)\times(n+1)$ subgrid of points that map to GLL nodes. Define

$$X = (\xi_i, \xi_j) \text{ where } \xi_i \text{ and } \xi_j \text{ are both members of the set of } n+1 \text{ GLL nodes}$$

$$\chi = \{(x_1, x_2) | \exists k \text{ s. t. } (x_1, x_2) \in \tau_k \text{ and } N_k((x_1, x_2)) = (\xi_i, \xi_j) \in X \tag{2.5}$$

$\chi$ will be the set of all nodes where data will be stored. On each cell we will have a set of points that maps to a tensor product GLL grid. So the natural basis functions to use in this situation are tensor products of Lagrange polynomials for the GLL nodes. So the basis functions will be $n$th order polynomials on the reference quadrilateral $\phi_{ij}((N_k(\mathbf{x}))$ such that

$$\phi_{ij}(\xi_l, \xi_k) = \delta_{li}\delta_{kj}$$

**2.4. Semi-Lagrangian Time-Stepping.** The Semi-Lagrangian scheme solves equation (1.1) by breaking it up into a series of ODEs of the form (2.3) coupled with an interpolation. The basic scheme is below:

*2.4.0.1.* **Semi-Lagrangian Scheme.**

1. $\forall \mathbf{x} \in \chi$ solve for $\tilde{\mathbf{x}}(t^n)$ subject to $\dot{\tilde{\mathbf{x}}} = \nu$ with $\tilde{\mathbf{x}}(t^{n+1}) = \mathbf{x}$
2. Then define $q(\mathbf{x}, t^{n+1}) = q(\tilde{\mathbf{x}}, t^n)$

This point $\tilde{\mathbf{x}}$ is generally referred to as the "trace back" in the MMOC literature. Essentially this is just using the high order spatial discretization to interpolate a function that is transported by a time-stepping scheme that approximates the advection equation. Since the GLL nodes posseses good quadrature properties this method is also an approximation of

$$(u(\mathbf{x}, t^{n+1}), v) = (u(\tilde{\mathbf{x}}, t^n), v) \forall v \in \mathbb{Q}^n \text{ where } \dot{\tilde{\mathbf{x}}} = \nu \text{ and } \tilde{\mathbf{x}}(\mathbf{t^{n+1}}) = \mathbf{x} \tag{2.6}$$

**2.4.1. Connection to Characteristic Galerkin Methods And Convergence Rates.** Say that the polynomial order of the spatial discretization used for the scheme in (2.4.0.1) is $k$. The scheme described in equation (2.4.0.1) approximately solves (2.6). This is also the variational form that is central to a class of methods known as Characteristic Galerkin Methods [11]. An interesting observation is that if it is assumed that these integrals are evaluated exactly then optimal FEM convergence

rates of $O(h^{k+1})$ can be proven [6] for this type of method. However in general these will not be evaluated exactly and so a lower convergence rate of $O(h^k)$ is actually observed [7]. The SLSEM is on the cusp between the Characteristic Galerkin method and the Semi-Lagrangian method because it uses approximately accurate quadrature rules to maintain a diagonal mass matrix and to approximately evaluate the projection integral on the right hand side of equation (2.6).

**2.4.2. A Stable Characteristic Galerkin Semi-Lagrangian Scheme.** As mentioned in section 1, in general hyperbolic equations allow discontinuous solutions. However the SL method outlined in (2.4.0.1) is not stable for discontinuous initial conditions. Because repeated orthogonal projection is, in general, more stable than repeated interpolation you may expect that enforcing the variational form (2.6) would provide a more stable scheme. However, an exact orthogonal projection on each cell (i.e. a decomposition in terms of Lagrange Polynomials) would result in a discontinuous-galerkin like scheme that would no longer enforce inter-cell continuity. Accurately performing the integrals involved in equation (2.6) when the solution is not even $C_0$ is very expensive. A compromise worth exploring is to still project onto the same piecewise Lagrange-polynomial basis corresponding to the GLL nodes, but to use different quadrature rules for each of the sides of the equation (2.6). On the left hand side it is ideal to still use GLL nodes to enforce a diagonal mass-matrix. On the right hand side a higher-order accurate quadrature rule can be used without greatly increasing the computational cost. Let us define $n_l$ and $n_r$ such that $n_l = k+1$ is the number of GLL quadrature nodes used for the left hand side integrals, and $n_r$ is the number of GLL quadrature nodes used for the right hand side integral.

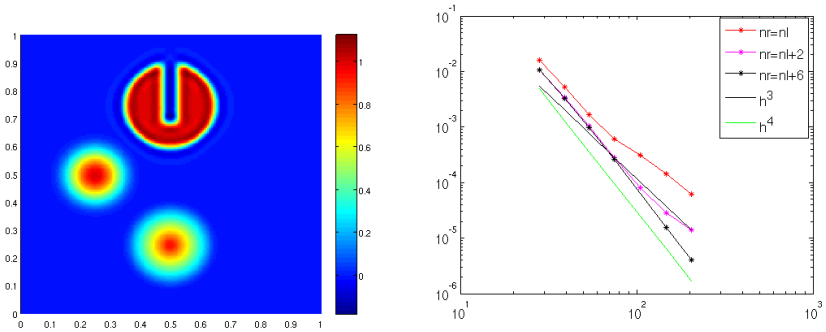
*2.4.2.1.* **Characteristic Galerkin Semi-Lagrangian Hybrid Scheme.**

1. Define $X_r = (\xi_i, \xi_j)$ where $\xi_i$ and $\xi_j$ are both in the size $n_r$ GLL nodes
2. $\forall \tau$ find all $\mathbf{x}_{ij}$ such that $N(\mathbf{x}_{ij}) \in X_r \times X_r$ solve for $\tilde{\mathbf{x}}_{ij}(t^n)$
   subject to $\dot{\tilde{\mathbf{x}}}_{ij} = \nu$ with $\tilde{\mathbf{x}}_{ij}(t^{n+1}) = \mathbf{x}_{ij}$
3. And find $\mathbf{x}_{lm}$ such that $N(\mathbf{x}_{lm}) \in X \times X$
4. $q(\mathbf{x}_{lm}, t^{n+1}) = \frac{1}{J(\mathbf{x}_{lm})\omega_l^{n_l}\omega_m^{n_l}} \sum_i^{n_r} \sum_j^{n_r} J(\mathbf{x}_{ij})\omega_i^{n_r}\omega_j^{n_r}\ell_l^{n_l}(x_i)\ell_m^{n_l}(x_j)q(\tilde{\mathbf{x}}_{ij}, t^n)$

In the scheme in (2.4.2.1) what is happening is that the basis functions $\ell_i(x)$ are being treated as orthogonal and essentially a function $q(\tilde{x}, t^n)$ is being approximated on each cell by the expression in equation (2.7) (the expression is slightly more complicated in reality because of interactions between cells but this is the basic idea)

$$q(\mathbf{x}, t^{n+1}) = \sum_i^{n_r} \sum_j^{n_r} \ell_i^{n_l}\ell_j^{n_l} \frac{\int_{-1}^{1}\int_{-1}^{1} J(\xi,\eta)\ell_i^{n_l}(\xi)\ell_j^{n_l}(\eta)q(\tilde{\mathbf{x}}(\xi,\eta), t^n)d\xi d\eta}{\int_{-1}^{1}\int_{-1}^{1} J(\xi,\eta)\ell_i(\xi)\ell_j(\eta)d\xi d\eta} \tag{2.7}$$

It can be shown that the one step accuracy for this approximation is actually $O(h^k)$. So this method should theoretically not give better accuracy than that described in (2.4.0.1). However the error from treating the GLL Lagrange polynomials as orthogonal is very small and it is not observed as you increase $n_r$ until $h$ is very small, and even then it is more accurate than the pure SL scheme. This can be seen in figure 2.1(b). In addition the example in figure 2.1(a) shows the results of using (2.4.2.1) on an example that the method (2.4.0.1) was not able to solve. The result displays oscillations as you would expect for the polynomial approximation of a discontinuous function, but it is certainly not blowing up. This increased stability is the main reason this method may be useful in some situations. The accuracy increase shown in figure 2.1(b) is not as important because it takes a much more expensive quadrature rule to realize significant improvments.




(a) A discontinuous initial condition advected for 50 rotations using equation (2.4.2.1) with $n_r = n_l + 1$

(b) The $L_2$ norm convergence rate as $n_r$ is increased leaving $n_l$ fixed

Fig. 2.1: The effect of Quasi-Monotone limiters on Convergence Rate

**3. Conservation.** In general numerical algorithms that preserve physical conserved quantities tend to be preferred over those that do not because they are observed to produce more accurate results, especially for long time integration. However looking at the variational form in equation (2.6) it is clear that conservation is not guaranteed.

$$(q(\tilde{\mathbf{x}}, t^n), 1) \neq (q(\mathbf{x}, t^n), 1) \tag{3.1}$$

This has been cited as a major problem with the MMOC and the Characteristic Galerkin methods. In general to have a conservative method in this Lagrangian framework requires tracking cells instead of points. The Characteristics-Mixed [2] method and the ELLAM[12] methods both achieve approximate local conservation this way. This still presents problems because there will be volume errors in tracking cells, motivating methods that seek to correct these volume errors[1].

There has been an attempt to modify the MMOC to maintain conservation[5]. That method, known as, MMOCAA (Modified Method of Characteristics with Adjusted Advection) allows a range of possible values for the "traceback" points (defining some perturbation around the calculated value). Conservation is achieved by solving for values in this range.

The following section will illustrate a possible way to modify the SLSEM algorithm so that it is theoretically conservative, however it will be apparent that it involves evaluating certain integrals that would be very difficult to implement accurately. Thus for this work what is actually implemented is a numerical optimization scheme that allows the enforcement of conservation in a way that can be numerically verified not to affect the convergence rate.

**3.1. A Conservative Scheme.** Starting with the variational form of the conservative advection equation...

$$\int_t \int_\Omega (\rho_t + \nabla \cdot (\nu\rho)) v d\Omega = 0$$

$$\int_\Omega (\rho^{n+1} v^{n+1} - \rho^n v^n) d\Omega - \int_t \int_\Omega \rho v_t d\Omega dt + \int_t \int_\Omega (\rho\nu \cdot \nabla(v)) d\Omega dt$$

To simplify things look for $v$ such that

$$v_t + \nu \cdot \nabla(v) = 0$$

then the equation simplifies to

$$(\rho^{n+1}, v^{n+1}) = (\rho^n, v^n)$$

Assume the underlying grid is Cartesian, indexed by an $x$ and a $y$ index. $v$ is chosen such that $v^{n+1}_{ijlk} = l^{ij}_l(x) l^{ij}_k(y)$ (a GLL node Lagrange polynomial on the $(i,j)$ cell) then this is essentially tracing cells back in time through the basis functions. In addition

$$(\rho^{n+1}, \sum_{l,k} v^{n+1}_{ijkl}) = (\rho^{n+1}, 1^{n+1}_{ij})$$

$$(\rho^{n}, \sum_{l,k} v^{n}_{ijkl}) = (\rho^{n}, 1^{n}_{ij})$$

and so since the method enforces

$$(\rho^n, v^n_{ijkl}) = (\rho^{n+1}, v^{n+1}_{ijkl})$$

it is then true that

$$(\rho^n, 1^n_{ij}) = (\rho^{n+1}, 1^{n+1}_{ij})$$

So, this preserves some notion of local conservation. However trying to do this it would be very difficult to correctly integrate $(\rho^n, v^n)$ because of the difficulty evaluating cell evolution accurately.

**3.2. Optimization Based Conservation.** The method actually chosen to enforce conservation is to treat the physical constraint as a linear constraint in a quadratic programming problem. Essentially the QP problem is to find the L2 minimal solution, compared to the result produced by interpolation at every timestep, that is conservative. This is all done discretely at the GLL nodes. The QP with a linear constraint is solved using a secant method that converges very quickly.

*3.2.0.2.* **Optimization Based Conservative Scheme.**

1. For each $\mathbf{x} \in \chi$ solve for $\tilde{\mathbf{x}}(t^n)$ subject to $\dot{\tilde{\mathbf{x}}} = \nu$ with $\tilde{\mathbf{x}}(t^{n+1}) = \mathbf{x}$
2. $q^T(\mathbf{x}, t^{n+1}) = q(\tilde{\mathbf{x}}, t^n)$
3. Then find $q$ that minimizes $\|q - q^T\|_2^2$ such that $\int_\Omega q d\Omega = \int_\Omega q^n d\Omega$

Since the GLL nodes are an accurate quadrature rule for polynomials up to order $2n-1$ the L2 minimization and conservation constraint can be discretized and enforced very efficiently using only function values at the GLL nodes and their corresponding quadrature weights.

$$\int_\Omega (q(\mathbf{x}) - q^T(\mathbf{x}))^2 d\Omega = \sum_{k=0}^{|\tau|} \int_{\tau_k} (\sum_{i=0}^{n}\sum_{j=0}^{n} q^k_{ij}\phi_{ij}(\xi) - \sum_{i=0}^{n}\sum_{j=0}^{n} q^{Tk}_{ij}\phi_{ij}(\xi))^2 d\tau_k$$

This integral can be approximated using the GLL quadrature rule.

$$\approx \sum_{k=0}^{|\tau|}\sum_{i=0}^{n}\sum_{j=0}^{n} J^k_{ij} w_i w_i \left((q^k_{ij})^2 - 2(q^k_{ij})(q^{Tk}_{ij}) + (q^{Tk}_{ij})^2\right)$$

Where $J_{ij}^k$ is the Jacobian of the transformation $N^k$ at the $(i,j)$ node of the $k$ cell. This can be written as

$$\int_\Omega (q(\mathbf{x}) - q^T(\mathbf{x}))^2 d\Omega \approx \mathbf{q}^* D\mathbf{q} - 2c^*\mathbf{q} + \mathbf{q^T}^* D\mathbf{q^T} \tag{3.2}$$

where $D$ is a diagonal matrix containing $J_{ij}^k w_i w_j$ and $\mathbf{q}^*$ is corresponds to the transpose of a vector (used to avoid confusion with $q^T$ the target). So minimizing $\int_\Omega (q(\mathbf{x}) - q^T(\mathbf{x}))^2 d\Omega$ is approximately equivalent to minimizing

$$\mathbf{q}^* D\mathbf{q} - 2\mathbf{c}^*\mathbf{q} = 2(\frac{1}{2}\mathbf{q}^* D\mathbf{q} + (-\mathbf{c})^*\mathbf{q}) \tag{3.3}$$

This is a standard quadratic programming problem with a diagonal matrix. The conservation constraint can also be similarly discretized, although in this case for non curvilinear cells the discretized constraint will be exactly equivalent to the continuous constraint.

**4. Optimization Based Limiters for Passive Tracers.** When advecting discontinuous profiles it is important to prevent the appearance of nonphysical oscillations. In Eulerian Finite Volume Methods this is accomplished with a class of methods that preserve the "monotonicity" properties of a solution. This boils down to preventing the solution from developing new minima and maxima. The theory of these monotonicity preserving limiters is very well understood and developed for low order Finite Volume methods. However, for higher order methods it is unclear how to preserve monotonicity without harming accuracy in smooth regions, or even if that is possible. An idea would be to present the limiter as a constrained optimization problem trying to find the closest function, in some norm, to the output of a given numerical method while enforcing that there are no new maxima or minima created in any given region. This method of finding the "closest" solution to the output of a numerical method is a key part of the optimization-based-remap framework [3].

For this Semi-Lagrangian type method there are two ways one could think of imposing monotonicity. First of all the high-order interpolation used is not monotonic, so one can enforce monotonicity on this interpolation method. This is a very low computational cost way of enforcing monotonicity as all bounds are available locally. However, this does not necessarily guarantee that the "reconstruction" on the new cell is monotonic. A way of achieving that will be discussed below. The type of maximum and minimum bounds introduced are not guaranteed to enforce exact monotonicity and so we will call this family of methods Quasi-Monotone Limiters [10]. Formulated as a constrained QP these limiters will look like:

*4.0.0.3.* **Quasi-Monotone Limiter.**

1. For each $\mathbf{x} \in \chi$ solve for $\tilde{\mathbf{x}}(t^n)$ subject to $\dot{\tilde{\mathbf{x}}} = \nu$ with $\tilde{\mathbf{x}}(t^{n+1}) = \mathbf{x}$
2. $q^T(\mathbf{x}, t^{n+1}) = q(\tilde{\mathbf{x}}, t^n)$
3. Somehow determine upper and lower bounds for $q$ $\underline{q}$ and $\bar{q}$
4. Find $q$ that minimizes $\|q - q^T\|_2^2$ such that $\int_\Omega q d\Omega = \int_\Omega q^n d\Omega$ and $\underline{q} \leq q \leq \bar{q}$

**4.1. Quasi-Monotone Interpolation Limiter.** Enforcing constraints on the interpolation operator itself is simple. It requires knowing what values would constitute a new maxima or minima on a given cell of the Lagrangian grid. For the back tracing SL method this can be done just by looking at the max and min values on the GLL nodes in that cell. This type of limiting will filter out the Runge phenomena inherent to attempting to interpolate a function of limited continuity.

*4.1.0.4.* **Quasi-Monotone Interpolation.**

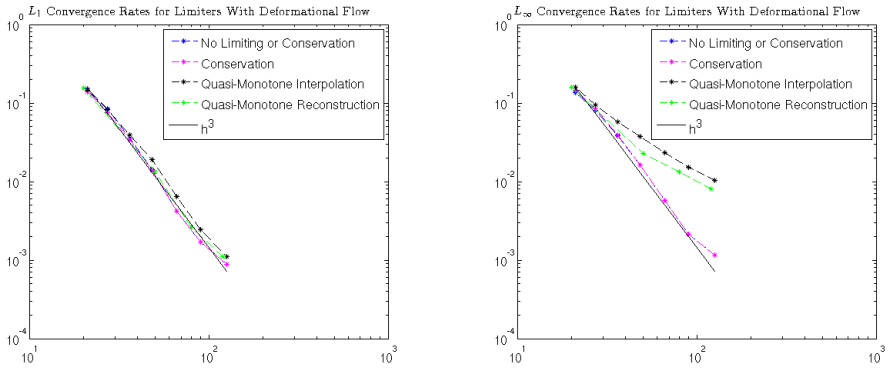
1. For each $\mathbf{x} \in \chi$ solve for $\tilde{\mathbf{x}}(t^n)$ subject to $\dot{\tilde{\mathbf{x}}} = \nu$ with $\tilde{\mathbf{x}}(t^{n+1}) = \mathbf{x}$
2. find $k$ such that $\tilde{\mathbf{x}}(t^n) \in \tau_k$
3. define $\chi_k = \mathbf{x} | \mathbf{x} \in \chi$ and $\mathbf{x} \in \tau_k$.
4. $\underline{q}(\mathbf{x}) = \min_{\mathbf{x} \in \chi_k} q(\mathbf{x}, t^n)$ and $\bar{q}(\mathbf{x}) = \max_{\mathbf{x} \in \chi_k} q(\mathbf{x}, t^n)$

**4.2. Quasi-Monotone Reconstruction.** In the Quasi-Monotone Interpolation limiter each point on the Eulerian grid gets its own individual bounds that may differ within a cell. To introduce something that looks closer at limiting the overall reconstruction on the Eulerian cell would require thinking about moving cells forward in time to keep track of which cells of the Lagrangian mesh intersect each cell the Eulerian mesh at a future time-step. In fact for this to really work well the Lagrangian cells would have to be split up into constant value subcells centered around the various GLL nodes. This would be too expensive to do accurately and so an alternative method that will approximate this is proposed. The key idea of this method is to track GLL nodes forward in time only keeping track of the cells they land in. For each cell define $(\bar{q})_k$ and $(\underline{q})_k$ by maximizing and minimizing over all of the nodes that land in the $\tau_k$ or its direct neighbors (to account for nodes whose containing "subcells" actually intersect the $(k)$ cell without the node actually landing in that cell itself). Note that as the CFL number increases a larger Halo may be needed to be included around each cell to account for the more highly distorted Lagrangian subcells.

*4.2.0.5.* **Quasi-Monotone Reconstruction.**

1. For each $\mathbf{x} \in \chi$ solve for $\tilde{\mathbf{x}}(t^{n+1})$ subject to $\dot{\tilde{\mathbf{x}}} = \nu$ with $\tilde{\mathbf{x}}(t^n) = \mathbf{x}$
2. find $k$ such that $\tilde{\mathbf{x}}(t^{n+1}) \in \tau_k$
3. define, for each $\tau_k$, $\chi_k = \mathbf{x} | \mathbf{x} \in \chi$ and $\tilde{\mathbf{x}}(t^{n+1}) \in \tau_k$
4. find $k_1$ such that $\mathbf{x} \in \tau_{k_1}$
5. $\underline{q}(\mathbf{x}) = \min_{\mathbf{x} \in \chi_{k_1}} q(\mathbf{x}, t^n)$ and $\bar{q}(\mathbf{x}) = \max_{\mathbf{x} \in \chi_{k_1}} q(\mathbf{x}, t^n)$

**4.3. Numerical Results.** Figures (4.1(a)) and (4.1(b)) illustrate convergence rates for two gaussian hills advected by a deformational but non-divergent flow field. Shown in each figure are convergence rates for the basic Semi-Lagrangian scheme, and schemes modified to enforce conservation and quasi-monotonicity. The conservation enforcement alone does not affect the convergence rate in the infinite or L1 norms. However the Quasi-Monotonicity limiters both affect the infinite norm convergence quite a bit. The optimization based strategy, however, seems to still allow high-order convergence in the $L_1$ norm indicating that large adjustments need to be made by the optimization scheme at only very few nodes. Both limiting schemes perform well, and the difference between them is not very great. This holds true when advecting discontinuous examples and so I recommend using the Quasi-Monotone Interpolation scheme as it is cheaper computationally.




(a) The $L_1$ norm convergence rate (b) The $L_\infty$ norm convergence rate

Fig. 4.1: The effect of Quasi-Monotone limiters on Convergence Rate

**5. Divergent Flows.** Up until now the focus has been on tracer quantities and how to deal with them. However everything said transfers directly to conserved quantities if the velocity field is non-divergent. Conserved quantities in divergent flows, however, must be treated somewhat differently. In this section the fluid density in a divergent flow field will be considered. The density satisfies the following equation

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\nu \rho) = 0 \tag{5.1}$$

Now in the divergent flow case $\nabla \cdot \nu = 0$ and this can be rewritten as the tracer advection equation. However in the non-divergent flow case this equation is not as trivially handled by the Method of Characteristics. Isolating the material derivative on the left hand side gives

$$\frac{\partial \rho}{\partial t} + \nu \cdot \nabla(\rho) = -\rho \nabla \cdot \nu \tag{5.2}$$

Thus along characteristics the density is governed by the ode

$$\frac{d\rho}{dt} = -\rho \nabla \cdot \nu \tag{5.3}$$

This means that the base numerical scheme must be amended

*5.0.0.6.* **SLSEM for a Conserved Quantity in a Divergent Flow.**

1. $\forall \mathbf{x} \in \chi$ solve for $\tilde{\mathbf{x}}(t^n)$ subject to $\dot{\tilde{\mathbf{x}}} = \nu$ with $\tilde{\mathbf{x}}(\mathbf{t^{n+1}}) = \mathbf{x}$
2. Then set $\rho(\mathbf{x}, t^{n+1}) = \rho(\tilde{\mathbf{x}}, t^n) \exp(-\int_{t^n}^{t^{n+1}} \nabla \cdot \nu dt)$

This method can be performed arbitrarily accurately by using high-order time stepping methods to perform the integral in the exponential factor.

**5.1. Limiting Density In the Divergent Case.** Limiting is not as straightforward in the Divergent flow case. Since the density is not constant along tracers it is difficult to say precisely what bounds should be enforced in order to achieve monotonicity. This is not much of a problem in the smooth density case, but it is possible for conserved quantities to have non-smooth profiles and more generally from

a mathematical viewpoint it is good to be able to handle equations in nonconservative or conservative form. It would also be nice to be able to solve equations with simple source terms of any kind (even if not due to a divergent flow field) while preventing the development of non-physical oscillations. One way to enforce monotonicity on densities is to look at evolving the mesh forward. Then, using the Reynolds transport theorem

$$\frac{d}{dt}\int_{\Omega(t)} \rho d\Omega(t) = 0 \tag{5.4}$$

it is possible to bound physically possible values of the density based on computing the change in cell volumes. In the high order case tracking cells forward is difficult and expensive. However notice that if one assumes that the sources are finite (i.e. in the divergent flow case that the velocity is differentiable) then it can be expected that any $q$ such that $L(q) = 0$ and $q(\mathbf{x}, 0) = 1.0$ will remain smooth as this is a linear problem. Also, if

$$\frac{\partial q}{\partial t} + \nu \cdot \nabla(q) = q\nabla \cdot \nu \tag{5.5}$$

$$\frac{\partial(\rho q)}{\partial t} + \nu \cdot \nabla(\rho q) = 0 \tag{5.6}$$

and this quantity $\rho q$ is constant along tracers. So if a very accurate solution of equation (5.5) can be obtained then that solution, coupled with the solution to equation (5.6), can be used to obtain $\rho$. This also gives a quantity $(\rho q)$ for which bounds can be defined as it will be constant along characteristics.

There is a great deal of freedom in choosing $q$, but there are two straightforward methods. Either define $q$ to initially be a constant function and then evolve it in time with $\rho$ or define a new $q$ at each timestep. The former method will be referred to as Evolving a Fictitious Density and the latter method will be referred to Evolving a Fictitious Density with Restarting. The latter method ends up being equivalent to just getting bounds from $\rho$ at a previous timestep and evolving those bounds along the characteristic as if they were subject to the same ODE as the density itself. This is a very efficient method. However, the former method gives better results in general for the linear problems considered with weakly divergent fields.

**5.1.1. Limiting with A Fictitious Density With Restarting.** This method of limiting requires solving, at each time-step

$$\frac{\partial \rho q}{\partial t} + \nu \cdot \nabla(\rho q) = 0 \text{ such that } \rho(t^n)q(t^n) = \rho(t^n)$$

As $Q = \rho q$ is a tracer it is easy to obtain its bounds $\bar{Q}^n$ and $\underline{Q}^n$. Also since $q(t^n) = 1$, $q$ can be evolved analytically (although integrals along characteristics will still be approximated).

$$q(\mathbf{x}, t^{n+1}) = \exp(\int_{t^n}^{t^{n+1}} \nabla \cdot \nu dt)$$

*5.1.1.1.* **Fictitious Density With Restarting.**

1. For each $\mathbf{x} \in \chi$ solve for $\tilde{\mathbf{x}}(t^n)$ subject to $\dot{\tilde{\mathbf{x}}} = \nu$ with $\tilde{\mathbf{x}}(t^{n+1}) = \mathbf{x}$
2. $(Q)^T(\mathbf{x}, t^{n+1}) = Q(\tilde{\mathbf{x}}, t^n)$
3. $q(\mathbf{x}, t^{n+1}) = \exp(\int_{t^n}^{t^{n+1}} \nabla \cdot \nu dt)$
4. Then find $\rho$ that minimizes $\|\rho - \frac{Q}{q}^T\|_2^2$ such that $\int_\Omega \rho d\Omega = \int_\Omega \rho^n d\Omega$ and $\underline{Q} \leq \rho q \leq \bar{Q}$

**5.1.2. Limiting with an Evolving Fictitious Density.** Instead of restarting every
timestep a simple idea is just to evolve $q$ as a separate quantity. As long as the velocity field is smooth and not too strongly divergent there should be very few problems with spurious oscillations in evolving $q$ as there is freedom to pick its initial condition. For example this should be fine for atmospheric flows where usually the divergence is not very strong. If there is ever a situation where $q$ becomes under-resolved or seems to be experiencing oscillations itself then it is possible just to restart $q$. The advantage of not restarting $q$, though, is that $\rho q$, which is a tracer , will be the stored quantity. Storing the tracer avoids any sort of buildup of oscillations that could come from introducing maxima or minima at intermediate steps. It will be shown that this does give better results for weakly divergent flow fields.

Assuming there is no need to restart the method is:

$$\frac{\partial \rho q}{\partial t} + \nu \cdot \nabla(\rho q) = 0 \text{ such that } \rho(t^0)q(t^0) = \rho(t^0)$$

As $Q = \rho q$ is a tracer we know how to obtain bounds $\bar{Q}^n$ and $\underline{Q}^n$.

*5.1.2.1.* **Evolving A Fictitious Density.**

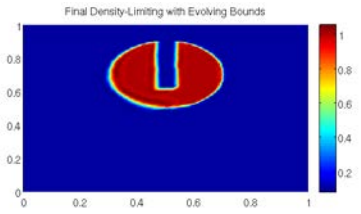
1. For each $\mathbf{x} \in \chi$ solve for $\tilde{\mathbf{x}}(t^n)$ subject to $\dot{\tilde{\mathbf{x}}} = \nu$ with $\tilde{\mathbf{x}}(t^{n+1}) = \mathbf{x}$
2. $(Q)^T(\mathbf{x}, t^{n+1}) = Q(\tilde{\mathbf{x}}, t^n)$
3. $q(\mathbf{x}, t^{n+1}) = q(\tilde{\mathbf{x}}, t^n) \exp(\int_{t^n}^{t^{n+1}} \nabla \cdot \nu dt)$
4. Then find $\rho$ that minimizes $\|\rho - \frac{(Q)}{q}^T\|_2^2$ such that $\int_\Omega \rho d\Omega = \int_\Omega \rho^n d\Omega$ and $\underline{Q} \leq \rho q \leq \bar{Q}$

**5.2. Numerical Results.**

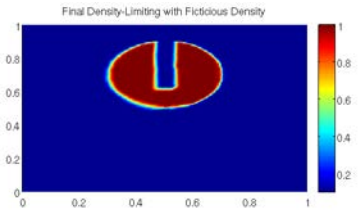
**5.2.1. A Divergent Flow Example.** The numerical example explored in this section has the following velocity field ($u$ is the $x$ velocity and $v$ is the y velocity).

$$v = \frac{1}{2}\sin(2\pi x)\cos(\pi(y-0.5))^3\cos(\frac{t\pi}{T}) \text{ and } u = -\sin(\pi x)^2\sin(2\pi(y-0.5))\cos(\pi(y-0.5))^2\cos(\frac{t\pi}{T})$$

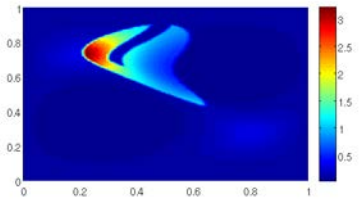
This flow field is divergent,and is designed to take densities to some intermediate state and then return to the initial state when $t = T$. Figure 5.1(c) shows the intermediate state. Figure 5.1(a) shows the final state with limiting done using the Fictitious Density with Restarting. Figure 5.1(b) shows the final state with limiting done using a Fictitious Density Without Restarting. Restarting the fictitious density requires the storing of intermediate states where new maxima and minima have been introduced, and this allows the formation of artificial oscillations. This is because in some cells there will be new extrema values and if those cells also have artificial oscillations, the oscillations may not be flagged as outside the range of physical values.




(a) The density at the final time using a fictitious density that is restarted at each timestep

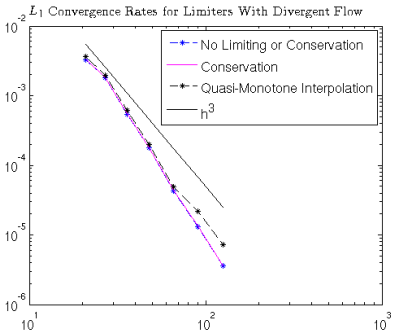



(b) The density at the final time using a fictitious density that is not restarted

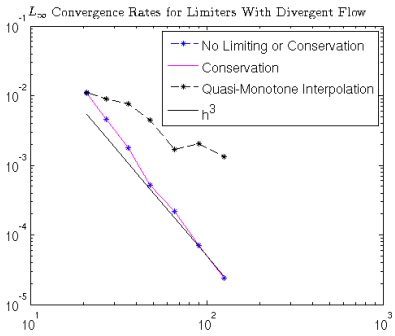


(c) The density in its intermediate state created by a divergent flow.

Fig. 5.1: Fictitious Density Methods to Limit a Divergent Flow with a Discontinuous Density




(a) The $L_1$ norm convergence rate




(b) The $L_\infty$ norm convergence rate

Fig. 5.2: The effect of Quasi-Monotone limiters on Convergence Rate for divergent flows

**5.2.2. Limiters effect on convergence.** Figures 5.2(a) and 5.2(b) show that the quasi-monotone limiters have the same pattern of effects on smooth density profiles for divergent flow field as they do for non-divergent flow fields (not surprising since the only thing that changes is that the density is no-longer constant).

**6. Conclusions.** A Spectral Element Method with Semi-Lagrangian time stepping has been introduced for the scalar advection equation. It was shown, computationally, that often higher accuracy and greater stability can be achieved by modifying

the SL method to more accurately compute the integrals in a Characteristic Galerkin variational form. The SLSEM was coupled to an Optimization Based Remap framework that allowed the development of very efficient Optimization Based Limiters as well as optimization techniques to enforce conservation. Work was done to extend this scheme and the optimization based limiters to handle all possible flow fields that would be encountered in atmospheric simulation. In particular the scheme and its limiters were developed for the tracers under the effect of a zeroth order source term. This is motivated by the fact that the fluid density in a divergent flow is a special case of this. The density, in this situation, does not display monotonicity but a "fictitious density" was introduced that allowed the definition of a quantity that does. The optimization based limiters were shown to generally not affect the $L_1$ norm convergence rate, even though they slow the $L_\infty$ convergence rate quite a bit.

## REFERENCES


[1] T. Arbogast and C.-S. Huang, *A fully mass and volume conserving implementation of a characteristic method for transport problems*, SIAM Journal on Scientific Computing, 28 (2006), pp. 2001–2022.

[2] T. Arbogast and M. F. Wheeler, *A characteristics-mixed finite element method for advection-dominated transport problems*, SIAM Journal on Numerical analysis, 32 (1995), pp. 404–424.

[3] P. Bochev, D. Ridzal, and K. Peterson, *Optimization-based remap and transport: A divide and conquer strategy for feature-preserving discretizations*, Journal of Computational Physics, 257 (2014), pp. 1113–1139.

[4] J. Douglas, Jr and T. F. Russell, *Numerical methods for convection-dominated diffusion problems based on combining the method of characteristics with finite element or finite difference procedures*, SIAM Journal on Numerical Analysis, 19 (1982), pp. 871–885.

[5] J. Douglas Jr, F. Furtado, and F. Pereira, *On the numerical simulation of waterflooding of heterogeneous petroleum reservoirs*, Computational Geosciences, 1 (1997), pp. 155–190.

[6] R. E. Ewing, T. F. Russell, and M. F. Wheeler, *Convergence analysis of an approximation of miscible displacement in porous media by mixed finite elements and a modified method of characteristics*, Computer Methods in Applied Mechanics and Engineering, 47 (1984), pp. 73–92.

[7] M. Falcone and R. Ferretti, *Convergence analysis for a class of high-order semi-lagrangian advection schemes*, SIAM Journal on Numerical Analysis, 35 (1998), pp. 909–940.

[8] F. Giraldo, J. Perot, and P. Fischer, *A spectral element semi-lagrangian (sesl) method for the spherical shallow water equations*, Journal of Computational Physics, 190 (2003), pp. 623–650.

[9] F. X. Giraldo, *The lagrange–galerkin spectral element method on unstructured quadrilateral grids*, Journal of Computational Physics, 147 (1998), pp. 114–146.

[10] O. Guba, M. Taylor, and A. St-Cyr, *Optimization-based limiters for the spectral element method*, Journal of Computational Physics, 267 (2014), pp. 176–195.

[11] M. Kaazempur-Mofrad and C. Ethier, *An efficient characteristic galerkin scheme for the advection equation in 3-d*, Computer methods in applied mechanics and engineering, 191 (2002), pp. 5345–5363.

[12] T. F. Russell and M. A. Celia, *An overview of research on eulerian–lagrangian localized adjoint methods (ellam)*, Advances in Water Resources, 25 (2002), pp. 1215–1231.

[13] M. Taylor, J. Edwards, S. Thomas, and R. Nair, *A mass and energy conserving spectral element atmospheric dynamical core on the cubed-sphere grid*, in Journal of Physics: Conference Series, vol. 78, IOP Publishing, 2007, p. 012074.

[14] L. N. Trefethen, *Approximation theory and approximation practice*, Siam, 2013.

[15] D. Xiu and G. E. Karniadakis, *A semi-lagrangian high-order method for navier–stokes equations*, Journal of computational physics, 172 (2001), pp. 658–684.

# MULTILEVEL METHODS FOR THE STOCHATIC GALERKIN METHOD FOR PDES WITH RANDOM INPUT DATA

SARAH V. OSBORN* AND ERIC T. PHIPPS†

**Abstract.** In this paper, we discuss solving partial differential equations (PDEs) with random input data using the stochastic Galerkin method. This method can often be computationally very demanding. We consider a multilevel solution strategy for the stochastic Galerkin method that employs hierarchies of spatial and stochastic approximations in hopes to diminish some of the computational burden. Numerical results are presented that compare this method to the traditional, single level method.

**1. Introduction.** In many applications, the values of parameters are often not known exactly and an important task is propagating this uncertainty in the inputs through a simulation. In particular, we are interested in finding the solution to partial differential equations (PDEs) with random coefficients. Several methods are used to solve these types of problems including sampling methods like Monte Carlo [12, 6]. While these types of methods are effective for many contexts, if higher accuracy is necessary the large number of samples necessary makes some computations infeasible. Other methods like stochastic collocation [17, 1, 13, 14] and stochastic Galerkin [8, 9, 18] have been explored that achieve faster convergence rates, but as the dimension of the stochastic space increases the computational cost increases greatly. We are interested in exploring a multilevel based solution strategy to alleviate some of this computational effort in the stochastic Galerkin method. This type of multilevel approach has been applied successfully for Monte Carlo methods, for instance in [3, 5], as well as for the stochastic collocation method in [10]. The idea is to use a hierarchical sequence of spatial approximations to the PDE as well as different stochastic approximations. We seek to maintain the overall accuracy of the solution while decreasing the computational cost of the calculation.

We start by describing the mathematical model and stochastic Galerkin formulation for the model problem in Section 2. In Section 3, the formulation of the multilevel method is given. The solution strategy and relevant linear algebra is discussed in Section 4. In Section 5, numerical results are provided that illustrate the performance of the proposed multilevel method compared against the standard, single-level stochastic Galerkin method.

**2. Problem Formulation.**

**2.1. Model Problem.** Let $D$ be an open subset of $\mathbb{R}^n$ and let $(\Omega, \Sigma, P)$ be a complete probability space, where $\Omega$ is the sample space, $\Sigma$ is the $\sigma$-algebra generated by $\Omega$ and $P : \Sigma \to [0, 1]$ is the probability measure. Given the random field $a(x, \omega) : \Sigma \times \overline{D} \to \mathbb{R}$ and function $f(x) \in L^2(D)$, stochastic steady-state diffusion equation with homogeneous Dirichlet boundary conditions is given by

$$-\nabla \cdot (a(x, \omega)\nabla u(x, \omega)) = f(x) \text{ in } D \times \Omega \tag{2.1}$$

$$u(x, \omega) = 0 \text{ on } \partial D \times \Omega. \tag{2.2}$$

We are interested in finding a random function $u(x, \omega) : \overline{D} \times \Omega \to \mathbb{R}$ which satisfies (2.1). Note that the divergence and gradient operators are considered to act on the

*Texas Tech University, sarah.osborn@ttu.edu
†Sandia National Laboratories, etphipp@sandia.gov

spatial variables $x \in D$ only. It is assumed that the random field $a$ can be expressed in terms of a finite number $M$ of random variables, denoted $\xi_i(\omega)$.

**2.2. Weak Formulation.** Let $H_0^2(D)$ be the subspace of the Sobolev space $H^1(D)$ that vanishes on the boundary $\partial D$ with norm $\|u\|_{H_0^1(D)} = [\int_D |\nabla u|^2 dx]^{\frac{1}{2}}$.

The equivalent variational formulation of (2.1) is given by the following: find $u \in H_0^1(D) \otimes L_2(\Omega) = V$ such that

$$b(u, v) = l(v), \ \ \forall v \in H_0^1(D) \otimes L_2(\Omega), \tag{2.3}$$

where

$$b(u, v) = \int_\Omega \int_D a(x, \omega) \nabla u \cdot \nabla v \, dx \, dP \tag{2.4}$$

$$l(v) = \int_\Omega \int_D f v \, dx \, dP. \tag{2.5}$$

**2.3. Discretization.** Following the stochastic Galerkin method [8, 9, 18], the solution to (2.3) is sought discretizing both the spatial and stochastic parts of the problem. Then the solution is sought in the finite-dimensional subspace defined as the tensor product of a spatial finite element basis and a generalized polynomial chaos expansion of a specified order denoted $V_N^n \subset V$ where $N$ and $n$ are discretization parameters for the stochastic and spatial dimension, respectively.

First, the PDE coefficients and the stochastic solution are discretized. This will be accomplished here by using a truncated series expansion which will separate the spatial variable $x$ from the stochastic variable $\omega$. It is assumed here that the random field is uniformly distributed and a truncated Karhunen-Loéve (KL) expansion is used to approximate $a(x, \omega)$. Assuming its covariance function $C(x_1, x_2)$ is known, then $a$ has the truncated KL expansion

$$a(x, \omega) \approx \sum_{i=0}^{M} a_i(x) \xi_i(\omega), \tag{2.6}$$

where $\xi_i(\omega)$, $i > 0$ are identically distributed, uncorrelated random variables with $\xi_0 = 1$. The mean of the random field is $a_0$ and $a_i(x) = \sqrt{\lambda_i} v_i(x)$ where $(\lambda_i, v_i(x))_{i \geq 1}$ are the solutions of the integral equation

$$\int_D C(x_1, x_2) v_i(x_2) dx_2 = \lambda_i v_i(x_1). \tag{2.7}$$

Using a generalized polynomial chaos (gPC) approximation, the solution $u$ can be written as

$$u(x, \xi) \approx \sum_{i=0}^{P} u_i(x) \psi_i(\xi)$$

where the basis functions $\{\psi_i(\xi)\}_{i=0}^P$ form an orthogonal basis of multidimensional polynomials of maximum total order $N$ in random variables $\xi_i$. The total dimension of the space spanned by $\{\psi_i\}$ is given by

$$P + 1 \equiv \frac{(M + N)!}{M! N!}.$$

As we assume a uniform probability distribution for the random variables, the Legendre polynomial basis is used.

For the spatial discretization, the finite element method is used where $\{\phi_j(x)\}_{j=1}^{N_{dof}}$ are suitable finite element basis functions of $H_0^1(D)$.

Then the discrete solution $u_N^n$ where $n$ is the number of nodes in one spatial dimension can be written as

$$u_N^n = \sum_{i=0}^{P} \sum_{j=1}^{N_{dof}} u_{ij} \phi_j(x) \psi_i(\xi). \tag{2.8}$$

Then, the Galerkin projection is applied in the tensor product space $V_N^n$ which yields a linear system of the form

$$A\mathbf{u} = \mathbf{f}$$

where $A \in \mathbb{R}^{(P+1)N_{dof} \times (P+1)N_{dof}}$, $u \in \mathbb{R}^{(P+1)N_{dof}}$, and $f \in \mathbb{R}^{(P+1)N_{dof}}$.

**3. Multilevel Formulation.** The challenge at hand is now solving the linear system of equations with $(P+1)N_{dof}$ degrees of freedom. In general, a fine mesh is used to obtain good spatial accuracy and a high order stochastic approximation is used to achieve good stochastic accuracy which overall leads to good overall accuracy. For a fixed number of random variables, if the order of the polynomial chaos expansion is increased then the computational cost dramatically increases. In order to alleviate some of this computationally demanding work, a multilevel solution strategy is now proposed. Let $\{u^{n_k}\}_{k \in \mathbb{N}}$ denote sequences of spatial approximations to the solution $u$. Then, for any $K \in \mathbb{N}$, we have the identity

$$u^{n_k} = \sum_{k=0}^{K} (u^{n_k} - u^{n_{k-1}}),$$

where we let $u^{n_{-1}} = 0$.

Based on bounds on the stochastic approximation, it can be shown that as $k \to \infty$ less accurate stochastic approximations are necessary in order to estimate the quantity $(u^{n_k} - u^{n_{k-1}})$ to a given accuracy. This means that for finer spatial discretizations lower order polynomial chaos expansions can be used to approximate the solution. Thus, our multilevel solution strategy can be written as

$$\hat{u}_N^{n_K} \approx \sum_{k=0}^{K} (u_{p_k}^{n_k} - u_{p_k}^{n_{k-1}}),$$

where $p_k$ are the polynomial orders used in the polynomial chaos expansion where $0 < p_K \leq p_{K-1} \leq \cdots \leq p_0 \leq N$.

**4. Solution Strategy.** Using the multilevel solution strategy, a hierarchy of linear systems with different spatial and stochastic discretizations now must be solved. Some consideration should be given to the structure of the matrices that arise from the stochastic Galerkin discretization.

**4.1. Matrix Structure.** After the stochastic Galerkin discretization, the resulting block linear system consists of $(P+1) \times (P+1)$ coupled blocks where the size

of each block is determined by the spatial discretization. The system can be written using Kronecker-product structure as

$$A = \sum_{i=0}^{P} G_i \otimes K_i,$$

where the matrices $G_i$ correspond to the stochastic discretization and each matrix $K_i$ is a deterministic finite element stiffness matrix. This choice of layout is made out of convenience to allow the reuse of deterministic solver data structures as well as preconditioners.

For certain problems because of some computational considerations, inverting the layout of the system is effective where the stochastic degrees of freedom corresponding to a given spatial node are grouped together. Using the Kronecker-product notation, this equivalent system can be written as

$$A = \sum_{i=0}^{P} K_i \otimes G_i.$$

This is the form of the linear system that will used in our linear solver.

**4.2. Linear Solver.** Many approaches to solving such systems have been proposed in the literature. A preconditioned iterative method is considered here, in particular, the preconditioned conjugate gradient (CG) method as the system is symmetric, positive-definite. The preconditioner used here is the mean-based preconditioner which approximates the random diffusion coefficient by its mean values[16]. It is inexpensive to apply and has been shown to perform very well under the assumption that the variance of the input data is small and a low-order polynomial chaos expansion is used.

**5. Numerical Experiments.** In this section, numerical experiments are reported comparing the multilevel approach with the standard, single-level approach. Consider the model problem (2.1) with $D = [0, 1]^3$ and source term $f(x) \equiv 1$. The diffusion coefficient $a(x, \xi)$ is modeled with a Karhunen-Loéve-like random field model with mean $\mu = 1$ and standard deviation $\sigma = 0.25$. We assume the variables in the expansion are uniform over $[-1, 1]$.

The equations are discretized in space using tri-linear finite element basis functions on a hexahedral mesh. The Trilinos Project [11] is used for the multilevel solution strategy. The Stokhos package offers tools for Embedded UQ methods and is used for the stochastic Galerkin discretization of the equations[15]. After discretization, the linear equations are solved using CG with a relative tolerance of $10^{-7}$ with mean-based preconditioning which is applied with a single V-cycle of algebraic multigrid. Tpetra is used for the linear algebra objects[2], Belos' CG is used as the iterative solver method[4] and the MueLu package provided the algebraic multigrid which is used to apply the mean-based preconditioner[7].

For the numerical experiments, we consider $M = 1$ random variables and $N = 5$ total order degree of the polynomial chaos expansion so $P + 1 = 6$. In the multilevel scheme, we use $K = 3$ and are interested in calculating the solution on a $32 \times 32 \times 32$ mesh where our spatial hierarchy is to coarsen the number of nodes in each spatial direction by a factor of 2. Then the multilevel formula for our solution is

$$\hat{u}_5^{32} = (u_{p_3}^{32} - u_{p_3}^{16}) + (u_{p_2}^{16} - u_{p_2}^{8}) + (u_{p_1}^{8} - u_{p_1}^{4}) + u_{p_0}^{4}.$$

We compute a reference solution, $u_{ref} = u_5^{64}$, using a $64 \times 64 \times 64$ mesh with $N = 5$ in order to compute the error of our calculated results against. Then, we vary the polynomial orders, $p_k$, in the multilevel formula to compute the multilevel approximation $\hat{u}_5^{32}$. The calculated solution is interpolated spatially to the reference mesh and the spatial $L^2$-norm of each polynomial chaos coefficient of the quantity $u_{ref} - \hat{u}_5^{64}$ is calculated. For the calculations, we chose a tolerance of $10^{-7}$ for the desired accuracy for each polynomial chaos coefficient in the computed solution. Then, we experimentally found the polynomial orders necessary to achieve the particular tolerance. The polynomial orders necessary were found to be $p_3 = 1$, $p_2 = 1$, $p_1 = 2$, and $p_0 = 4$ for a tolerance of $10^{-7}$ when compared to the reference soltion. In Figure 5.1, the spatial $L^2$-errors for each polynomial chaos coefficient are shown for the solution computed directly and the solution computed with the multilevel formula with the previously specified polynomial orders. It should be noted that we only expect the multilevel solution error to be at most $10^{-7}$ as that is our chosen accuracy.

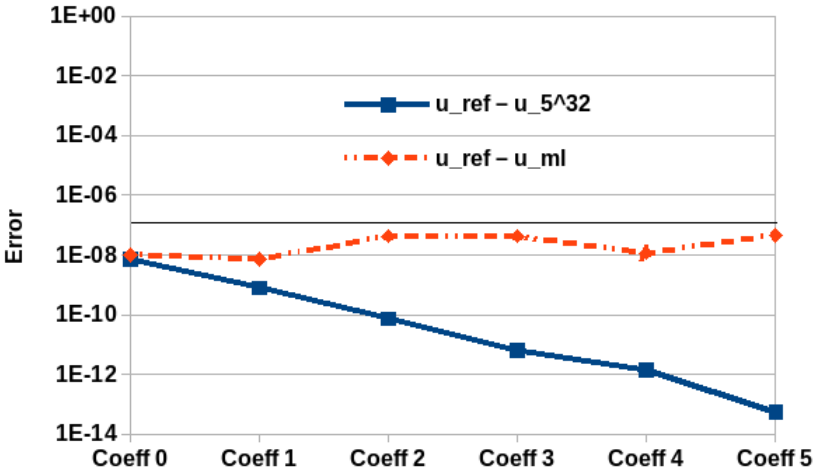



Fig. 5.1: Error in each coefficient of the solution computed directly and the solution compute with the multilevel method

The computational benefit in the multilevel method is the fact that on the fine grid the desired accuracy is achieved with $N = 1$, in our experimental example. In our example, only $M = 1$ random variable is considered, but in more realistic computations more random variables are considered. For example, consider $M = 10$ random variables then $P + 1 = 3,003$. Table 5.1 shows the degrees of freedom in the linear system for solving the PDE with different number of spatial nodes and polynomial orders for $M = 10$ random variables. If the solution $u_5^{32}$ were computed directly, a linear system with 107,918,811 degrees of freedom must be solved. Using the multilevel formula to compute $\hat{u}_5^{32}$, only 692,901 degrees of freedom must be found which offers considerable savings in computational cost.

| | Degrees of freedom | | | |
|---|---|---|---|---|
| Order | $N_x = 32$ | $N_x = 16$ | $N_x = 8$ | $N_x = 4$ |
| 1 | 395,307 | 54,043 | 8,019 | 1,375 |
| 2 | 2,371,842 | 324,258 | 48,114 | 8,250 |
| 3 | 10,277,982 | 1,405,118 | 208,494 | 35,750 |
| 4 | 35,972,937 | 4,917,913 | 729,729 | 125,125 |
| 5 | 107,918,811 | 14,753,739 | 2,189,187 | 375,375 |

Table 5.1: Total degrees of freedom for varying mesh sizes where $N_x$ is the number nodes in one-dimension and polynomial order for $N = 10$ random variables.

**6. Conclusions.** Using the stochastic Galerkin to solve PDEs with random coefficients is an effective yet expensive task especially as the stochastic dimension increases. A multilevel solution strategy was proposed based on the fact that the solution can be written as a telescoping sum of a hierarchy of spatial approximations. The ability to use lower-order stochastic approximations within the sum allows for computational savings. Numerical results were provided that show this method can achieve considerable computational savings while maintaining overall accuracy in the solution. In the future, we hope to extend this idea to a multigrid preconditioner for the stochastic Galerkin system where by using a spatially coarse-grid representation of the problem with different stochastic approximations in a multigrid hierarchy.

REFERENCES

[1] I. Babuska, F. Nobile, and R. Tempone, *A stochastic collocation method for elliptic partial differential equations with random input data*, SIAM Journal on Numerical Analysis, 45 (2007), pp. 1005–1034.

[2] C. Baker and M. Heroux, *Tpetra, and the use of generic programming in scientific computing*, Scientific Programming, 20 (2012), pp. 115–128.

[3] A. Barth, C. Schwab, and N. Zollinger, *Multi-level monte carlo finite element method for elliptic pdes with stochastic coefficients*, Numerische Mathematik, 119 (2011), pp. 123–161.

[4] E. Bavier, M. Hoemmen, S. Rajamanickam, and H. Thornquist, *Amesos2 and Belos: Direct and iterative solvers for large sparse linear systems*, Scientific Programming, 20 (2012), pp. 241–255.

[5] K. Cliffe, M. Giles, R. Scheichl, and A. L. Teckentrup, *Multilevel monte carlo methods and applications to elliptic pdes with random coefficients*, Computing and Visualization in Science, 14 (2011), pp. 3–15.

[6] G. Fishman, *Monte Carlo*, Springer Series in Operations Research, Springer-Verlag, New York, 1996. Concepts, algorithms, and applications.

[7] J. Gaidamour, J. Hu, C. Siefert, and R. Tuminaro, *Design considerations for a flexible multigrid preconditioning library*, Scientific Programming, 20 (2012), pp. 223–239.

[8] R. Ghanem and P. D. Spanos, *Polynomial chaos in stochastic finite elements*, Journal of Applied Mechanics, 57 (1990).

[9] R. G. Ghanem and P. D. Spanos, *Stochastic finite elements: a spectral approach*, Springer-Verlag, New York, 1991.

[10] M. Gunzburger, P. Jantsch, A. L. Teckentrup, and C. G. Webster, *Multilevel acceleration of stochastic collocation methods for pdes with random input data*, arXiv preprint arXiv:1404.2647, (2014).

[11] M. Heroux, R. Bartlett, V. Howle, R. Hoekstra, J. Hu, T. Kolda, R. Lehoucq, K. Long, R. Pawlowski, E. Phipps, A. Salinger, H. Thornquist, R. Tuminaro, J. Willenbring, A. Williams, and K. Stanley, *An overview of the Trilinos package*, ACM Trans. Math. Softw., 31 (2005). `http://trilinos.sandia.gov/`.

[12] N. Metropolis and S. Ulam, *The Monte Carlo method*, Journal of the American Statistical Association, 44 (1949), pp. 335–341.

[13] F. Nobile, R. Tempone, and C. G. Webster, *A sparse grid stochastic collocation method for partial differential equations with random input data*, SIAM Journal on Numerical Analysis, 46 (2008), pp. 2309–2345.

[14] ———, *An anisotropic sparse grid stochastic collocation method for partial differential equations with random input data*, SIAM Journal on Numerical Analysis, 46 (2008), pp. 2411–2442.

[15] E. T. Phipps, *Stokhos Stochastic Galerkin Uncertainty Quantification Methods*. `http://trilinos.sandia.gov/packages/stokhos/`, 2011.

[16] C. E. Powell and H. Elman, *Block-diagonal preconditioning for spectral stochastic finite-element systems*, IMA Journal of Numerical Analysis, 29 (2009), pp. 350–375.

[17] D. Xiu and J. Hesthaven, *High-order collocation methods for differential equations with random inputs*, SIAM Journal on Scientific Computing, 27 (2005), pp. 1118–1139.

[18] D. Xiu and G. Karniadakis, *The wiener-askey polynomial chaos for stochastic differential equations*, SIAM Journal on Scientific Computing, 24 (2002), pp. 619–644.

# LAMMPS KOKKOS PACKAGE

SIKANDAR Y. MASHAYAK*, CHRISTIAN R. TROTT†, AND STEVEN J. PLIMPTON‡

**Abstract.** In this work, we developed performance enhancements of the LAMMPS[4] molecular dynamics package for hybrid HPC platforms that use accelerators, such as GPUs and Intel Xeon Phi co-processors, in addition to multi-core CPUs. Specifically, we used the Kokkos [3] C++ library to port portions of LAMMPS to multiple accelerator devices. The following LAMMPS components were accelerated using Kokkos: atoms styles charge, bond, angle, molecular, and full; pair styles coul/cut and lj/cut/coul/cut; and fix style langevin. Benchmark tests using GPUs and Intel Xeon Phi accelerators were performed to analyze the peformance gain due to the Kokkos-enhanced styles.

**1. Introduction.** LAMMPS is a molecular dynamics (MD) code used to simulate the properties of fluids, solids, and molecules. MD simulations of systems such as biomolecules, polymers, materials, and mesoscale models are computationally intensive. MD calculations, however, are inherently parallel, and simulations can be performed faster by distributing the parallel calculations across multiple processes. LAMMPS uses spatial-decomposition techniques that partition a simulation domain into smaller subdomains and performs the subdomain computations in parallel via MPI on multiple CPUs. With the advent of hybrid HPC platforms, further speed-up can be achieved by utilizing the diverse and evolving accelerator hardware. In addition to parallel domain decomposition, an accelerator allows for exploitation of finer-grain parallelism by spreading the computations assigned to a single MPI task across the dozens or many hundreds of cores available on the accelerator.

To run LAMMPS on an accelerator, portions of LAMMPS code typically need to be re-written or ported to the specific accelerator. A major challenge in this effort is that the instruction sets, programming model, compilers, and memory access patterns for different accelerator hardware are very different. Various LAMMPS packages that port LAMMPS code to a specific accelerator have been developed. For example, the GPU[2, 1] and USER-CUDA packages implement a subset of LAMMPS styles for GPUs. The USER-OMP package does the same for multi-core CPUs using OpenMP threads. Because accelerator hardware and programming models are continually evolving and becoming more diverse, it is increasingly complicated to adapt LAMMPS code for multiple accelerator targets. This is a problem Kokkos attempts to solve, by providing a unifying abstraction that allows one version of ported code to be run on a variety of accelerator hardware. Other programming models, such as OpenCL, C++ AMP, and Thrust have a similar goal.

In this work, we used the Kokkos C++ library to port key LAMMPS kernels to various accelerators. Kokkos provides a unifying abstraction for both the data parallelism and memory access patterns across various architectures. This abstraction allows Kokkos to maximize the portions of a legacy code that can be compiled for various devices and still achieve nearly the same performance as if the code kernels were specifically written for a specific device. Kokkos currently supports CUDA for NVIDIA GPUs, and pthreads or OpenMP threads for CPUs and Intel Xeon Phi co-processors.

**2. Methods.**

*University of Illinois at Urbana-Champaign, Department of Mechanical Science and Engineering, mashaya1@illinois.edu,

†Sandia National Laboratories, crtrott@sandia.gov,

‡Sandia National Laboratories, sjplimp@sandia.gov

**2.1. LAMMPS.** LAMMPS is a modular object-oriented C++ code. It uses virtual base classes to define core components of an MD simulation, such as the geometric domain, lists of atoms and their properties, molecular topology, and interaction potentials between atoms. Likewise it defines integrators and computations that take place during a timestep, such as constraint forces, thermostats and barostats, as well as analaysis calculations such as thermodynamic and dynamic properties of the system. These base classes can be used to implement new styles for computing interactions, per-timestep computations, and analysis calculations. For example, the Lennard-Jones pairwise interaction is implemented as a derived class of the pair style base class. We leveraged this modular design to make new styles that use Kokkos functionality to perform their computation on Kokkos-supported hardware. To make a specific style work with Kokkos, we define a new class, which can be derived from the CPU version of the style, or from the base style class, and implement only the subset of methods defined by the base class that will run on the device via Kokkos.

**2.2. Kokkos.** As mentioned in the Introduction, the unifying abstractions within Kokkos enable performance portability across several flavors of manycore architectures. An MPI process, running on compute node acts as a single master thread and dispatches computational kernels for worker threads running on the manycore devices connected to the node. Kokkos executes the data parallel computational kernels in an execution space and the kernels operate on data laid out as multi-dimensional arrays located in the memory space accessible from the execution space. To make the multidimensional arrays portable across different hardware, Kokkos provides a polymorphic data layout via a templated class called *View*, such that the optimal layout of the array can be tuned to specific hardware. Typically, on a single node of a hybrid HPC platform, Kokkos divides the execution and memory spaces into two categories: the host space and device space. For example, for a node connected to a GPU, the host space is on the CPU and the device space is on the GPU. For an Intel Xeon Phi co-processor running in native mode, both the host and device spaces are on the Phi. Further details about Kokkos abstractions, multidimensional arrays, and parallel execution are given in [3].

**2.3. LAMMPS-Kokkos.** To port individual LAMMPS styles to various accelerators using the Kokkos, we followed the migration strategy given in [3]. This strategy has five steps: (1) change data structures, (2) develop functors, (3) enable dispatch for device execution, (4) optimize algorithms for threading, and (5) specialize kernels for specific architectures. The details of these steps for LAMMPS styles are similar to the steps used for the Mantevo miniMD application, as explained in [3].

Specifically, we created Kokkos versions of a small subset of the LAMMPS atom, pair, and fix styles. These were added to the Kokkos package in LAMMPS which also contains various core classes which are added to or replace core LAMMPS classes when building with Kokkos support. Before this work, the Kokkos package had a few styles (atom style atomic, pair style lj/cut, and fix style nve) which were previously implemented by Christian Trott and released in the 29May2014 version of LAMMPS.

In this work, we added several new styles to the Kokkos package:

1. Atom styles:
   (a) charge
   (b) bond
   (c) angle
   (d) molecular
   (e) full

2. Pair styles:
   (a) coul/cut
   (b) lj/cut/coul/cut
3. Fix style:
   (a) langevin

We also made modifications to a few of the core classes of the Kokkos package. Specifically, *PairKokkos* was extended to include Coulomb components of force and energy, and *NeighborKokkos* was extended to allow exclusion or inclusion of bonded atoms when a neighbor list is built.

**3. Results and Discussion.** We performed tests to validate the accuracy of the newly implemented Kokkos styles by comparing simulation results with the original CPU-based styles. We then performed benchmarks to evaluate the performance gain due to the Kokkos package, and compared to the performance of other packages available in LAMMPS. Specifically, we compared the Kokkos performance to the CPU package, which is the standard LAMMPS MPI-only source code, to the GPU and USER-CUDA packages which implement various LAMMPS styles for NVIDIA GPUs, and to the USER-OMP package which implements styles with OpenMP threading for multi-core CPUs. Results of the tests are presented in the next two sub-sections.

**3.1. Validation.** In this section, we check the accuracy of the fix langevin style as implemented in Kokkos. For fix langevin, the routines that loop over owned atoms, such as the *post_force, compute_scalar*, and *end_of_step* were parallelized using Kokkos. Also, the random force component of fix langevin was altered, to enable thread-safe parallel generation of random numbers. This was done using built-in Kokkos random number generator. One limitation of the Kokkos-based fix langevin is that the packing of data exchanged between processors must be performed on the host, which can impact performance. To test the new implememtation, an atomic fluid system with the Lennard-Jones potential was simulated using fix nve and fix langevin for thermostatting. The time evolution of the system temperature, potential energy, and pressure were compared with the CPU-based LAMMPS results. Fig. 3.1 shows that the results from the two implementations match each other.

**3.2. Benchmarks.** For performance benchmarking, we also used an atomic fluid system with the Lennard-Jones potential. The force cut-off was $2.5\sigma$, the neighbor skin was $0.3\sigma$, and time integration was done in the constant NVE ensemble. Benchmark runs were performed on the Shannon and Compton testbeds at Sandia. Kokkos performance was compared to both CPU performance and to other packages in LAMMPS that provide accelerator support, namely the GPU, USER-CUDA, and USER-OMP packages. In addition, we also compared LAMMPS performance on this benchmark problem with the popular GROMACS[5] MD program, which is known to provide very high performance.

**3.2.1. NVIDIA GPU accelerator on Shannon machine.** Each node of the Shannon machine, used in this section, has two 8-core Sandy Bridge Xeon E5-2670 2.6GHz HT-deactivated CPUs as well as 2 NVIDIA K20x GPUs.

The first set of benchmark results are for simulations using various LAMMPS packages running on a single node of Shannon. Problem sizes from 2K to 8M atoms were used. For each problem size and package, the run times for different combinations of the number of MPI tasks, number of OpenMP threads per task, and number of GPUs were obtained. From this data, for each problem size and package, the combination that produced the fastest run time was determined. For example, in

Fig. 3.2(a), for the Kokkos package compiled with OpenMP support (CPU-only, no use of the GPUs), the optimal performance is achieved with 16 MPI ranks and 1 thread per MPI rank for all problem sizes. In Fig. 3.2(b), for the Kokkos package compiled with both OpenMP support (on the CPU) and GPU support, for problem sizes $\leq$ 16K optimal performance was achieved with only one GPU. For larger problem sizes two GPUs were optimal. In both cases, the number of OpenMP threads per MPI tasks did not significantly affect the performance, because nearly all computations were performed on the GPUs.

Fig. 3.3 shows results for all the CPU and GPU accelerator packages tested, running on a single node of Shannon. Each data point is the performance using the optimal combination of the MPI tasks, OpenMP threads/task, and number of GPUs for that package and problem size. The Y-axis performance metric is millions of atom-timesteps per second, i.e. a value of 10 for the 256,000 atom system means the simulation runs at the rate of roughly 40 timesteps/second. The data shows that for problem sizes smaller than 16K atoms the USER-OMP package is the fastest; for larger problem sizes the Kokkos package compiled with GPU support is the fastest. Of particular interest is that for large problems the Kokkos package with GPU support outperforms both the GPU and USER-CUDA packages previously implemented in LAMMPS. Both those packages were tested in double-precision mode which is the only GPU mode currently available in Kokkos. For mixed- and single-precision mode, the GPU and USER-CUDA package performance is closer to the Kokkos double-precision performance, offering the hope that Kokkos performance might also increase if other precision options are implemented in the future.

We note that the performance of the packages that use the GPUs (Kokkos, GPU, and USER-CUDA packages) improves signficantly as the problem size increases from 2K to 2M atoms. For the Kokkos-GPU package the improvement is 26x, for the GPU package it is 6x, and for the USER-CUDA package it is 29x.

The CPU-only simulations, with Kokkos compiled with OpenMP support and the USER-OMP package, performed best when all using 16 MPI tasks (on 16 cores). There is little increase in performance with problem size for the CPU runs, reflecting the $O(N)$ linear scaling of short-range MD models. It is worth noting that the performance of the Kokkos with OpenMP is similar to the CPU Lennard-Jones (LJ) style in LAMMPS. The USER-OMP package is about 20% faster, because it implements an optimized version of the LJ pairwise force kernel.

One of the main conceptual differences between the Kokkos (with GPU support) and GPU packages is that the GPU package transfers data between the GPU and host CPU at every time-step. By contast, in the Kokkos package, data stays on a GPU until a non-Kokkos operation (pair, fix, compute, or communication style) is invoked. For the simple LJ benchmark, both the pair_style lj/cut and fix nve styles were implemented in the Kokkos package. This means data can stay on the GPU for many timesteps, resulting in less communication. More realistic (non-benchmark) input scripts might not allow for this reduced communication, due to invocation of commands that use pair, fix, or compute styles not (yet) supported by Kokkos. To evaluate the effect of CPU/GPU communication on performance, we ran the Kokkos simulations using the CPU-based fix nve style, which force data communication between the GPU and CPU at every time step. Fig. 3.4 shows the the results. The Kokkos simulations are now slower than the GPU package, using either 1 or 2 GPUs. This indicates that there may be communication performance improvements that can be made in the Kokkos package, to take advantage of similar optimizations imple-

mented by the GPU package.

To test strong-scaling for the various CPU- and GPU-based packages in LAMMPS on Shannon, a fixed problem size of 2M (2097152) atoms was run on different number of nodes. To test weak-scaling a scaled problem with of 512K (524288) atoms per node was run on different number of nodes. In both cases the node count varied from 1 to 24 nodes. The results are shown in Fig. 3.5(a) and 3.5(b). In both cases the Y-axis is now plotted as millions of atom-timesteps per second per node. This means that ideal scaling in both plots would be a horizontal line.

For the strong-scaling test with the GPU-based packages, performance falls off as node count increases. This is because the problem size per node is decreasing and the performance is essentially back-tracking to the left on the corresponding data curves in Fig. 3.4. The CPU-based packages do not show the same effect, since their performance in Fig. 3.4 is largely independent of problem size. The weak-scaling test shows as less-dramatic fall-off with increasing node count, since the problem size per node remains constant. There is still some decrease in performance for the Kokkos with GPU-support results, which is likely due to the cost of MPI-based inter-node communication which reaches a constant cost at 8 nodes for the style of spatial-decomposition LAMMPS uses for a periodic 3d problem.

We also compared LAMMPS performance to GROMACS on Shannon. For the GROMACS simulations (current version 5.0), to make a fair comparison, we tried to select parameters and algorithms in GROMACS that were as similar to the LAMMPS simulations as possible. Specifically, the GROMACS runs were performed using LJ units, a LJ pair potential with a cutoff of $2.5\sigma$, grid-based neighbor list formation with an update frequency of 20 timesteps, and time integration using the NVE ensemble. We note that the GPU-based version of GROMACS performs its calculations on the GPU in single precision, whereas Kokkos (with GPU support) is double-precision. To obtain reliable estimates of the GROMACS timings, we ran the GROMACS simulations for 1000 timesteps and timed the last 500 steps for only the loop timing (excluding setup costs). Fig. 3.6 shows the comparison of the two codes on this benchmark problem, where "gmx" stands for GROMACS, for the same strong- and weak-scaling tests described for the previous plots. The CPU performance of GROMACS is nearly 2x faster than LAMMPS, though the difference would be less if the USER-OMP (or new USER-INTEL) package were used for this problem. For GPUs, the Kokkos (with GPU support) package is slightly faster than GROMACS for small node counts and essentially the same for larger node counts, even with the single- vs double-precision trade-off. We note that this is a simple LJ atomic fluid. Further benchmarking for more complex systems (water, solvated proteins, etc) need to be performed, especially since these are the kinds of systems GROMACS is specifically optimized for.

**3.2.2. Intel Xeon Phi accelerator on Compton machine.** Each node of the Compton machine, used in this section, has two 8-core 2.60 GHz Intel Xeon E5-2670 CPUs as well as two 57-core, 228 threads, 1.10 GHz pre-production Intel Xeon Phi co-processors.

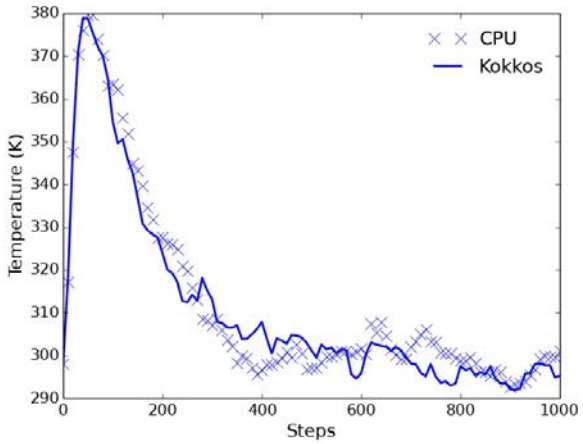
The next set of benchmark results compare the performance of the LAMMPS USER-OMP and CPU packages, with Kokkos compiled with Xeon Phi support. This is for runs on a single Intel Xeon Phi co-processor. Using the Kokkos package in this mode, Kokkos allocates both the host and device spaces on the co-processor, and it can use mulitple physical cores on the co-processor via MPI and/or OpenMP threads. Therefore, to benchmark Kokkos we ran each problem size on various combinations of the number of MPI tasks and the number of OpenMP threads per task. The

LAMMPS CPU and USER-OMP packages were also run in similar manner. Note that these runs were also on the Xeon Phi co-processor, not on the Compton CPUs. The Compton CPUs are very similar to the Shannon CPUs, and so we expect that running the Kokkos, CPU, and USER-OMP packages entirely on the Compton CPUs, without using the Phi co-processor, would give performance identical to that indicated by the graphs in Fig. 3.3 labelled Kokkos-OMP, CPU, and USER-OMP, respectively.

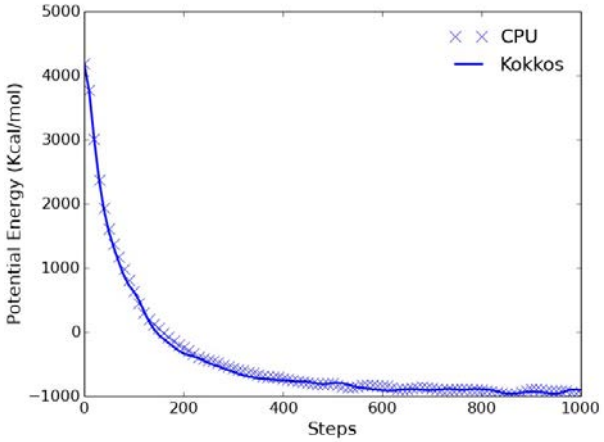
The performance results for the 3 packages running on a single Intel Xeon Phi accelerator are shown in Figures 3.7(a), 3.7(b), and 3.8. For all the problem sizes, the CPU package performance was best for 128 MPI tasks (not the maximum available 224). For the USER-OMP package, performance was best for 64 MPI tasks and 3 OpenMP threads per task (using 192 = 64*3 physical cores). For the Kokkos package, performance was best for 32 MPI tasks and 7 OpenMP threads per MPI task (all 224 cores). However the overall performance of Kokkos with Xeon Phi support was only slightly better than the CPU package performance and slightly worse than the USER-OMP package. More importantly, the performance is about 2x slower (at the largest problem sizes) than the Shannon CPU results shown in Fig. 3.3 for the same benchmark test, and dramatically slower than the GPU results in the same plot.

Our current thinking is that the relatively poor performance of LAMMPS on the Xeon Phi co-processor is due to limiations in the current Phi hardware and not the Kokkos package itself. For example, on the current generation Phi, gather operations from cache are slower compared to on a GPU, and in the LJ pairwise force kernel used in this benchmark, there are 7 gather operations per 24 math operations. Specifically the cutoff distance, x, y, z, type, epsilon, and sigma parameters must be gathered for each atom's neighbors.

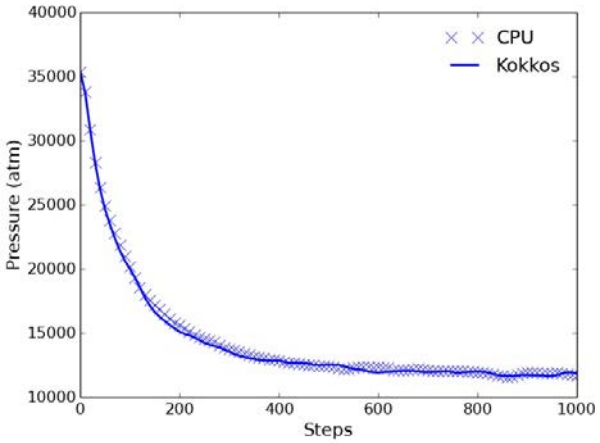
We also note that Kokkos uses the Xeon Phi co-processors in "native" mode. There is an independent effort by Intel to develop a USER-INTEL package for LAMMPS with Xeon Phi support for various pair styles which uses the Phi in "offload" mode. We plan to make a performance comparison of the two modes in the future.




(a) Temperature.

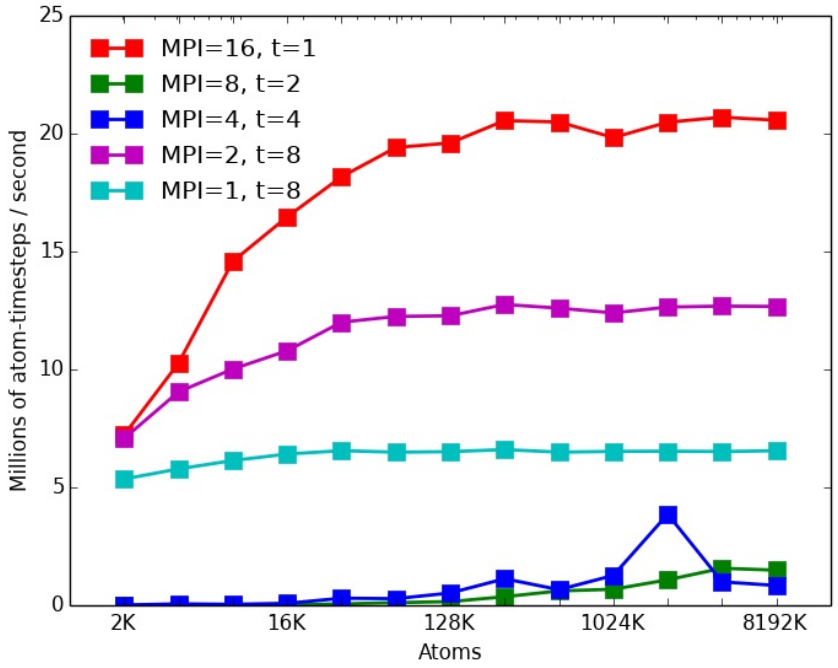



(b) Total potential energy.

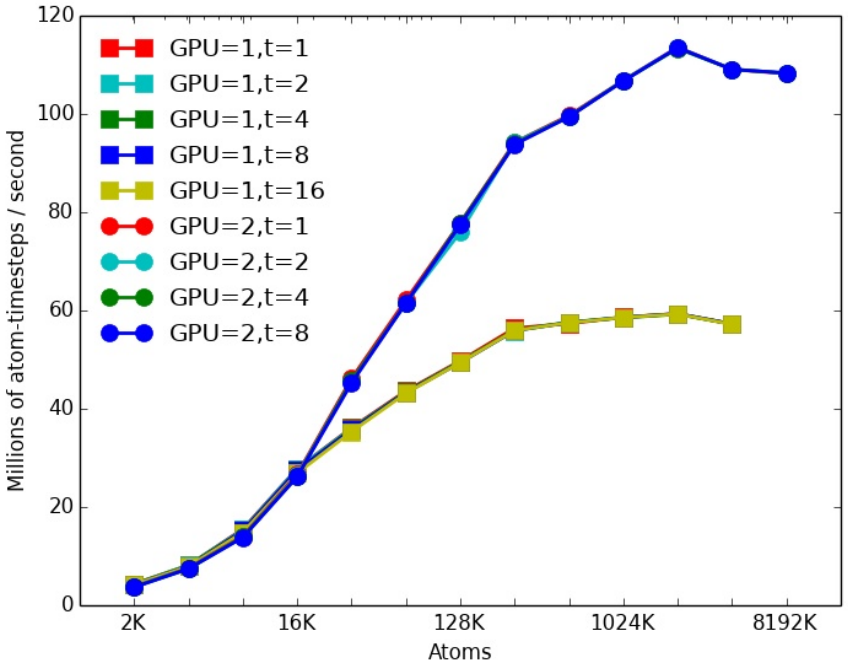



(c) Pressure.

Fig. 3.1: Comparison of thermodynamic properties from simulations using the Kokkos and CPU-based fix langevin thermostat.




(a) With OpenMP threads.




(b) With GPU and OpenMP threads.

Fig. 3.2: Kokkos performance on a single node of Shannon.

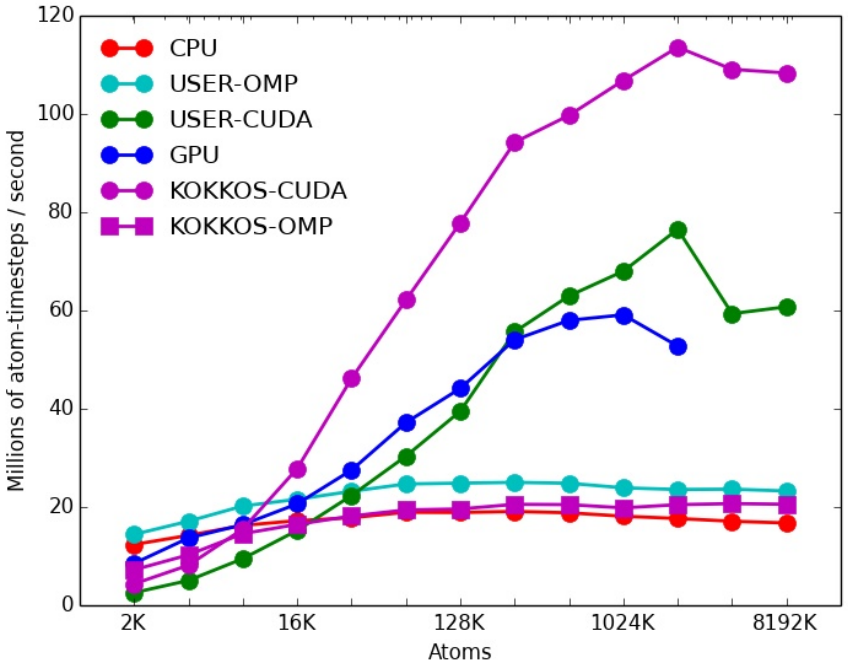



Fig. 3.3: Performance of various CPU- and GPU-based packages in LAMMPS on a single node of Shannon.

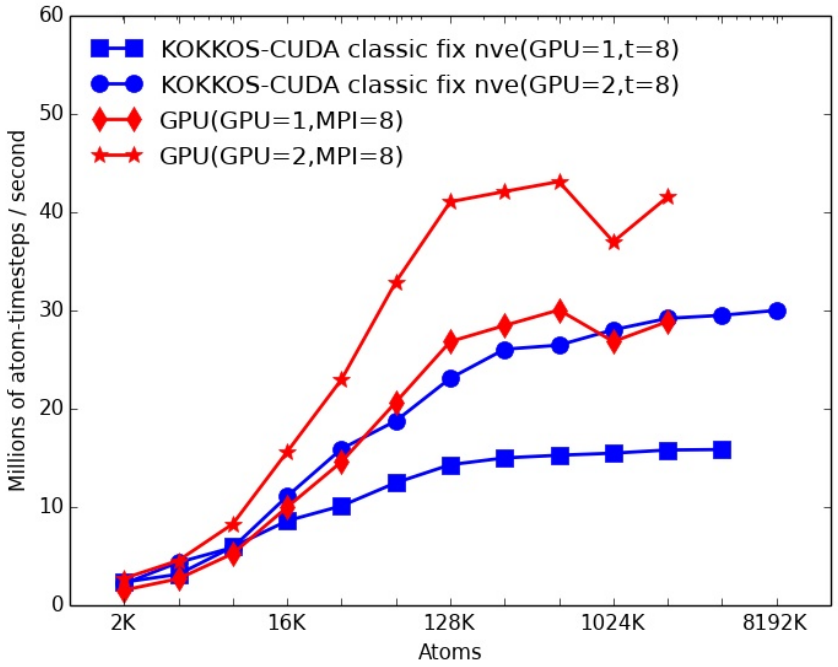
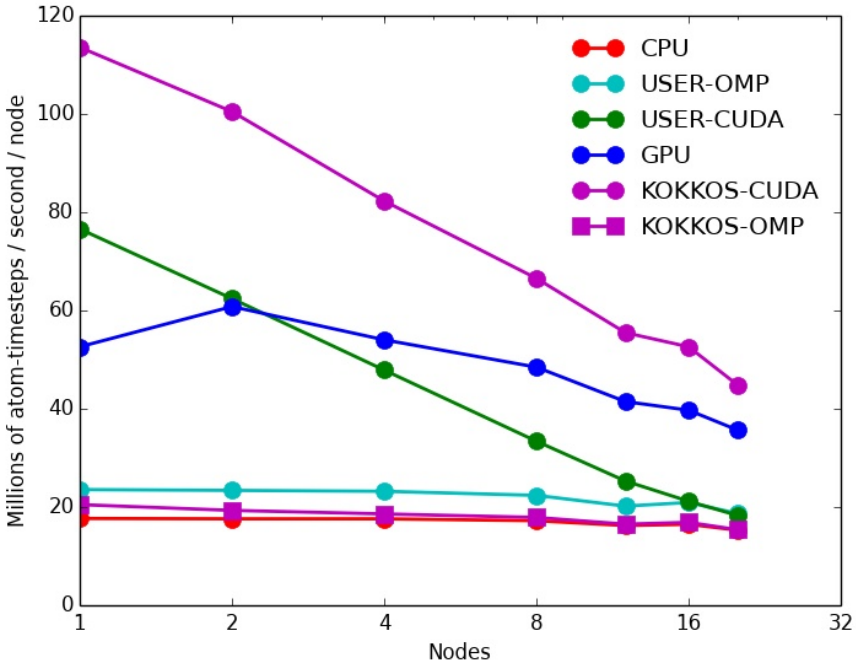
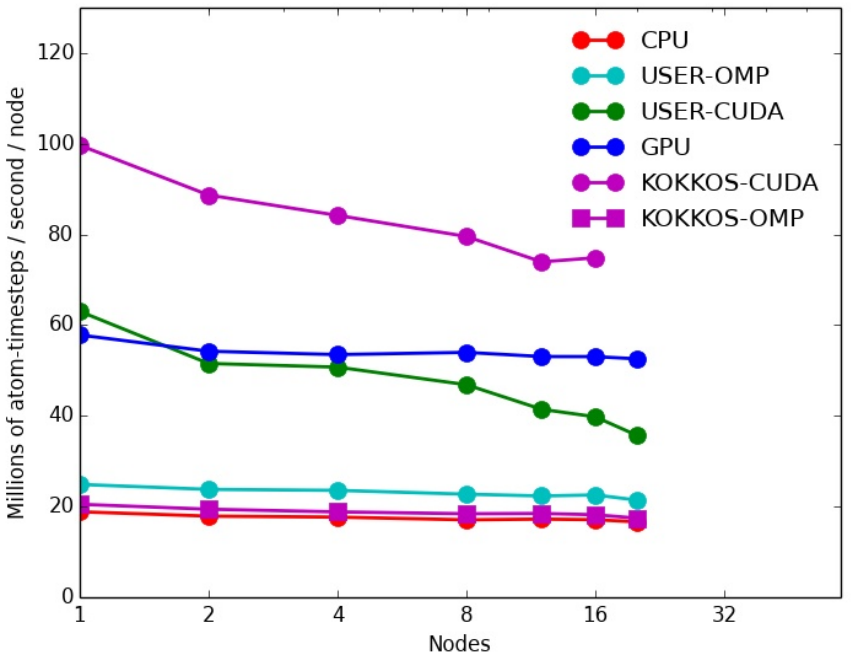



Fig. 3.4: Comparsion of the GPU and Kokkos packages on a single node of Shannon, as communication costs were varied.

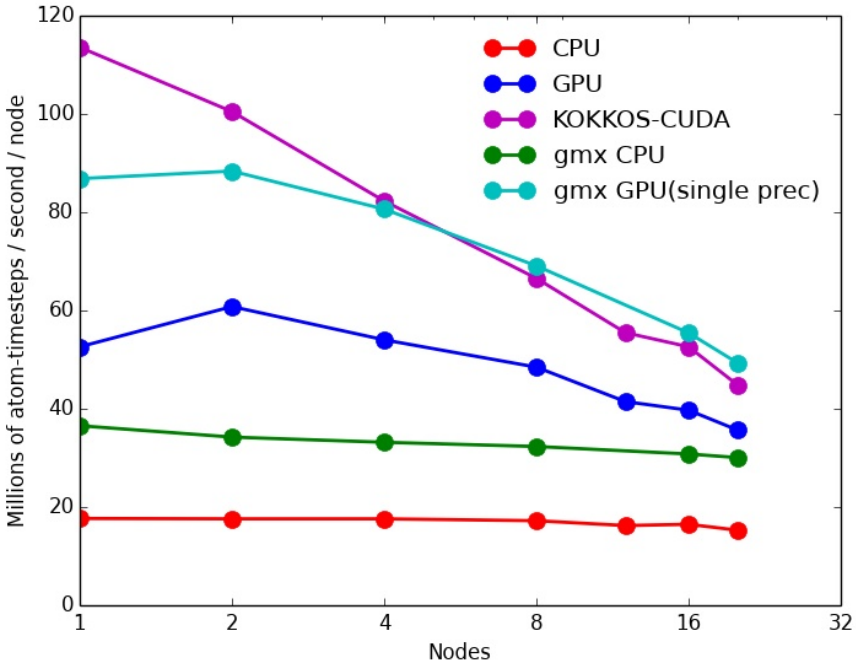



(a) Strong-scaling.

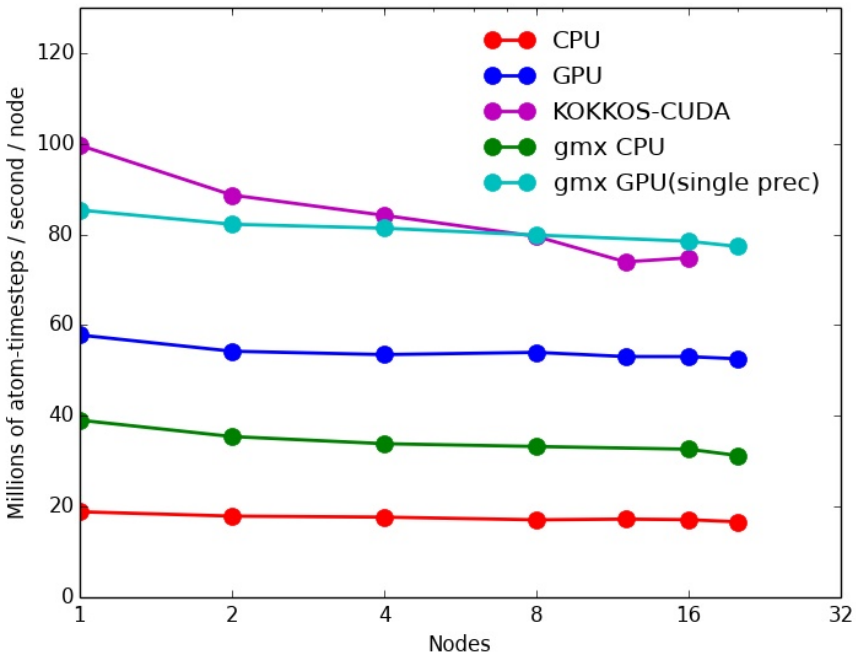



(b) Weak-scaling.

Fig. 3.5: Strong-scaling and weak-scaling performances on Shannon.

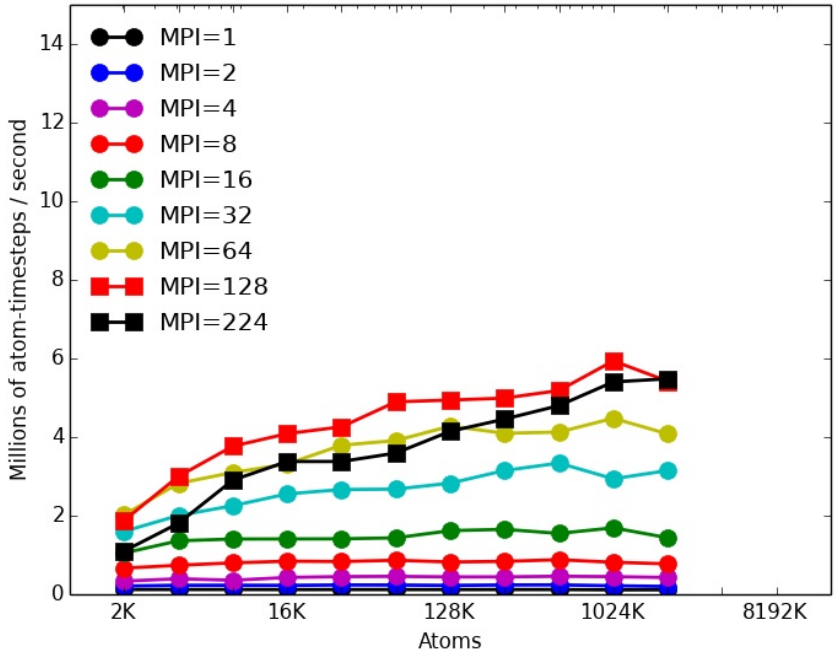



(a) Strong-scaling.

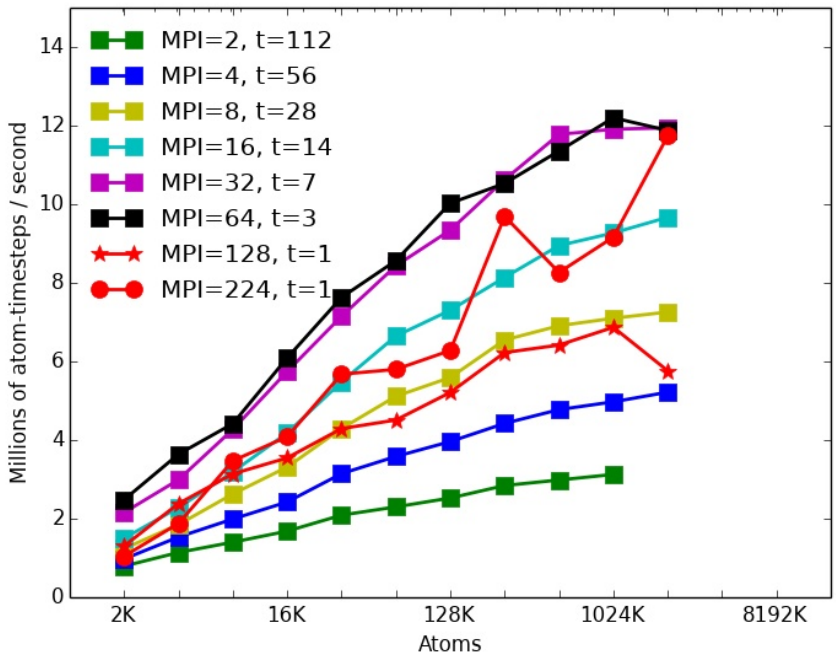



(b) Weak-scaling.

Fig. 3.6: Comparison of LAMMPS and GROMACS performances on Shannon.




(a) CPU.




(b) USER-OMP.

Fig. 3.7: Performance of the CPU and USER-OMP package on a single Intel Xeon Phi accelerator of a Compton node.

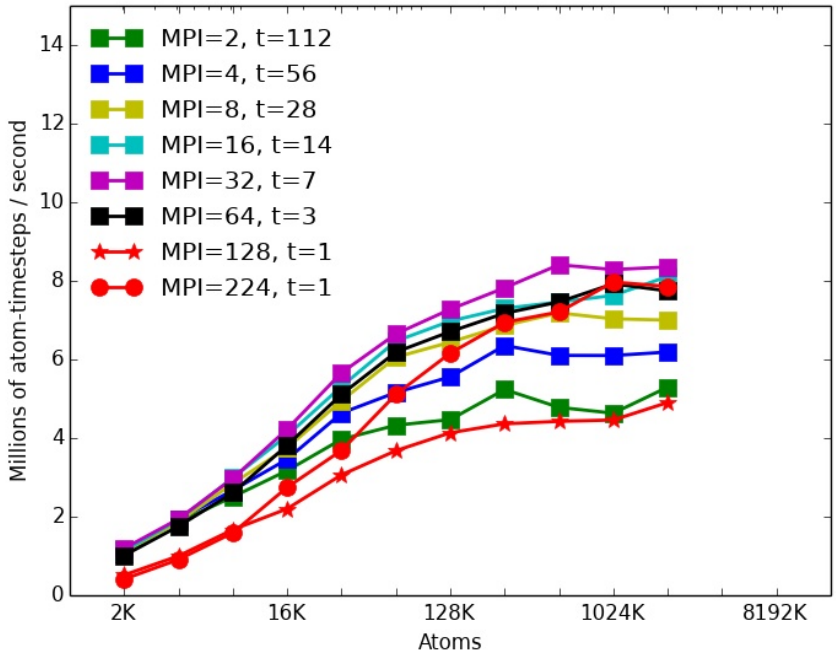



Fig. 3.8: Performance of the Kokkos package on a single Intel Xeon Phi accelerator of a Compton node.

**4. Conclusions.** In this work, we implemented various "styles" available in the LAMMPS molecular dynamics simulator to enable Kokkos support. These include atom styles charge, bond, angle, molecular, and full; pair styles coul/cut and lj/cut/-coul/cut; and fix style langevin. The accuracy of the newly implemented styles were validated against the original CPU versions. To assess the performance benefits offered by Kokkos for multi-cores, GPUs and Intel Xeon Phis, we ran benchmark tests of a simple Lennard-Jones atomic fluid model on the Shannon and Compton testbeds at Sandia. For large problem sizes we found the peformance of Kokkos on GPUs to be faster than any of the other accelerator packages currently available in LAMMPS. Due to hardware limitations of the current-generation Intel Xeon Phi co-processors, the Kokkos package performed relatively poorly on Intel Xeon Phi co-processors. We are hopeful that the next-generation Xeon Phi will show improved performance for LAMMPS. The various Kokkos-enabled styles discussed in this report were released in LAMMPS in the 29Aug2014 version, as part of its Kokkos package.

REFERENCES

[1] W. M. Brown, A. Kohlmeyer, S. J. Plimpton, and A. N. Tharrington, *Implementing molecular dynamics on hybrid high performance computers particleparticle particle-mesh*, Computer Physics Communications, 183 (2012), pp. 449–459.

[2] W. M. Brown, P. Wang, S. J. Plimpton, and A. N. Tharrington, *Implementing molecular dynamics on hybrid high performance computers short range forces*, Computer Physics Communications, 182 (2011), pp. 898–911.

[3] H. Carter Edwards, C. R. Trott, and D. Sunderland, *Kokkos: Enabling manycore performance portability through polymorphic memory access patterns*, Journal of Parallel and Distributed Computing, (2014).

[4] S. Plimpton, *Fast parallel algorithms for short-range molecular dynamics*, Journal of Computational Physics, 117 (1995), pp. 1–19.

[5] S. Pronk, S. Páll, R. Schulz, P. Larsson, P. Bjelkmar, R. Apostolov, M. R. Shirts, J. C. Smith, P. M. Kasson, D. van der Spoel, B. Hess, and E. Lindahl, *Gromacs 4.5: a high-throughput and highly parallel open source molecular simulation toolkit*, Bioinformatics, 29 (2013), pp. 845–854.

# FORCE CONVERGENCE IN STOPPING POWER FROM MOLECULAR DYNAMICS

PHILIP M. VAN EVERY*, ANDREW D. BACZEWSKI†, AND RUDOLPH J. MAGYAR‡

**Abstract.** Using Ehrenfest time-dependent density functional theory (TDDFT), we simulate a proton passing through several elemental metals under ambient conditions. Convergence in the force acting on the proton with varying k-point grids is analyzed. We also investigate the inclusion of a gauge correction that remedies issues with charge non-conservation in the projector augmented wave (PAW) formalism. The computational speed of varying band parallelization parameters in different system sizes is also explored. We simulate stopping power curves in Lithium, Beryillium, and Aluminum. All simulations are performed with a modified version of VASP and post-processed with local scripts.

**1. Introduction.** The energy loss by an ionized particle travelling through a crystal lattice, called stopping power, is a characteristic of interest in many fields. It has been studied in the context of nuclear applications, including radiation damage, and its implications in space and medical applications have been duly noted [3]. The ability to simulate and calculate stopping power has important implications in magnetic liner intertial fusion and other applications studied with the Z-machine here at Sandia National Laboratories.

Classical molecular dynamics [4] simulations, as well as density functional theory (DFT) molecular dyanmics [3] simulations have been used to compute stopping power and related characteristics. The method explored herein employs an implemented Ehrenfest time-dependent density functional theory (TDDFT) capability in a modified version of the Vienna Ab-Inito Simulation Package (VASP) [7, 8, 5, 6] in the Projector Augmented Wave (PAW) [9, 2, 11] formalism.

K-point grids play a fundamental role in periodic electronic structure simulations. They have not been extensively studied in the context of Ehrenfest-TDDFT. In Section 2 we assess the convergence of force curves in Γ-Centered, Monkhorst Pack [10] schemes. High symmetry and mean value k-point schemes are also explored.

To conserve electron number, a gauge correction has been implemented. The effects of this gauge correction on the force curve and stopping power calculation for a system are explored in Section 3. We also address its computational cost.

In high-performance computing environments, computational speed is a ubiquitous constraint. Distribution of work over electronic bands can have a dramatic effect. In Section 4, we spend some time exploring the effect on computational speed of varying band parallelization parameters and levels of computational resource usage.

With some insight into the roles of the above variables, we compute stopping power curves in Lithium, Beryllium, and Aluminum.

**2. Force Convergence in Varying K-Point Grid Schemes.** To explore the effects of varying k-point grids, we used a 32 atom, body-centered cubic (BCC) Lithium supercell at 300K, with a $3.49\mathring{A}$ lattice constant, tiled from a 2 atom unit cell in an aspect ratio of $4 \times 2 \times 2$. A proton traverses the cell in a channeled path at a velocity of 1.25 au or 27.346 $\mathring{A}$/fs as in Figure 2.1.

*University of New Mexico, pvanevery@unm.edu
†Sandia National Laboratories, adbacze@sandia.gov
‡Sandia National Laboratories, rjmagya@sandia.gov

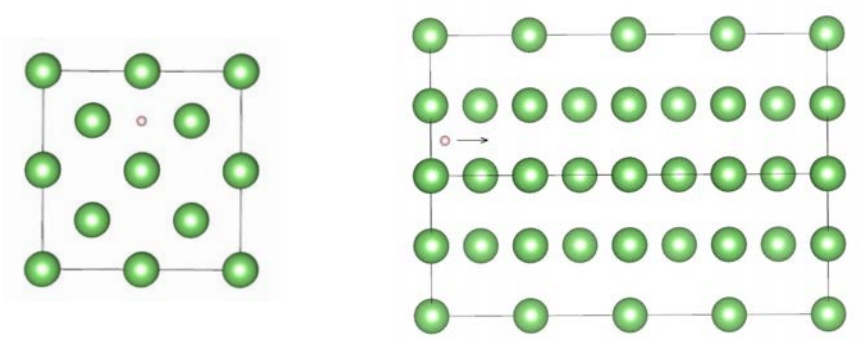


Fig. 2.1: rear (000) and side (101) views of the proton in the 32 atom Li supercell

The following types of k-point grids were tested for force convergence:

**Γ-Centered:** A symmetric k-point mesh is automatically generated with its origin at the Γ-point [7]. Γ-Centered k-point grids were tested in aspect ratios of $1 \times 2 \times 2$, $2 \times 3 \times 3$, $3 \times 6 \times 6$, and $4 \times 8 \times 8$. As seen in Figure 2.2, the $2 \times 3 \times 3$, $3 \times 6 \times 6$, and $4 \times 8 \times 8$ configurations yielded stable curves, and the $1 \times 2 \times 2$ configuration did not, suggesting that it did not adequately sample the system.

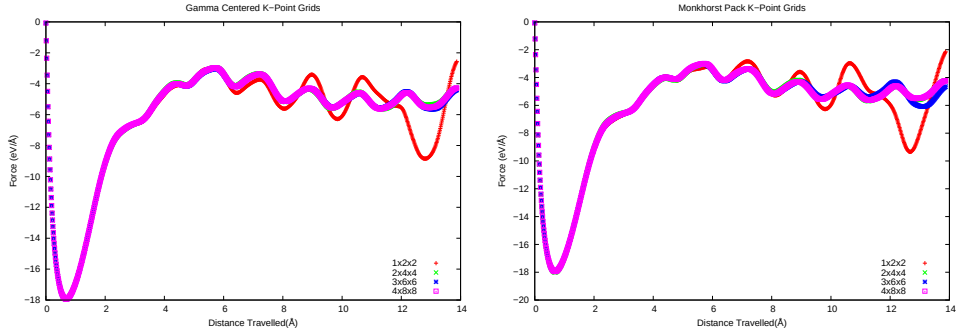



Fig. 2.2: Γ-Centered and Monkhorst Pack k-point mesh yield almost identical force curves. In both k-point schemes, the 1x2x2 force curve shows unstable oscillation starting at around 7 $\mathring{A}$.

**Monkhorst Pack:** K-point grids with even subdivisions are shifted off Γ [7] by an algorithm developed by Hendrik Monkhorst and James Pack [10]. Monkhorst Pack k-point grids were tested in the same aspect ratios listed above. Again, the $2\times3\times3$, $3\times6\times6$, and $4\times8\times8$ configurations yielded visually stabilizing force curves, while the $1\times2\times2$ configuration did not, as seen in Figure 2.2.

**High symmetry and mean-value points K-Point:** The Baldereschi mean-value point [1] for a BCC cell, the Γ-point, and several other mean-value k-point variations were explored. None of the mean-value k-point calculations yielded a stabilizing force curve, as seen in Figure 2.3.

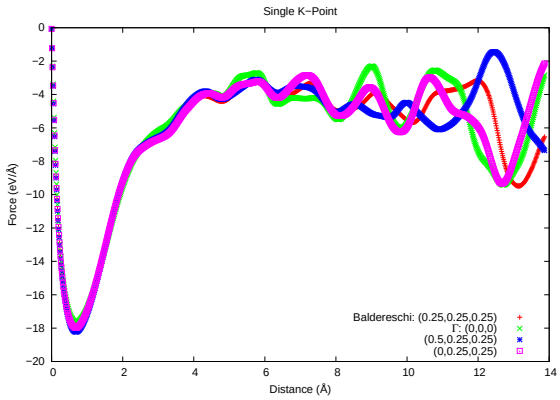



Fig. 2.3: Single k-point force curves in the 32 atom Li system fail to stabilize. Each force curve shows unstable oscillation beginning at around 6 $\mathring{A}$.

The $4\times8\times8$ Γ-Centered and Monkhorst Pack grids contained the greatest number of k-points. The $4\times8\times8$ Γ-Centered grid is arbitrarily used here as a basis for comparison to determine the accuracy of other k-point grids. An average relative error was calculated in all of the above force curves to show their deviation from the Γ-Centered $4\times8\times8$ curve.

As seen in Table 2.1 and Figure 2.2, the Monhorst Pack and Γ-Centered k-point grids yielded force curves that are nearly identical in behavior and relative error. A convergent force curve was created with either mesh generation technique, provided a sufficient number of k-points were used.

Although they make for a much quicker calculation, the high symmetry and mean-value k-point configurations, including the Γ-point and the Baldereschi mean value point, did not produce convergent force curves.

Table 2.1: 32 atom Lithium system ($4 \times 2 \times 2$)

| K-Point Grid | Stopping Power ($eV/\AA$) | % Error Stopping Power | % Error Force |
|---|---|---|---|
| Γ-Centered | | | |
| $1 \times 2 \times 2$ | 5.349 | 2.94 | 10.9 |
| $2 \times 4 \times 4$ | 5.185 | 0.206 | 0.855 |
| $3 \times 6 \times 6$ | 5.207 | 0.207 | 0.734 |
| $4 \times 8 \times 8$ | 5.196 | 0.0 | 0.0 |
| Monkhorst Pack | | | |
| $1 \times 2 \times 2$ | 5.207 | 0.215 | 11.4 |
| $2 \times 4 \times 4$ | 5.195 | 0.0205 | 1.06 |
| $3 \times 6 \times 6$ | 5.187 | 0.179 | 2.19 |
| $4 \times 8 \times 8$ | 5.197 | 0.0294 | 0.373 |
| Single K-Point | | | |
| Γ:(0,0,0) | 5.107 | 1.72 | 17.5 |
| (0.5, 0.25, 0.25) | 5.126 | 1.34 | 11.2 |
| (0.0, 0.25, 0.25) | 5.202 | 0.126 | 11.3 |
| BMV:(0.25, 0.25, 0.25) | 5.15 | 0.883 | 13.1 |

**3. Gauge Correction.** In initial simulations, the total charge decreased at a constant rate, and the system would lose .07 to .09 electrons. A gauge correction was implemented in the VASP source code to correct this, holding the number of electrons in the system constant throughout the simulation. In a modified version of VASP, it is actived with a 'LIONGAUGE' flag in the the INCAR file. Figure 3.1 shows typical behavior in this regard with and without the gauge correction for comparison.

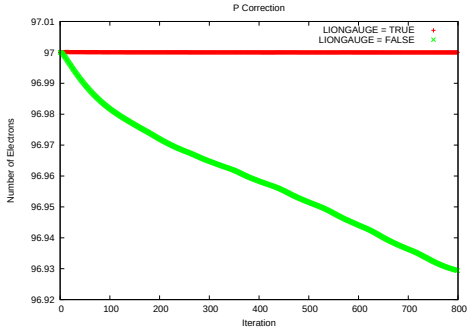



Fig. 3.1: number of electrons in a supercell with and without the gauge correction

Simulations were run with the same parameters as in Section 2 using Monkhorst Pack and single k-point configurations, this time with the gauge correction turned off. The contrast in force curves with and without the gauge correction enabled is quantified in Tables 3.1 and 3.2. Table 3.2 shows the error for each force curve without an enabled gauge correction relative to the analogous force curve with the gauge correction. In every case, the gauge correction affects the force curve by less than 4%. Table 3.1 shows error in force relative to the same Γ-Centered $4 \times 8 \times 8$ force curve used as a basis for comparison in Section 2. The errors do not differ very much from those depicted in Table 2.1 for corresponding k-point grids.

Table 3.1: 32 atom Lithium values without gauge correction, relative to $4 \times 8 \times 8$ Γ-Centered grid with gauge correction enabled

| K-Point Grid | Stopping Power ($eV/\AA$) | % Error Stopping Power | % Error Force |
|---|---|---|---|
| (0.25,0.25,0.25) | 5.02 | 3.4 | 14.1 |
| (0,0,0) | 4.98 | 4.1 | 19.0 |
| $1 \times 2 \times 2$ | 5.07 | 2.3 | 12.8 |
| $2 \times 4 \times 4$ | 5.39 | 3.8 | 3.6 |
| $3 \times 6 \times 6$ | 5.16 | 0.6 | 4.5 |
| $4 \times 8 \times 8$ | 5.16 | 0.6 | 3.7 |

Table 3.2: 32 atom Lithium values without gauge correction, relative to values with gauge correction enabled

| K-Point Grid | % Error Stopping Power | % Error Force |
|---|---|---|
| (0.25,0.25,0.25) | 0.9 | 3.6 |
| (0,0,0) | 1.7 | 3.8 |
| $1 \times 2 \times 2$ | 0.2 | 3.8 |
| $2 \times 4 \times 4$ | 0.02 | 3.6 |
| $3 \times 6 \times 6$ | 0.2 | 3.6 |
| $4 \times 8 \times 8$ | 0.03 | 3.6 |

**4. Computational Speed.** In this Section we explore the effects on computational speed of varying band parallelization. Band parallelization dictates the distribution of bands across the compute nodes used to run the simulation. The computer cluster *Redsky* at Sandia National Laboraties was used to obtain the following data. It contains eight 2.93 GHz Intel Nehalem cores per compute node. Communication amongst nodes during a simulation, and thus its parallelization scheme play a pivotal role in its completion time. In VASP, the parameter that determines the number of bands treated in parallel is called NPAR.

Varying values of NPAR were used with 32 atom, 90 atom, and 192 atom Lithium systems with aspect ratios of $4 \times 2 \times 2$, $3 \times 5 \times 5$, and $6 \times 4 \times 4$ respectively. In each case, short simulations were run in order to gather time data for 5 - 20 force calculaton steps in varying numbers of cores and values of NPAR. Results can be seen in Table 4.1 and Figure 4.1.

The data in Table 4.1 on the 32 atom system shows a typical curve that is representative of the general behavior of all of the tested systems in varying values of NPAR. A minimum time value exists at NPAR = 32, in between the extreme values of NPAR = 4 and NPAR = 256. The NPAR curves in the other systems follow a similar behavior, with the exception of the 400 core set up in Figure 4.1.

It clear that computational time grows with system size. Comparison of the 192 and 384 core simulations in Figure 4.1, and of the 400 and 800 core simulations in Figure 4.1 illustrates that increasing the number of cores decreases the computational time for a given system. Comparison of the 800 and 1600 core simulations in Figure 4.1 demonstrates, however, that in number of cores vs. computational time there is a point of diminishing returns, in this case occuring around 800 cores.

Comparison of the LIONGAUGE = TRUE and LIONGAUGE = FALSE values in Figure 4.1 suggests that employing the gauge correction increases computational time by a factor of 2 - 4.

Table 4.1: 32 atom Lithium,$2 \times 4 \times 4$ Monkhorst Pack k-point grid, LIONGAUGE = TRUE

| NPAR | 128 Core Time(s) | 256 Core Time(s) |
|---|---|---|
| 2 | 76.2 | 188.9 |
| 4 | 22.2 | 35.4 |
| 8 | 13.0 | 12.7 |
| 16 | 12.0 | 8.5 |
| 32 | 12.3 | 8.5 |
| 64 | 15.0 | 9.7 |
| 128 | 20.6 | 18.7 |
| 256 | - | 35.4 |

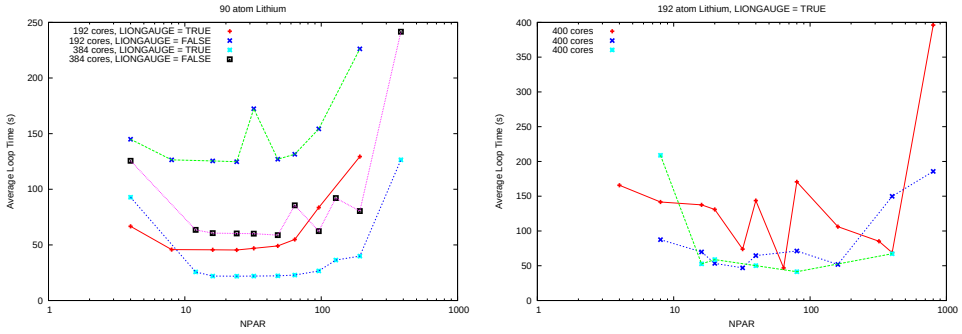



Fig. 4.1: average loop times in the 90 and 192 atom Lithium systems

**5. Stopping Power.**

As mentioned in Section 1, stopping power is a measure of the energy loss by an ion travelling through a system, given by:

$$S = \frac{-dE}{ds} \tag{5.1}$$

To obtain stopping power, a work curve is generated by numerically integrating the dot product of the forces on the proton and an incremental distance it travels during each time step in the simulation:

$$W(s) = \sum_{n=0}^{s} \vec{F_n} \cdot \triangle \vec{s_n} \tag{5.2}$$

The resulting ascending work curve is then linearly fit, as in Figure 5.1. The slope of the linear fit is taken as the approximate stopping power ($dE/ds$) for the system.

**5.1. Effects of Varying K-Point Grids on Stopping Power Calculation.** Using the method described above, stopping power values were calculated for all of the k-point configurations described in Section 2. The results are shown in Tables 5.1, 5.2, and 2.1.

Although some of the k-point grids yielded stabilizing force curves and some did not, all of them yield an almost identical stopping power. In each system size, average relative stopping power error is less than 2% for all k-point meshes compared to their

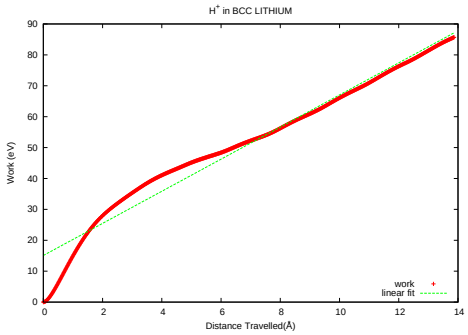



Fig. 5.1: typical work curve for a channeling H atom and its linear fit

respective comparison values. This can probably be attributed to error cancellation in the somewhat equally increasing peak and valley amplitudes in the unstable force curves.

**5.2. Effects of System Size on Stopping Power Calculation.** The effect of system size on stopping power calculations is analyzed here in two ways: with respect to different k-point meshes at a set proton velocity and with respect to a range of proton velocities in all three Lithium systems.

For the former, stopping powers for each k-point mesh in each system size are listed in Tables 2.1, 5.1, and 5.2. It is clear from this data that our calculated stopping power tends to decrease as our system size grows, averaging at $5.19eV/\mathring{A}$ for the 32 atom system, $4.94eV/\mathring{A}$ for the 90 atom system, and $4.77eV/\mathring{A}$ for the 192 atom system. Relative the to 32 atom cell, the average stopping power for the 90 atom and 192 atom cells drops by about 5% and 8% respectively. It can be speculated that this behavior is due to finite size effects of small systems, and that the larger systems are more representative of a single proton traversing the solid.

In the latter method of stopping power analysis described above, calculations are done at proton velocities ranging from 0.5 to 6.5 a.u. in 0.5 a.u. increments and compared for 32, 90, 192, and 350 atom Lithium systems. Stopping power curves produced in each case can be seen in Figure 5.2. The Stopping and Range of Ions in Matter(SRIM) is a software package used to calculate stopping power and related characteristics [12]. The developers of SRIM share calculated stopping power results as well as a best fit to a data base of experimental results. With errors relative to the SRIM best fit reaching a max of about 5%, all of the systems produce stopping power curves that agree well with SRIM. However, a trend to converge to a set value is seen as the system size increases. The difference between the 32 atom system curve and the 90 atom system curve is visibly greater than the difference between the 90 atom curve and the 192 atom curve. The stopping power curve for the 32 atom system differs from that of the 90 atom system by about 14%. Subsequent changes in curves moving from the 90 atom system to the 192 atom system and from the 192 atom system to the 350 atom system yield 4.4% and 6.4% respectively, suggesting that a 90 atom system is large enough for reasonably accurate results. Difference between SRIM experimental values and calulated stopping power values seen in Figure 5.2 may be related to our particular choice of trajectory for the proton.

Table 5.1: 90 atom Lithium

| K-Point Grid | Stopping Power ($eV/\AA$) | % Error Stopping Power | % Error Force |
|---|---|---|---|
| (0.25,0.25,0.25) | 4.899 | 1.14 | 6.95 |
| (0,0,0) | 4.947 | 0.182 | 8.68 |
| 1x3x3 | 4.974 | 0.376 | 6.39 |
| $1 \times 4 \times 4$ | 4.97 | 0.295 | 6.44 |
| $2 \times 3 \times 3$ | 4.956 | 0.0 | 0.0 |

Table 5.2: 192 atom Lithium

| K-Point Grid | Stopping Power ($eV/\AA$) | % Error Stopping Power | % Error Force |
|---|---|---|---|
| (0,0,0) | 4.763 | 0.34 | 3.38 |
| (0.25,0.25,0.25) | 4.766 | 0.29 | 6.75 |
| $1 \times 2 \times 2$ | 4.777 | 0.049 | 0.779 |
| 1x3x3 | 4.780 | 0.0 | 0.0 |

**5.3. Effects of gauge correction on Stopping Power Calculation.** As described in the beginning of this Section, a stopping power was calculated for the systems in Sections 2 and 3. Comparison of Tables 3.1 (no gauge correction) and 2.1 (gauge correction enabled) reveals similar deviations from the Γ-Centered $4 \times 8 \times 8$ k-point mesh used as a basis for comparison in Section 2. Table 3.2 shows the percent change in the stopping power calculation observed when switching off the gauge correction. Errors of less than 2% in every case and less than a tenth of a percent in some cases suggest that the gauge correction has very little effect on the calculated stopping power of a system.

**5.4. Beryllium.** 96 and 64 atom hexagonal close-packed (HCP) Beryllium systems with a lattice constant of $2.29\AA$ and a c/a ratio of 3.58 at a temperature of 300K were simulated to calculate stopping power.

Due to Beryllium's HCP structure, it produces an anisotropic supercell. The proton traversing such a cell does not see the same symmetry that it would in a FCC or BCC cell. To probe the affects on the stopping curve of different trajectories in this system, a proton was set to channel both through the hexagonal layers of the Beryllium system and in between them in two orthogonal paths shown in Figures 6.1, 6.2, and 6.3.

As seen in Figure 5.3, calculated stopping power lies in a range that agrees reasonably well with experimental data provided by SRIM [12]. At higher velotices there is a visible difference between the curves produced by different trajectories, starting at about 1.5 a.u.

**5.5. Aluminum.** A 96 atom face centered cubic (FCC) Aluminum system with a lattice constant of $4.05\AA$ at a temperature of 300K was simulated to calculate stopping power. A proton traversed the cell in both channeled and offchanneled paths, shown in Figure 6.4. Both 3 electron and 11 electron PAWs were used for the Aluminum in these systems. Results can be seen in Figure 5.4.

In the 3 electron PAW, core electrons in the Aluminum ions are not able to be perturbed by a passing proton. In the 11 electron PAW, core electrons are moved to the valence of the Aluminum ions and are able to interact with the system.

Because the forces on the proton due to the ions in the lattice are inversely proportional to the distance between the two, it is expected that forces in the offchanneled path, and thus the stopping power in this path, would be higher than

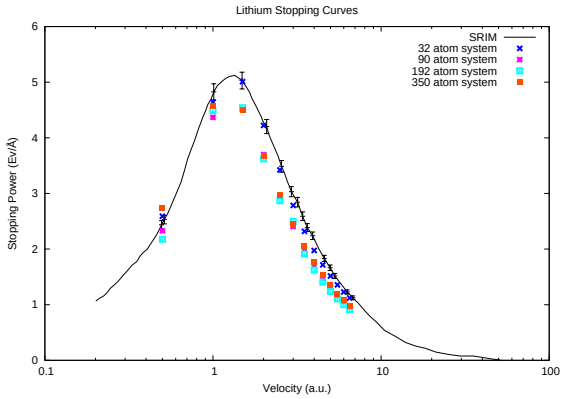



Fig. 5.2: experimental and calculated stopping curves for Hydrogen in Lithium

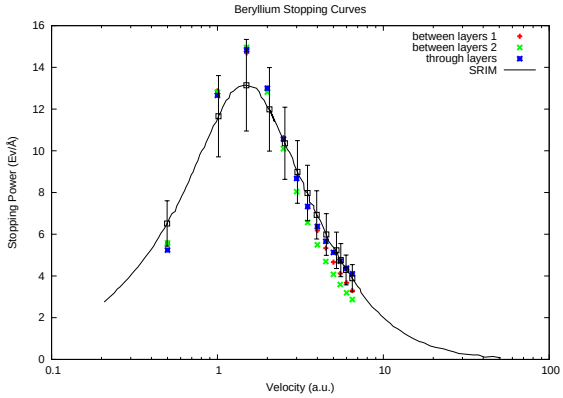



Fig. 5.3: calculated and SRIM stopping power data for Be

those in the channeled path. This is not reflected with the 3 electron PAW, where no siginificant difference in stopping power is seen in the different trajectories. A deficiency of this nature has already been seen in off-channeling in Aluminum at high velocities [3], which prompted us to try a similar experiment with an 11 electron PAW.

A difference in stopping power is clearly seen between the channeled and off-channeled trajectories using the 11 electron PAW at higher velocities, with an increase of around 15% when moving from the channeled to the off-channeled trajectory.

Calculated stopping power for Aluminum did not agree as well with SRIM data is as it did for Lithium and Beryllium. As noted before, it is possible that this disagreement comes from our particular choice of trajectory, which completely avoids ion collisions.

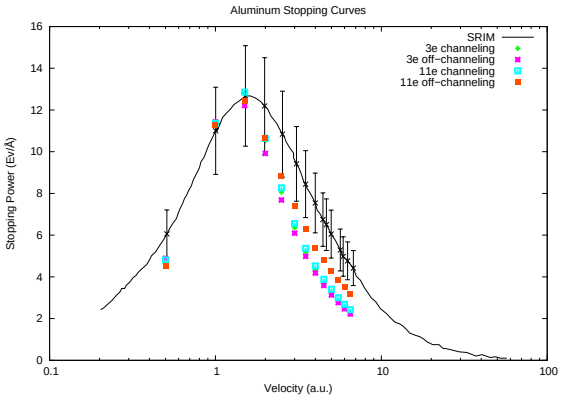



Fig. 5.4: calculated and SRIM stopping power data for Al

**6. Conclusions.** With regard to force convergence, no siginificant difference is seen between Γ-Centered and Monkhorst Pack k-point meshes. The Baldereschi mean-value point and other singularity points tend to be inaccurate.

K-point sampling has very little effect on the stopping power calculations. This can probably be attributed to error cancellation in cases where a particular k-point mesh does not produce a convergent force curve.

As system size increases, stopping power tends to converge. In our case, 90 atom Lithium was large enough to produce results very similar to any larger size.

In Beryllium, it was seen that different trajectories, even in the case where they are all channeling trajectories, can produce different stopping powers. This is particularly apparent in velocities at or above 1.5 a.u.

Sometimes core electrons must be considered in a system to get more accurate results. This was seen to be the case with the off-channeling trajectory at velocities at or above 1.5 a.u. in the Aluminum supercell.

Although charge non-conservation in these simulations is unsettling, it has little to no effect on the data they produce. The implemented gauge correction, while negligably altering force curves and stopping power, increases computational time by a factor of two to four.

With regard to scaling, there exists a cut off point in computational resources above which calculations will not run any faster. The most efficient band parallelization for a particular number of cores tends to lie somewhere below the midway point between the minimum and maximum number of bands that can be treated in parallel.

With an adequate system size, k-point sampling, and PAW, the implemented Ehrenfest-TDDFT capability used herein produces stopping power data that agrees reasonably well with experiment for Beryllium and Lithium. This is not the case for Aluminum, but it is again worth noting that disagreement with experiment could arise from our particular choice of channeled trajectory for the proton.

REFERENCES

[1] A. Baldereschi, *Mean-value point in the brillouin zone*, Physical Review B, 7 (1973), pp. 5212–5215.

[2] P. Blochl, *Projector augmented-wave method*, Physical Review B, 50 (1994), p. 17953.

[3] A. Correa, J. Kohanoff, E. Artacho, D.Sanches-Portal, and A.Caro, *Nonadiabatic forces in ion-solid interactions: the inital stages of radiation damage*, Physical Review Letters, 108 (2012), p. 213201.
[4] P. Grabowski, M. Surh, D. Richards, F. Graziani, and M. Murillo, *Molecular dynamics simulations of classical stopping power*, Physical Review Letters, 111 (2013), p. 215002.
[5] G. Kresse and J. Furthmuller, *Efficiency of ab-initio total energy calculations for metals and semiconductors using a plane-wave basis set*, Computational Materials Science, 6 (1996), p. 15.
[6] ———, *Efficient iterative schemes for ab initio total-energy calculations using a plane-wave basis set*, Physical Review B, 54 (1996), p. 11169.
[7] G. Kresse and J. Hafner, *Ab initio molecular dynamics for liquid metals*, Physical Review B, 47 (1993), p. 558.
[8] ———, *Ab initio molecular-dynamics simulation fo the liquid-metal-amorphous-semiconductor transition in germanium*, Physical Review B, 49 (1994), p. 14251.
[9] G. Kresse and D. Joubert, *From ultrasoft pseudopotentials to the projector augmented-wave method*, Physical Review B, 59 (1999), p. 1758.
[10] H. Monkhorst and J. Pack, *Special points for billouin-zone integrations*, Physical Review B, 13 (1976), pp. 5188–5192.
[11] J. Perdew and A. Zunger, *Self-interaction correction to density-functional approximations for many-electron systems*, Physical Review B, 23 (1981), p. 5048.
[12] J. Ziegler and J. Biersack, *Srim - the stopping and range of ions in matter*, Nuclear Instruments and Methods in Physics Research, Section B: Beam Interactions with Materials and Atoms, 268 (2010), pp. 1818–1823.

**Supplemental Figures and Tables.**

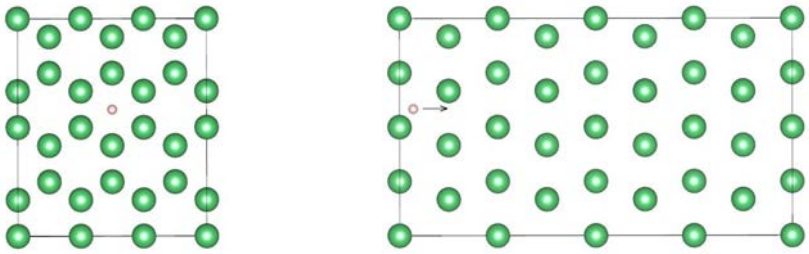


Fig. 6.1: Rear (000) and side (101) views of the proton's path through hexagonal layers in the Be supercell

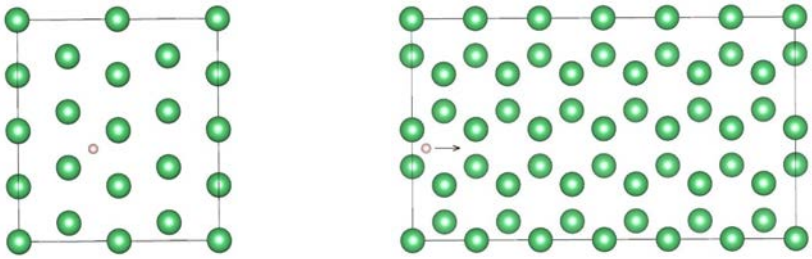


Fig. 6.2: Rear (000) and side (001) views of the proton path 1 between hexagonal layers in the Be supercell

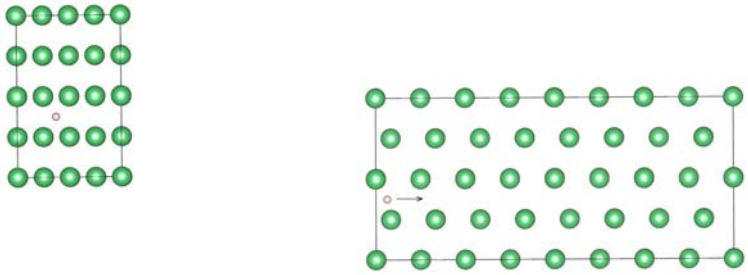


Fig. 6.3: Rear (000) and side (100) views of the proton path 2 between hexagonal layers in the Be supercell

Fig. 6.4: Different trajectories in the Al supercell: the proton's velocity is in the $\otimes$ direction

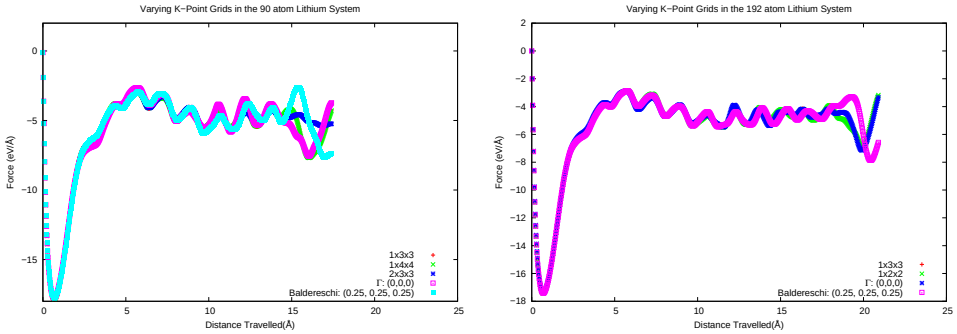



Fig. 6.5: varying k-point grids in 90 and 192 atom Lithium systems

Table 6.1: 90 atom Lithium, 2x3x3 Monkhorst Pack k-point grid

| NPAR | LIONGAUGE = FALSE | LIONGAUGE = TRUE |
|---|---|---|
| 192 Cores | Average Loop Time (s) | Average Loop Time (s) |
| 4 | 66.7 | 144.95 |
| 8 | 45.8 | 126.46 |
| 16 | 45.6 | 125.47 |
| 24 | 45.4 | 124.78 |
| 32 | 46.9 | 172.4 |
| 48 | 49.1 | 127.05 |
| 64 | 54.9 | 131.48 |
| 96 | 83.6 | 154.33 |
| 192 | 129.4 | 226.24 |
| 384 Cores | Average Loop Time (s) | Average Loop Time (s) |
| 4 | 92.8 | 125.6 |
| 12 | 25.7 | 63.5 |
| 16 | 22.0 | 60.6 |
| 24 | 21.9 | 60.3 |
| 32 | 22.0 | 60.1 |
| 48 | 22.2 | 58.8 |
| 64 | 22.9 | 85.6 |
| 96 | 26.6 | 62.4 |
| 128 | 36.4 | 92.1 |
| 192 | 39.9 | 80.4 |
| 384 | 126.5 | 241.6 |

Table 6.2: 192 atom Lithium, 1x2x2 Monkhorst Pack k-point grid, LIONGAUGE = TRUE

| NPAR | 400 Core Time(s) | 800 Cores Time(s) | 1600 Core Time(s) |
|---|---|---|---|
| 4 | 165.8 | - | - |
| 8 | 141.7 | 87.5 | 208.8 |
| 16 | 137.6 | 69.9 | 52.7 |
| 20 | 131.0 | 53.4 | 58.9 |
| 32 | - | 73.9 | 47.0 |
| 40 | 143.7 | 64.6 | 50.1 |
| 64 | - | - | 46.9 |
| 80 | 170.7 | 71.2 | 41.4 |
| 160 | - | 106.2 | 51.9 |
| 320 | - | - | 85.2 |
| 400 | 68.86 | 149.9 | 67.3 |
| 800 | - | 396.2 | 185.7 |

## Applications

Articles in this section discuss the use of computational techniques such as those discussed in the previous section to simulate physical systems. The applications presented here include nuclear reactor simulation, the impact of elastic bodies, and atmospheric computation.

*Dances and Mousseau* discuss the simulation of thermal hydraulics of a light water reactor using the COBRA-TF code. The authors discuss updating COBRA-TF to handle the residual form of the physical conservation equations. The residual form permits greater flexibility in using numerous modern nonlinear solution algorithms. *Kelly et al.* discuss the use of ALEGRA to simulate low density shock physics and multiphysics. Moreover, the authors provide documentation for the RMIN feature in ALEGRA. This feature computes the smallest density in which extreme or unphysical behavior is observed. *Li et al.* introduce an inverse problem for trace-gases using convection-diffusion-reaction physics. They consider both inverting for material coefficients and source terms. Moreover, consider problems with parametric uncertainties in the PDE coefficients. *Staten and Robinson* describe an angle-dependence issue found in ALEGRA. The authors perform extensive testing and have pinpointed the error to the Johnson-Cook fracture model, demonstrating that this fracture model is grid-dependent. *Sullivan and Taylor* employ SQuadGen to generate spherical quadrilateral meshes for athmospheric application. SQuadGen generates meshes to be used by the Community Atmosphere Model. Sullivan and Taylor document a procedure to rotate grids generated by SQuadGen. *Takhtaganov, Keiter, Kouri, and Ridzal* discuss inverse and optimal control problems for electrical circuit models. The authors provide ensight into the computational challenges that arise in solving these systems and demonstrate the performance of the Rapid Optimization Library (ROL) for the numerical solution of such problems.

D.P. Kouri
M.L. Parks

December 18, 2014

# PRELIMINARY RESIDUAL FORMULATION OF COBRA-TF

CHRISTOPHER A. DANCES * AND VINCENT A. MOUSSEAU †

**Abstract.** The thermal hydraulics of a LWR core is an important part of nuclear reactor desigin. COBRA-TF solves 8 conservation equations for liquid, entrained droplet, and vapor phases of water boiling within the rod structure of a LWR reactor core [2]. Currently, the conservation equations analytically reduce into a pressure matrix in a semi-implicit method with rod temperatures solved for explicitly. This work involves representing the 1-D single phase liquid conservation equations and calculated variables in a residual formulation. The full jacobian matrix can then be built numerically, and can then either be reduced to a pressure matrix or solved directly. Verification of the residuals was done by comparing calculated results to analytical solutions for isokinetic advection and shock tube problems. For each verification problem, a scaling study of the truncation error was compared to the predicted behaviour derived from modified equation analysis. Further work was then applied to represent 1-D heat conduction within the heater rods. Some initial work was done to allow the code to solve either semi-implicitly, or fully impicitly.

**Nomenclature.**

**COBRA-TF** Coolant-Boiling in Rod Arrays - Three Fluids
**LWR** Light Water Reactor
**PWR** Pressurized Water Reactor
**BWR** Boiling Water Reactor
**PDE** Partial Differential Equation
**CFL** Courant - Friedrichs - Lewy number
**PETSc** Portable, Extensible Toolkit for Scientific Computation
**P** Pressure
**h** Enthalpy
**u** Velocity
$\rho$ Density
**x** Spatial variable
**t** Temporal variable
**i** Spatial index
**n** Temporal index
**k** Iteration index

**1. Introduction.** For the past several decades, the primary focus in nuclear engineering within the United States has been focused on light water reactors (LWR). Commercially, all nuclear reactors are either boiling water reactors (BWR) or pressurized water reactors (PWR). Correct computation of the thermal hydraulics within the reactor core leads to efficient design and accuracy in the safety analysis. A popular subchannel code for modelling the hydrodynamics with in the reactor core is COBRA-TF. This FORTRAN based code solves 8 conservation equations for liquid, entrained droplet, and vapor phases in 3-D dimmensions [2]. The conservation equations analytically reduce into a pressure matrix in a semi-implicit method with rod temperatures solved for explicitly. Because the physics are integrated into the numerical solution, the equations must be linear and the solution method semi-implicit. With a residual formulation, greater flexibility and control over the numerical solution is possible. COBRA-TF was originally written in FORTRAN 77, but over the years has been partially updated to newer versions of Fortran.

*The Pennsylvania State University, cad39@psu.edu
†Sandia National Laboratories, vamouss@sandia.gov

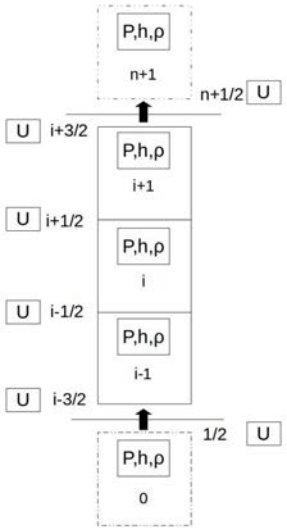



Fig. 1.1: The finite volume structure for COBRA-TF

The finite volume structure in COBRA-TF in figure 1.1 is for a one-dimmensional channel in the axial direction with $n$ number of cells. The first and last cells at 0 and $n+1$ are ghost cells and act as the boundary conditions for the problem. Pressure, enthalpy, and density are averaged over the cell volume and are located at the center of the cell. Mass flow rate and velocity are located at the faces in between cells. The cells are represented with an index $i$, and the faces with indexes of $i+\frac{1}{2}$ or $i-\frac{1}{2}$. This project will initially focus on this 1-D configuration. Usually the code is three dimensional, with channels connecting to each other in two more dimmensions. Fully 3-D equations will be considered in future work.

**2. The Euler Equations.** The single phase Euler partial differential equations for mass (2.1), momentum (2.2), and energy (2.3) corresond to the unknown variables density $\rho$, velocity $u$, pressure $P$, and enthalpy $h$. The first terms in each of the equations are temporal terms. The rest of the terms are steady state spatial terms.

$$\frac{\partial \rho}{\partial t} + \bigtriangledown \rho u = 0 \tag{2.1}$$

$$\frac{\partial \rho u}{\partial t} + \bigtriangledown \rho u^2 + \bigtriangledown P - \rho g = 0 \tag{2.2}$$

$$\frac{\partial \rho h}{\partial t} - \frac{\partial P}{\partial t} + \bigtriangledown (\rho u h) = 0 \tag{2.3}$$

The 1-D formulation of the Euler Equations will assume a direction $x$ as shown in the 1-D mass equation (2.4). The momentum and energy equations are represented in a non-conservative form as shown in equations (2.5) and (2.7). The momentum

equation contains a term that has a product of the left hand side of the 1-D mass equation. This terms can therefore be dropped since it is equivalent to zero, and the entire equation can be divided by density to give a simpler form of the momentum equation (2.6).

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u}{\partial x} = 0 \tag{2.4}$$

$$\rho \frac{\partial u}{\partial t} + u\left(\frac{\partial \rho}{\partial t} + \frac{\partial \rho u}{\partial x}\right) + \rho u \frac{\partial u}{\partial x} + \frac{\partial P}{\partial x} - \rho g = 0 \tag{2.5}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + \frac{1}{\rho}\frac{\partial P}{\partial x} - g = 0 \tag{2.6}$$

$$\rho \frac{\partial h}{\partial t} - \frac{\partial P}{\partial t} + h\frac{\partial \rho}{\partial t} + \rho u \frac{\partial h}{\partial x} + h\frac{\partial \rho u}{\partial x} = 0 \tag{2.7}$$

The 1-D equations are then evaluated at a position index $i$ and a certian time $n$ in order to solve for the next time value of $n+1$. In the mass equation (2.8), the velocities are located at the cell faces $i+\frac{1}{2}$ and $i-\frac{1}{2}$. The density at a corresponding face is either upwinded $\dot{\rho}^n_{i+\frac{1}{2}}$, or averaged $\bar{\rho}^n_{i+\frac{1}{2}}$. In equation (2.9), the derivative $\frac{\partial u}{\partial x}$ is upwinded assuming that the flow is positive. In the energy equation, (2.10) the enthalpy values in the first spatial term are upwinded and shown here assuming a positive velocity. The equation of state (2.11) solves for density assuming that it is a linear combination of changes due to pressure and enthalpy. The partial derivatives in the equation are calculated from steam tables as functions of old time pressure and enthalpy.

$$\frac{\rho^{n+1}_i - \rho^n_i}{\Delta t} + \frac{\dot{\rho}^n_{i+\frac{1}{2}} u^{n+1}_{i+\frac{1}{2}} - \dot{\rho}^n_{i-\frac{1}{2}} u^{n+1}_{i-\frac{1}{2}}}{\Delta x} = 0 \tag{2.8}$$

$$\frac{u^{n+1}_{i+\frac{1}{2}} - u^n_{i+\frac{1}{2}}}{\Delta t} + u^n_{i+\frac{1}{2}}\left(\frac{u^n_{i+\frac{1}{2}} - u^n_{i-\frac{1}{2}}}{\Delta x}\right) + \frac{1}{\bar{\rho}^n_{i+\frac{1}{2}}}\frac{P^{n+1}_{i+1} - P^{n+1}_i}{\Delta x} - g = 0 \tag{2.9}$$

$$\rho^n_i \frac{h^{n+1}_i - h^n_i}{\Delta t} + h^n_i \frac{\rho^{n+1}_i - \rho^n_i}{\Delta t} - \frac{P^{n+1}_i - P^n_i}{\Delta t} + (\rho u)^n_i \frac{h^n_i - h^n_{i-1}}{\Delta x} + h^n_i \frac{\dot{\rho}^n_{i+\frac{1}{2}} u^{n+1}_{i+\frac{1}{2}} - \dot{\rho}^n_{i-\frac{1}{2}} u^{n+1}_{i-\frac{1}{2}}}{\Delta x} = 0 \tag{2.10}$$

$$\rho^{n+1}_i - \rho^n_i = \left(\frac{\partial \rho}{\partial P}\right)\left(P^{n+1}_i - P^n_i\right) + \left(\frac{\partial \rho}{\partial h}\right)\left(h^{n+1}_i - h^n_i\right) \tag{2.11}$$

**3. Residuals and Jacobian Construction.** A residual is simply the difference between the value at some future time $n+1$ and the value at the current iteration $k$. This can be applied to desired variables as shown in equations (3.1), (3.2), (3.3), and (3.4). Residuals can also be applied to the conservation equations by substituting the definition of the residual variables into the conservation equations. This will effectively change any variables evaluated at $n+1$ to $k$. Each cell will have three residual variables and three residual equations. For the entire solution, we will then have a residual variable array $\delta X$, and a residual function array $F(X)$ which defines a linear system as seen in equation (3.5).

$$\delta P_i = P_i^{n+1} - P_i^k \tag{3.1}$$

$$\delta h_i = h_i^{n+1} - h_i^k \tag{3.2}$$

$$\delta u_{i+\frac{1}{2}} = u_{i+\frac{1}{2}}^{n+1} - u_{i+\frac{1}{2}}^k \tag{3.3}$$

$$\delta \rho_i = \rho_i^{n+1} - \rho_i^k \tag{3.4}$$

$$J\delta X = -F(X) \tag{3.5}$$

The Jacobian matrix is defined in equation (3.6) as the derivative of each response of the function $F_j$ with respect to each variable $X_i$. The derivative can be calculated numerically as shown by equation (3.7) where $\epsilon$ is a small numerical value. For COBRA-TF the equations are linear, and this numerical approximation of the Jacobian matrix is exact. This should produce the same jacobian matrix that COBRA-TF currently generates analytically.

$$J_{i,j} = \frac{\partial F_j(X)}{\partial X_i} \tag{3.6}$$

$$J_{i,j} \approx \frac{F_j(X_i + \epsilon) - F_j(X)}{\epsilon} \tag{3.7}$$

To build the jacobian matrix, an object oriented class was created that contains three arrays. An array that points to the residual functions, an array that points to the position within a target variable arrray, and an array that has the index that the function is to be evaluated at. These lists can be appended to in any order, but have to be appended all at the same time so that variables and functions must correspond with each other. Then to construct the jacobian matrix, the residual function and residual variable arrays can each be looped over to numerically build the jacobian matrix as seen in figure 3.1.

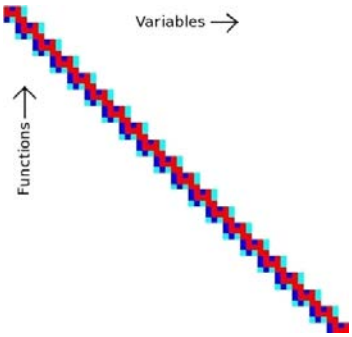



Fig. 3.1: Strucutre of the jacobian matrix for single phase liquid

## 4. Single Phase Liquid Verification Problems.

**4.1. Isokinetic Advection.** A tube with no gravity acting in the direction of fluid flow has an initial condition of $U_1, \rho_1, h_1, P_1, \dot{m}_1$ everywhere except at the starting position that has an intitial conditions of $h_2, \rho_2, \dot{m}_2$. When the time step for the calcuation is taken be exactly equal to the CFL number as seen in equation 4.1, the inlet conditions should advect through the rest of the system in the form of a square wave. This is a unique situation where the CFL can be held constant throughout the simulation, and where the spatial and temporal trucation error can cancel each other out at $CFL = 1$. When the CFL is less than 1, numerical diffusion occurs based on the truncation terms produced by the modified equation analysis.

$$CFL = \frac{\Delta t U_0}{\Delta x} \tag{4.1}$$

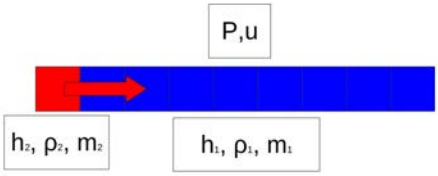



Fig. 4.1: Setup for the isokinetic advection problem

**4.1.1. Density Advection and Error.** Figure 4.2 compares the analytical and numerical advection of density through the domain for a CFL number of 0.500. The higher density in the colder region is on the left, and the lower density of the warmer region is on the right. The red line on the right of the figure depicts the truncation error at the current time step. The truncation error occurs around the original dicontinuity as it advects through the solution. As the discontinuity propogates it becomes more diffuse spatially. Once it reaches the outlet the discontiuty leaves the domain and the overall error drops to nearly zero. For this problem, the truncation error can be shown to be a direct function of the CFL number and can be reduced to nearly zero throughout the simulation. This simplified problem with an exact solution is used for code verification.

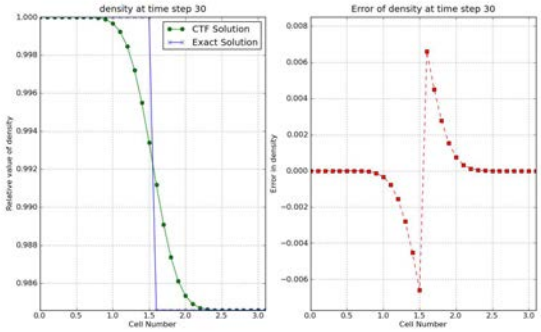



Fig. 4.2: Comparison of density advection to analytical solution

**4.1.2. Modified Equation Analysis.** For this isokinetic problem, the original mass balance equation can be re-written to look like equation 4.2. Using upwinding, the finite difference can be written to look like equation 4.3. A second order Taylor series approximation can be used for $\rho_i^{n+1}$ and $\rho_{i-1}^n$ as shown in equations 4.4 and 4.5 respectively. The higher order terms ($O(\Delta x^2, \Delta t^2)$) are not taken into account for this approximation. The Taylor series approximations can then be substituted into 4.3 to yield 4.6. This is the beginning of the modified equation analysis. The goal will be to isolate the original PDE and define the truncation error.

$$\frac{\partial \rho}{\partial t} + U_0 \frac{\partial \rho}{\partial x} = 0 \tag{4.2}$$

$$\frac{\rho_i^{n+1} - \rho_i^n}{\Delta t} + U_0 \frac{\rho_i^n - \rho_{i-1}^n}{\Delta x} = 0 \tag{4.3}$$

$$\rho_i^{n+1} = \rho_i^n + \frac{\partial \rho}{\partial t}\Delta t + \frac{1}{2}\frac{\partial^2 \rho}{\partial t^2}\Delta t^2 + O(\Delta t^3) \tag{4.4}$$

$$\rho_{i-1}^n = \rho_i^n - \frac{\partial \rho}{\partial x}\Delta x + \frac{1}{2}\frac{\partial^2 \rho}{\partial x^2}\Delta x^2 + O(\Delta x^3) \tag{4.5}$$

The lengthy equation 4.6 can be reduced to equation 4.7 since the $\rho_i^n$ terms subtract out and the $\Delta t$ and $\Delta x$ terms in the denominator cancel out. This reduced equation can the be re-written into equation 4.8, with the original PDE followed by the truncation terms. Notice how the terms on the right are dependent on both the numerical spacing $\Delta t$ and $\Delta x$, but also on the second derivatives of density with respect to space and time.

$$\frac{\left(\rho_i^n + \frac{\partial \rho}{\partial t}\Delta t + \frac{1}{2}\frac{\partial^2 \rho}{\partial t^2}\Delta t^2\right) - \rho_i^n}{\Delta t} + U_0 \frac{\rho_i^n - \left(\rho_i^n - \frac{\partial \rho}{\partial x}\Delta x + \frac{1}{2}\frac{\partial^2 \rho}{\partial x^2}\Delta x^2\right)}{\Delta x} + O(\Delta x^2, \Delta t^2) = 0 \tag{4.6}$$

$$\frac{\partial \rho}{\partial t} + \frac{1}{2}\frac{\partial^2 \rho}{\partial t^2}\Delta t + U_0\left(\frac{\partial \rho}{\partial x} - \frac{1}{2}\frac{\partial^2 \rho}{\partial x^2}\Delta x\right) + O(\Delta x^2, \Delta t^2) = 0 \tag{4.7}$$

$$\frac{\partial \rho}{\partial t} + U_0\frac{\partial \rho}{\partial x} + \frac{1}{2}\frac{\partial^2 \rho}{\partial t^2}\Delta t - U_0\frac{1}{2}\frac{\partial^2 \rho}{\partial x^2}\Delta x + O(\Delta x^2, \Delta t^2) = 0 \tag{4.8}$$

Before we can procede, we need to take the derivative of the original PDE with respect to space and time as shown in equations 4.9 and 4.10 respectively. These two derivatives can substitute into each other using the common term $\frac{\partial^2 \rho}{\partial x \partial t}$. The second derivatives of density with respect to space and time are therefore related by the velocity squared as shown by equation 4.11.

$$\frac{\partial^2 \rho}{\partial t^2} + U_0\frac{\partial^2 \rho}{\partial x \partial t} = 0 \tag{4.9}$$

$$\frac{\partial^2 \rho}{\partial t \partial x} + U_0\frac{\partial^2 \rho}{\partial x^2} = 0 \tag{4.10}$$

$$\frac{\partial^2 \rho}{\partial t^2} = U_0^2\frac{\partial^2 \rho}{\partial x^2} \tag{4.11}$$

This relationship can then be substituted back into equation 4.8, which can be reduced to equation 4.13 after igonoring the higher order terms. The error depends on the CFL number, the axial spacing, and the second order derivative of density with respect to space. This derivative is what gives the error the characterisitcs of diffusion. When the CFL number is less than one, the error term is negative and the diffusion is dampening. When the CFL number is greater than one, the error term becomes positive, and the accumulation of the error destabilizes the solution.

$$\frac{\partial \rho}{\partial t} + U_0\frac{\partial \rho}{\partial x} - \frac{1}{2}\left(\Delta x U_0\frac{\partial^2 \rho}{\partial x^2} - U_0^2\frac{\partial^2 \rho}{\partial x^2}\Delta t\right) + O(\Delta x^2, \Delta t^2) = 0 \tag{4.12}$$

$$\frac{\partial \rho}{\partial t} + U_0\frac{\partial \rho}{\partial x} - \frac{\Delta x U_0}{2}\frac{\partial^2 \rho}{\partial x^2}(1 - CFL) + O(\Delta x^2, \Delta t^2) = 0 \tag{4.13}$$

Modified eqauation analysis can be applied to the energy balance equation presented in equation 4.14. The energy equation is presented in a form where the momentum equation was substituted in as zero and then divided through by density. The result presented in equation 4.15 is similar in form to the result for the mass balance equation 4.13.

$$\frac{\partial h}{\partial t} - \frac{1}{\rho}\frac{\partial P}{\partial t} + U_0 \frac{\partial h}{\partial x} = 0 \tag{4.14}$$

$$\frac{\partial h}{\partial t} - \frac{1}{\rho}\frac{\partial P}{\partial t} + U_0 \frac{\partial h}{\partial x} - \frac{\Delta x U_0}{2}\frac{\partial^2 h}{\partial x^2}(1 - CFL) = 0 \tag{4.15}$$

**4.1.3. Scaling of Error.** From the modifed equation analysis, the advection problem should prove to be first order accurate in time and space. While holding the CFL number constant, the $\ell 1$-normalize error should scale linearly as a function of axial mesh size and with the second derivative of density with respect to space. However, when performing the scaling study, we find that the error scales to the $\frac{1}{2}$ power instead as shown in figure 4.3.

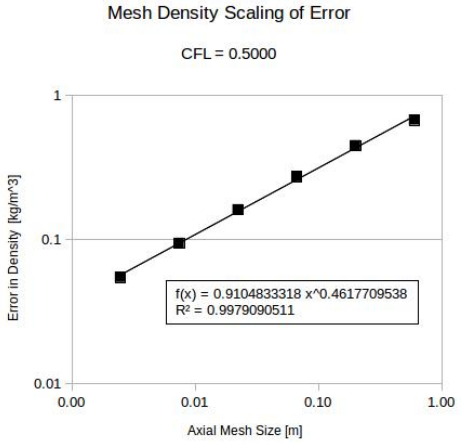



Fig. 4.3: Scaling of numerical error with constant CFL number

Holding the mesh size constant, the time step size can be varied in order to determine the order of accuracy with respect to time as shown in figure 4.4. As time step size approaches zero, the error linearly approaches a constant value that represents the constant spatial error. This first order accurate behaviour in time matches the modified equation analysis for the isokinetic advection.

Holding the time step size constant, the axial mesh size can be varied in order to determine the order of accuracy with respect to space as shown in figure 4.5. As the axial mesh size decreases, the the error approaches zero with an order of accuracy of $\frac{1}{2}$. This nonlinear behaviour is due to the advection of the square wave, which is a spatial discontinuity. As the mesh size decreases, the second derivative of density with respect to position begetting the nonlinear error distribution. An order of accuracy of $\frac{1}{2}$ in space is expected with discountinuous shocks.

**4.2. Shock Tube.** A shock tube is a very common and standard method of verification for momentum and pressure. However, an exact analytical solution is more readily obtained for an ideal gas such as air. A shock tube is created by setting the initial mass flow rate and velocity to zero with no gravity. The boundary conditions

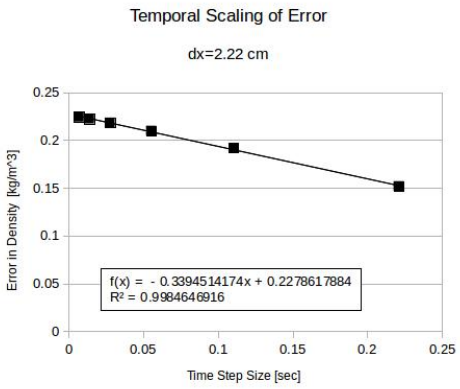



Fig. 4.4: Scaling of numerical error with constant mesh size

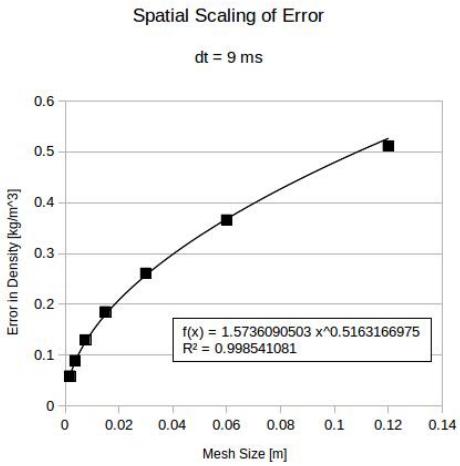



Fig. 4.5: Scaling of numerical error with constant time step size

at the inlet and outlet are also set to zero, simulating a closed system. A region of high pressure is defined for one half of the domain, and a region of low pressure for the second half. An imaginary diaphragm divides the two regions before the simulation, and at $t = 0$ disappears.

Each region has a unique density corresponding to the different pressures using the equation of state for air given in equation 4.16 where $\gamma = 1.4$ is the ratio of specific heats for air. Additionally, the specific heat $Cp = 1.005 \frac{kJ}{kg-K}$ to convert between enthalpy and temperature. The initial enthalpy of the system is constant, but will change non-uniformly as a function of time. The velocity is initially set to zero, but will change as the compression and rarefaction waves move. Since the velocity is set to zero initially, it can't be used to evaluate the time step size. Instead the speed of sound can be evaluated using equation 4.17, and this velocity can be used to calculate the time step. While there might be some slight change in the speed of sound due to

enthalpy changes, it should remain effectively constant.

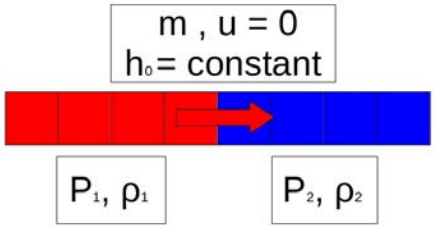



Fig. 4.6: Setup for the shock tube problem

$$\rho = \left(\frac{\gamma}{\gamma - 1}\right) \frac{P_{abs}}{h} \tag{4.16}$$

$$a = \sqrt{(\gamma - 1)\, h} \tag{4.17}$$

$$\gamma = \frac{C_p}{C_v} \tag{4.18}$$

**4.2.1. Analytical Solution.** When the diaphragm disappears, a compression wave will move to the right and a rarefaction wave to the left. These two waves split the domain into four distinct regions. There is a region to the left of the rarefaction wave that has the same properties as the initial left region. There is a region between the rarefaction wave and the initial location of the diaphragm. There is a region between the initial location of the diaphragm and the compression wave. There is a region to the right of the compression wave, that has the same properties as the initial right region. The analytical solution does not take into account reflection off of the walls, however the numerical solution can due to the applied boundary conditions for mass flow rate.

For a perfectly caloric gas, the following equations are provided [1, p. 238] given the initial conditions for state 1 and 4 in conjuction with equation 4.17. An iterative method is required to solve 4.19 for $\frac{P_2}{P_1}$. Once the region properties are obtained, the regions themselves are mapped by comparing the current position and time to the velocity of the rarefaction and compression wave.

$$\frac{P_4}{P_1} = \frac{P_2}{P_1} \frac{\left(1 - (\gamma - 1)(\frac{a_1}{a_4})(\frac{P_2}{P_1} - 1)\right)^{-\frac{2\gamma}{\gamma - 1}}}{\sqrt{2\gamma\left[2\gamma + (\gamma + 1)(\frac{P_2}{P_1} - 1)\right]}} \tag{4.19}$$

$$\frac{T_2}{T_1} = \frac{P_2}{P_1}\left(\frac{\frac{\gamma+1}{\gamma-1} + \frac{P_2}{P_1}}{1 + (\frac{\gamma+1}{\gamma-1})\frac{P_2}{P_1}}\right) \tag{4.20}$$

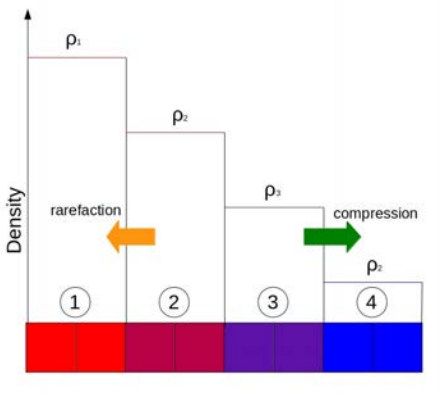



Fig. 4.7: Regions within the shock tube based on rarefaction and compression

$$\frac{\rho_2}{\rho_1} = \frac{1 + (\frac{\gamma+1}{\gamma-1})\frac{P_2}{P_1}}{\frac{\gamma+1}{\gamma-1} + \frac{P_2}{P_1}} \tag{4.21}$$

$$W = a_1\sqrt{\frac{\gamma+1}{2\gamma}\left(\frac{P_2}{P_1} - 1\right) + 1} \tag{4.22}$$

$$P_2 = P_3 \tag{4.23}$$

$$\frac{P_3}{P_4} = \left(\frac{\rho_3}{\rho_4}\right)^{\gamma} = \left(\frac{T_3}{T_4}\right)^{\frac{\gamma}{\gamma-1}} \tag{4.24}$$

**4.2.2. Results and error.** Enthlapy and density have a discontinuity as seen in figure 4.8 where the diaphragm was initially placed . Similarly there are discontinuities that move with the rarefacation and compression waves for pressure, density, and velocity. The largest error occurs around these discontinuities are difficult to measure spatially as seen in figure 4.9, and require a fine mesh to approach the exact solution.

**4.2.3. Modified Equation Analysis.** The primary equation for this problem is the momentum equation (2.9) and the target variable velocity $u$ as shown in (4.25). The taylor series expansions for velocity with respect to space (4.27), velocity with respect to time (4.28) (4.29), and pressure with respect to space pressure (4.26) are needed for this analysis. Subsituting these approximations back into the momentum equation produces eqution (4.29) that contains the original PDE on the left and the truncation error terms on the right.

$$\frac{u^{n+1}_{i+\frac{1}{2}} - u^{n}_{i+\frac{1}{2}}}{\Delta t} + u^{n}_{i+\frac{1}{2}}\left(\frac{u^{n}_{i+\frac{1}{2}} - u^{n}_{i-\frac{1}{2}}}{\Delta x}\right) + \frac{1}{\bar{\rho}^{n}_{i+\frac{1}{2}}}\frac{P^{n+1}_{i+1} - P^{n+1}_{i}}{\Delta x} = 0 \tag{4.25}$$

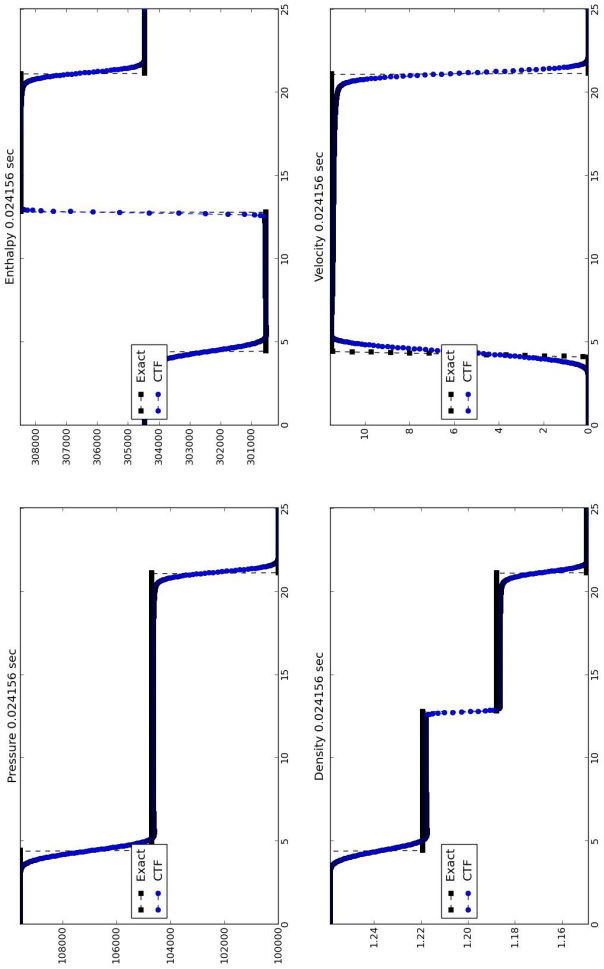



Fig. 4.8: Comparison of analytical and numerical results for shock tube

$$P_i^{n+1} = P_{i+1}^{n+1} - \Delta x \frac{\partial P}{\partial x} + \frac{\Delta x^2}{2} \frac{\partial^2 P}{\partial x^2} + o(\Delta x^3) \tag{4.26}$$

$$u_{i-\frac{1}{2}}^{n} = u_{i+\frac{1}{2}}^{n} - \Delta x \frac{\partial u}{\partial x} + \frac{\Delta x^2}{2} \frac{\partial^2 u}{\partial x^2} + o(\Delta x^3) \tag{4.27}$$

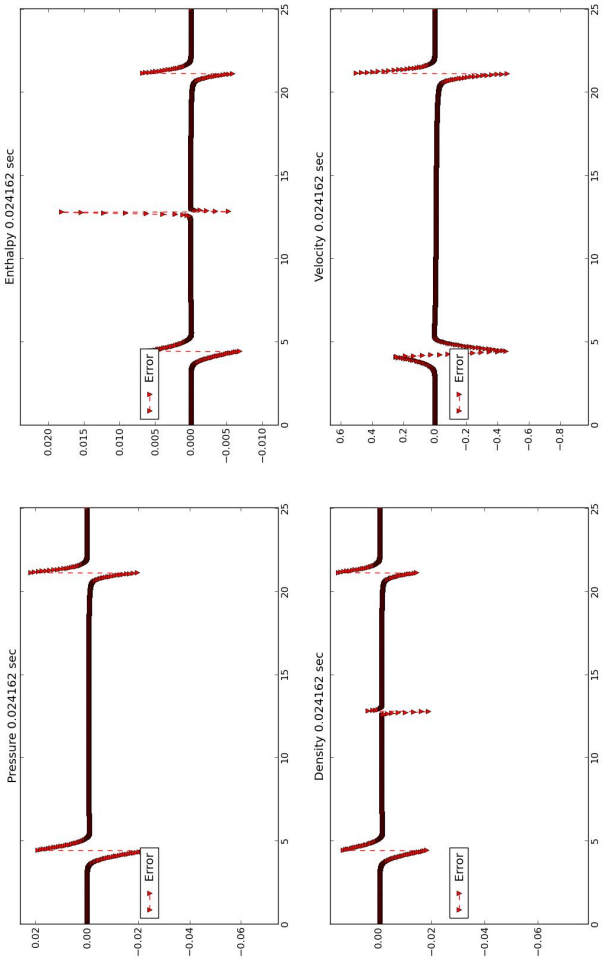



Fig. 4.9: Truncation error for shock tube

$$u^n_{i+\frac{1}{2}} = u^{n+1}_{i+\frac{1}{2}} - \Delta t \frac{\partial u}{\partial t} + \frac{\Delta t^2}{2}\frac{\partial^2 u}{\partial t^2} + o(\Delta t^3) \tag{4.28}$$

$$\left[\frac{\partial u}{\partial t} + u^n_{i+\frac{1}{2}}\frac{\partial u}{\partial x} + \frac{1}{\overline{\rho}^n_{i+\frac{1}{2}}}\frac{\partial P}{\partial x}\right] - \frac{1}{2}\left[\Delta t\frac{\partial^2 u}{\partial t^2} + u^n_{i+\frac{1}{2}}\Delta x\frac{\partial^2 u}{\partial x^2} + \frac{\Delta x}{\overline{\rho}^n_{i+\frac{1}{2}}}\frac{\partial^2 P}{\partial x^2}\right] + o(\Delta t^2, \Delta x^2) = 0 \tag{4.29}$$

**4.2.4. Scaling of Error.** For the first scaling study, the mesh spacing was held constant and the time step size was varied. The scale of the $\ell 1$-normalize error in figure 4.10 shows the problem to be first order accurate in time. Due to the spatial discontinuities, the problem is $\frac{1}{2}$ order in space as shown in figure 4.11.

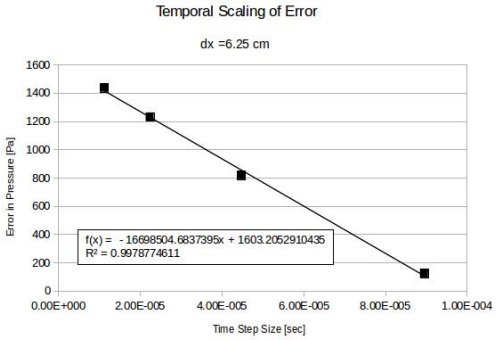



Fig. 4.10: Temporal scaling of the truncation error for the shock tube problem

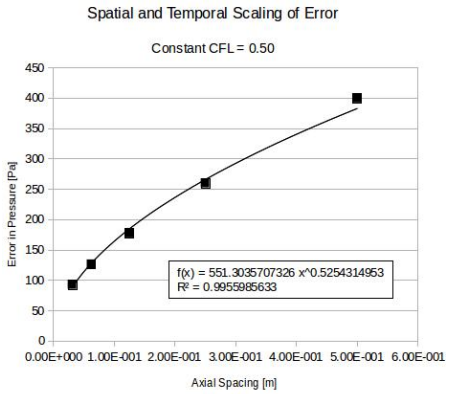



Fig. 4.11: Mesh scaling of the truncation error for the shock tube problem

**5. Coupling to solid equations.** COBRA-TF also explicitly solves for the temperatures within the rods. Empirical correlations determine the heat transfer coefficient between the solid and the fluid. The heat conduction in the rods would normally be dependent on the axial, radial, and azimuthal position of the rod. The scope of this project focuses just on the radial conduction at each axial level. There will be no azimuthal conduction since the fluid flow in this scope is 1-D. The temperature gradients in the axial direction will be heavily dependent on the axial fluid flow. The specific heat capacity and thermal conductivity property tables in the input were held constant as a function of temperature for the following problems. The solid nodes are modelled as uniform heater rods with no cladding and gap.

**5.1. Conduction Equations.** For this project, the equation for a 1-D slab (5.1) was used instead of using cylindrical coordinates.

$$Mc_p\left(\frac{\partial T_{i,solid}}{\partial t}\right) - A\left(\frac{\partial k_{solid}T_{i,solid}}{\partial x}\right) - Q + A_{surf}h_{ht}\left(T_{w,solid} - T_{liq}\right) = 0 \quad (5.1)$$

The first term is the thermal mass of the control volume multiplied by the rate of change of the temperature.The next term describes the conduction to adjacent control volumes. If the cell face is at the center, then the gradient through the face is zero to account for symmetry within the control volume. The area is taken as the cross sectional area at the middle of the cell. The next term is the heat generated within the region. The last term is the heat transfer out of the region due to convection. If the cell is not adjacent to the fluid, then this term gets dropped by setting the heat transfer coefficient to zero. This same term must be added to the single phase liquid energy residual to conserve energy.

$$M_i c_{p_i}\left(T_i^n\right)\left(\frac{\partial T_{i,solid}}{\partial t}\right) - A\left(\frac{\partial k_{solid}T_{i,solid}}{\partial x}\right) - Q + A_{surf}h_{ht}\left(T_{w,solid} - T_{liq}\right) = 0 \quad (5.2)$$

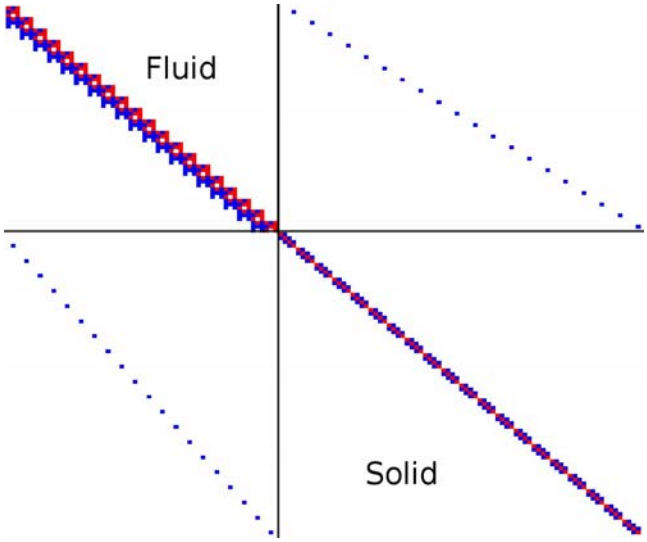



Fig. 5.1: Jacobian matrix for fluid and solid residuals

When the conduction equation (5.1) is written as residual function, it can be appended to the Jacobian structure as shown in figure 5.1. The original fluid jacobian matrix is in the top left and has a block entry for each fluid cell. The solid jacobian matrix is in the bottom right, where each row corresponds to a single solid energy balance. Each block in the diagram represents an axial level with 4 radial positions. The first entry only has two temperatures due to symmetry about the center. The last entry only has two entries since it is interacting with the fluid and the surface temperature is lagging. While this represents the full jacobian matrix, the heat conduction solution is not fully coupled with the single phase liquid solution. This can be seen by the lack of terms in the upper right and lower left quadrants.

**5.2. Initial cooling problem.** An initial test was devised to cool down a rod that had no heat generation but was at a higher initial temperature than the surrounding fluid. The fluid has a constant mass flow rate through the domain and a constant inlet temperature. The rod takes roughly 250 seconds to come to thermal equilibrium as shown in figure 5.2. The rate of cooling is dependent on the radial position as also shown by figure 5.3, where the green values are the starting points and the red values are the end points. Each black line in between represents the raidal profiel at that axial level at an intermediate time step. As the rod cools, the temperature distribution in the radial direction is non-linear. As the rod approaches the inlet temprature, the profile becomes flat.

The initial temperature distribution is not strongly axially dependent as seen by figure 5.4. Near the end of the solution, the rod temperatures demonstrate some small changes in axial direction as shown in figure 5.5. This change is due to the axial temperature gradient experience in the fluid from the inlet to the outlet as it cools down the rod. This trend is not strong from this problem since all of the temperatures are converging to the inlet temperature due to the lack of heat generation in the heater element.

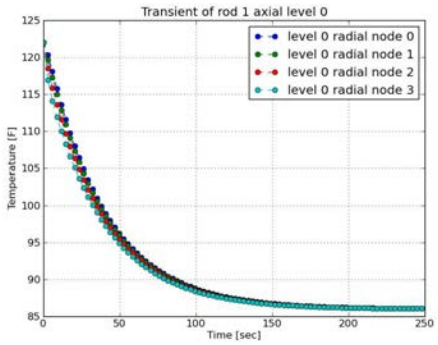



Fig. 5.2: Plot of the temperatures for different positions as functions of time

**5.3. Heat generation problem.** For this test, the rod has a uniform and constant heat generation rate. The initial temperature of the rod matches the initial temperature of the fluid. The temperature of the rod increases to a steady state value as shown by figure 5.6. The radial profile at steady state is non-linear due to the volumetric heat generation within the spacer rod as shown by figure 5.7. The axial temperature profile has a gradient that reflects the temperature change in the fluid from the inlet to the outlet.

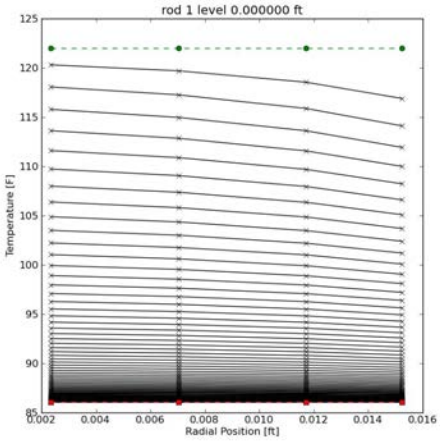



Fig. 5.3: Plot of the temperatures for different times as functions of position

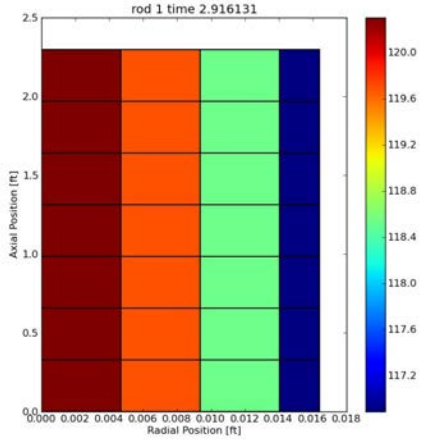



Fig. 5.4: Plot of the temperatures for different times as functions of position

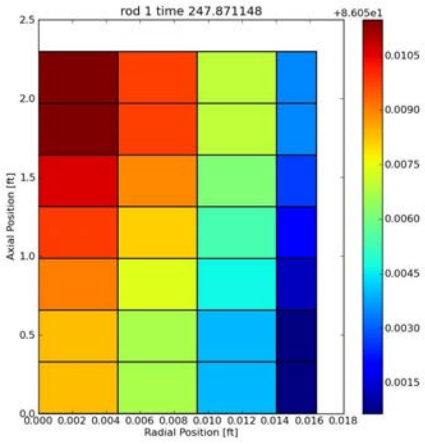



Fig. 5.5: Plot of the temperatures for different times as functions of position

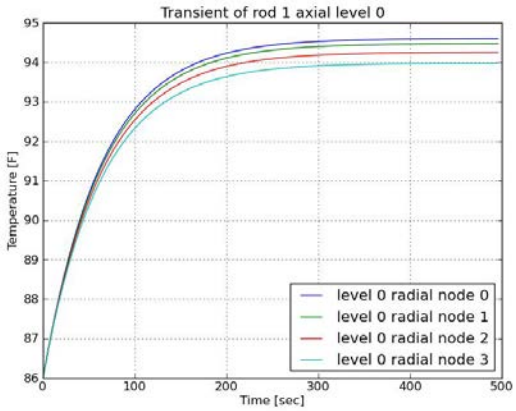



Fig. 5.6: Plot of the temperatures for different positions as functions of time

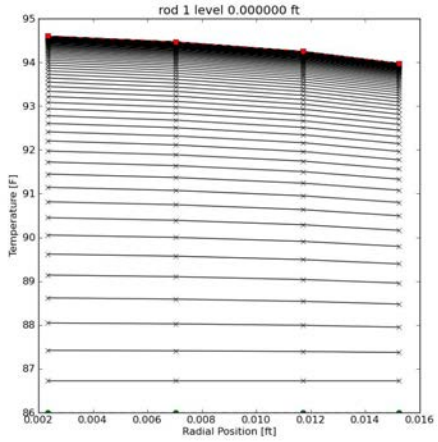



Fig. 5.7: Plot of the temperatures for different times as functions of position

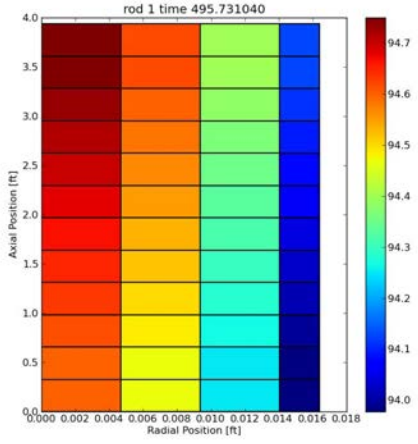



Fig. 5.8: Plot of the temperatures for different times as functions of position

**6. Conclusions.** The project succesfully contructed a method within COBRA-TF to numerically build a jacobian matrix given residual functions and variables. Single phase 1-D residuals were written into COBRA-TF and verified using two different verification problems. Both showed that the problem was first order accurate in time and $\frac{1}{2}$ order accurate in space due to the shocks being modeled. An initial frame work for modeling the solid conduction in the rods was developed and quantitatively shown to model correctly with and without heat generation.

Further work for the verification of the code will be the advection of a smooth function, such as a sine wave. This should allow for a more reliable determination for the order of accuracy of the probelm spatially. Future work with the fluid residuals will entail adding form losses such as fiction and area changes into the residual equations. The residual formulation could also be written so as to include vapor and entrained liquid phases with multiple dimensions. Future work with the conduction equations will be applying cylindrical coordinates for all of the rod geometry types. Currently the code can tentatively solve the 1-D single phase fluid equations and the 1-D conduction equations implicitly. However, the heat transfer coefficeint was still being lagged and was not implicitly solved for. Future work will involve calculating the heat transfer coefficient implicitly. This will add significantly more cross terms to the coupled fluid-solid jacobian matrix. Future work might also involve having the option the add the fluid and solid equations together so as to build a homogenous fluid-solid equilibrium model.

REFERENCES

[1] J. D. Anderson, *Modern Compressible Flow With Historical Perspective*, McGraw-Hill Inc., 1990.
[2] R. K. Salko, *CTF Theory Manual*, tech. rep., The Pennsylvania State University, July 2014.

# ALEGRA AND EQUATION OF STATE TABLES

BRIAN M. KELLEY*, SHARON V. PETNEY†, AND DAVID M. HENSINGER‡

**Abstract.** ALEGRA [2] is a family of shock and multiphysics simulation codes. ALEGRA can simulate a large variety of materials and scenarios, in a wide range of densities, temperatures and pressures. At very low densities and pressures, the materials' state variables can exhibit physically impossible relationships, such as a negative change in pressure over a positive change in density. In order to remove the unphysical data from the data set, one must first determine the least extreme value (i.e. the greatest density) where the extreme behavior is observed. This point is called RMIN; it is the minimum valid density ($\rho$) in the data set.

**1. Introduction.** ALEGRA is a family of physics simulation software that is developed, maintained and distributed by Sandia National Laboratories. Some examples of what ALEGRA codes are capable of simulating are high-speed collisions, hydrodynamic scenarios, and magnetic field interactions. To achieve accurate results with a finite amount of computing power, ALEGRA creates an abstract spatial mesh, which contains many small elements. The elements can be thought of as sample points whose characteristics reasonably represent some amount of the surrounding area. This technique is called the Finite Element Method (FEM). The simulation is performed by determining the interactions between sets of neighboring elements, treating each as an indivisible unit. The user can select from a wide variety of materials, and initialize them with the desired phase (solid, fluid, vapor) and state variables (temperature, pressure, etc.) These state variables are the primary way of quantifying the interactions between mesh elements. As the simulation progresses, ALEGRA solves differential equations (equations of state, abbreviated EOS) that govern the relationships between these state variables.

Most of the time, ALEGRA executables produce accurate, realistic data that could be verified with a real-world experiment. However, certain unusual circumstances, such as very low densities and pressures, can cause ALEGRA to produce unphysical (physically impossible) relationships between state variables. For example, it may produce an EOS output table that suggests that

$$dP/d\rho < 0 \tag{1.1}$$

where $P$ is pressure and $\rho$ is density. In other words, the data in the equation of state table says that the derivative of pressure with respect to density is negative. RMIN ($\rho_{min}$) is the largest density quantity for which Equation 1.1 is true. When observing isotherms in pressure vs. density space, all data points with densities less than RMIN should be discarded, to ensure that the data set does not contain any unphysical state variable relationships. The primary tool for viewing ALEGRA output data sets is SHIV (Specialized HIstory Viewer), which can color segments of isotherms based on the sign of their slopes (facilitating the manual determination of RMIN). However, SHIV is unable to distinguish between negative slopes of sufficient magnitudes to be of interest, and those with very small magnitudes caused solely by floating point round-off error. If one of these slightly downsloping isotherm segments occurs at a greater density then the desired RMIN, SHIV will report that as RMIN. If such an RMIN

*Albuquerque Academy, kelb150@aa.edu
†Sandia National Laboratories, svpetne@sandia.gov
‡Sandia National Laboratories, dmhensi@sandia.gov

value is used to determine what portion of the table is valid (i.e. not unphysical), an unnecessarily large portion of the table will be discarded.

## 2. Projects and Tasks.

### 2.1. RMIN Documentation.

The RMIN documentation was a task that supported the ALEGRA documentation. A large set of Sesame [1] EOS tables existed in the Lambda repository, and many contained unphysical regions (i.e., nonzero RMIN values). However, neither these tables nor their RMIN values were listed in any single document. The purpose of this task was to create such a document, along with various supporting files, in a format that was compact yet readable.

The Sesame tables that needed processing were distributed throughout several different subdirectories. The tables were named after their numerical material ID. One factor that complicated the task of iterating through these files was that multiple folders contained EOS tables with the same file name. Despite referring to the same material (or family of materials), the tables with repeated names contained completely different data, had their own RMIN value, and required their own entries in each list that would be created.

To determine the RMIN values, each table was opened in SHIV and displayed as isotherms on a pressure vs. density plot. In this mode, SHIV also shades each segment of the isotherms based on their slope: red means upsloping, blue means zero slope, and green means downsloping. As viewed in SHIV, RMIN is effectively the rightmost green segment (with a significant downward slope). The "import" command was used to create a snapshot of the isotherm plot in the SHIV window. These images were captured while SHIV was zoomed in to the area where RMIN was located. The SHIV window is also shown displaying the coordinates of the segment endpoint serving as the definition of RMIN. Figure 2.1 shows one of these images; all captured images were saved for inclusion in the final report.

After obtaining this image, the view was further zoomed in. The endpoint provided a higher-precision approximation of RMIN. Once at least three significant figures of the value had been determined, the actual value was located in the Sesame table with a text editor (e.g., vi). After performing the conversion from SI units to CGS (conversion factor of $10^{-3}$), it was possible to find the single, exact RMIN value. With the value in the table highlighted, another window snapshot was taken with the "import" utility.

Finally, this value was copied to a text file called "rmin.txt," along with the table's material ID number and directory name. These two pieces of information eliminated any ambiguity about different tables with the same name in different directories. In this file, RMIN is included twice. The value preceded by 'c' is the "candidate" value presented by SHIV (in SI units) and the value after 't' is the value directly from the table (in CGS units). Both were included in order to handle cases where the correct RMIN value is different from the SHIV candidate.

Here is a sample line from rmin.txt:

```
3331 aneos: c 8107.65553 t 8.10765553E+00
```

After populating rmin.txt with data about each EOS file, another plain text list was created. This file was named "Rmin_Values.txt." It was designed to include all the

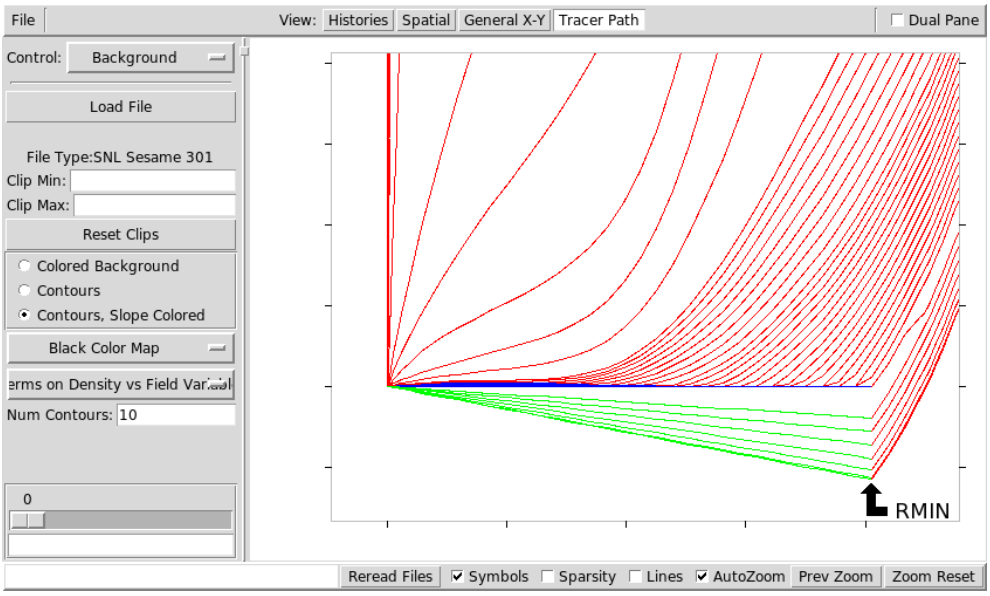



Fig. 2.1: Example view of SHIV displaying RMIN region.

important data fields in a relatively compact manner, and to be a useful supplement to the main PDF in the ALEGRA documentation.

This is the heading for "Rmin_Values.txt":

```
#DIRECTORY  MATID  RMIN(CGS)  RMIN(SI)  CGS-RND  EOS_Data  MATERIALNAME  SR
#-------------------------------------------------------------------------
```

CGS-RND is the value of RMIN (in CGS units), with four significant figures. The last digit is always rounded up. Rounding up ensures that no densities greater than RMIN exhibit the unphysical negative $dP/d\rho$. The SR column contains the scaling ratio for all density values in the table. The RMIN values were multiplied by their corresponding scaling ratios when Rmin_Values.txt was generated. The EOS_Data column either contains a "Y" or an "N," depending on which list the material's name was found: either "SES_EOS_data" or "EOS". If it came from the file SES_EOS_data, this column will contain "Y"; if it was only listed in EOS, this column contains "N." This is useful for the end user to know because if the name is from SES_EOS_data, it can be used verbatim as a material identifier in ALEGRA input decks. The names from EOS lack this property. Also, the names from SES_EOS_data are written in all capital letters, while the names from EOS are not.

Here is an example line from "SES_EOS_data":

```
'AL2024'   3700  0.96923 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 2.0 0 0 1 !
      'sesame'  1 !
```

and an example from "EOS":

```
Number=    3331
Material = Copper (Z=29.0, A=63.546)
File= ./aneos/3331
Tables==>     201        301
```

Because of the large number of EOS tables, it was more efficient to automate the process of creating the necessary text documents and tables. A Python script was created to parse the list SES_EOS_data and extract material names. The Lambda project already contained a script to iterate through every EOS table and extract information about it. It saves its output to a file called "EOS." Because the material names in the list SES_EOS_data can be used verbatim in ALEGRA input decks, they take priority over those only listed in " EOS." The script worked by iterating over every line in rmin.txt, and searching for the material's name in the file SES_EOS_data. If that search came up empty, it then looked in " EOS", which is guaranteed to have an entry for every equation of state table that was processed.

After finding the material's name, the main script had several tasks to perform for each equation of state table. If a table was listed in SES_EOS_data, it checked the scaling ratio (in the above example, the SR is 0.96923). The scaling ratio was then multiplied by the RMIN values read from rmin.txt. After this step, the script calculated the "recommended" RMIN with four significant figures. All of this information was saved to Rmin_Values.txt.

The document that ALEGRA end users are most likely to find useful is a SAND Report PDF file (the SAND Report format can include Offical Use Only markings, which were necessary in this case). The PDF was generated by pdflatex, and included a "quick reference" table immediately after the Table of Contents. The LaTeX code for the table is contained in its own .tex file, which is generated automatically by the Python script. This was the heading for the PDF table:

| Material Name * | Directory | MatID | Sr | RMIN ($g/cm^3$) |
|---|---|---|---|---|

This table is designed to contain only the amount of information the user will need to see at a quick glance. More in-depth information would be available in Rmin_Values.txt if it were needed. It was not feasible to fit every column from Rmin_Values.txt into a table the width of a page and still use a readable font size, etc. The asterisk after "Material Name" is explained in the footer, which says that material names that appear in all capital letters are present in SES_EOS_data. The other names are all lowercase, which makes the capitalization a reliable way of distinguishing between the two types of names.

After the quick reference table, the SAND report contains some textual information and two images for each EOS table. The material's name and all three formats of RMIN (CGS, SI, CGS rounded) are all included and labeled. Below the text are two images: the SHIV zoomed view and the EOS table values with RMIN highlighted.

The text and two images all fit onto one page. Because there are over 400 documented EOS tables, and twice as many PNG images, minimizing the document's length and file size was a major concern. The Python script automated this part of the document as well. The documentation for each EOS table is contained in its own .tex file, all of which are the same except for textual information and image filenames.

In addition to producing Rmin_Values.txt and the necessary LaTeX code for the PDF, the script needed to produce a C header file called Rmin_Values.h. The header's purpose is to create a large array of structs. Each struct contains data about an EOS table: the directory name, material ID, RMIN, name, and hash code of the EOS file. This header will be used to look up information about EOS tables in ALEGRA's input materials. The hash code can be used to detect whether any EOS tables have been modified.

### 2.2. SHIV User Interface Improvements.

Before the RMIN documentation was complete, a few ideas for improving the SHIV user interface were considered:

- Drawing line segments that only partially fit inside the current view clip. Before, only line segments with both endpoints inside the clip were drawn.
- Adding a "Previous Zoom" button that allowed the user to navigate through a history of zoom states.
- Using the right mouse button to pan.
- Using the mouse scroll wheel to zoom in and out.

The graphical user interface of SHIV is managed by Tkinter, the Python version of Tk. Tkinter makes it very simple to build interfaces with a standardized look-and-feel. The way the interface interacts with program logic is event-based; interactive components like buttons and menus simply call a function specified in their constructors when they are clicked. Thanks to this simplicity, it was easy to understand the flow of program logic, and add clean modifications to existing code.

To approach the line segment problem, SHIV's procedure for culling line segments outside the canvas was examined. It loops through a list of all segments available from the data, and tests whether both points (x1, y1) and (x2, y2) were within the left, right, bottom and top limits of the clip. If this check passed, the segment was added to a second list, "clipped." This check and result can be seen in the code below. Two other conditions were inserted after that to catch situations where exactly one of the two endpoints is inside the clip. Inside of the elif block, a third point was calculated where the line segment intersects the boundary of the clip. The intersect() method returns this third point in a tuple (x, y), and works regardless of which edge (left, right, bottom or top) the line intersects. The method had already been written because it was needed elsewhere in the LineSet class. Because of the two separate elif blocks, it is known which point (1 or 2) is inside the clip, and which is outside. With this information, all that needs to be done is append a new tuple to the "clipped" list, containing a line running from the inside point to (x3, y3). It was necessary to calculate the intersection in the first place because the clip defines the area within the axes on the canvas, rather than the entire Tk canvas. There is a small margin between the axes and the edge of the canvas used for drawing axis labels, and it was important that no line segments be drawn in this margin. Finding the third point

prevents this side effect.

```
self.clipped = []
for tup in self.segments:
    x1,y1,x2,y2 = tup
    if L <= x1 <= R and B <= y1 <= T and \
        L <= x2 <= R and B <= y2 <= T:
        # both points are inside the clip box
        self.clipped.append(tup)
    elif L <= x1 <= R and B <= y1 <= T:
        # point 1 is in the clip box but point 2 is not
        x3,y3 = intersect(x1, y1, x2, y2, L, R, B, T, x1, y1)
        self.clipped.append((x1, y1, x3, y3))
    elif L <= x2 <= R and B <= y2 <= T:
        # point 2 is in the clip box but point 1 is not
        x3,y3 = intersect(x2, y2, x1, y1, L, R, B, T, x2, y2)
        self.clipped.append((x3, y3, x2, y2))
```

The previous zoom feature was needed in SHIV because it was impossible for the user to undo zoom actions. Zooming in was easy, but there was no way to zoom out except to reset the clip to the default, which is sized so that all data points are simultaneously visible. A stack was the right type of container to build up a "history" of clip sizes. If the user wants to zoom in, the zoom state is pushed onto this stack before completing the request. If the user wants to return to a previous zoom, clicking the "Previous Zoom" button will cause the most recent zoom state to be popped off the stack and applied to the view.

Panning the view was another useful feature that SHIV initially lacked. The right mouse button was not used in the context of the SHIV canvas, so the right-click and right-release events were bound to appropriately named functions. When a right-click is detected, the canvas instance stores the (x, y) coordinates of the mouse location. Then, each time the mouse is moved, the displacement between the mouse's new and old locations is calculated and the clip is shifted by an equivalent amount. When the right button is released, the canvas stops maintaining a variable for the mouse's location and no longer shifts the clip when the mouse is moved. The ShivCanvas class already owned a reference to a Transformer object, which can translate between canvas coordinates (screen pixels) and plot coordinates (the locations of data points). This translation function was called on the mouse's $\Delta$x and $\Delta$y to find the distance the plot space needed to be shifted, and this information was used to call the LineSet method for setting the clip. Finally, the axes were refreshed and the canvas was redrawn.

Since redrawing thousands of line segments can be very slow, a way to improve the panning feature would be to draw an arrow that indicates the direction and magnitude of the panning motion. Assuming the entire canvas contents (or faster still, a dirty rectangle containing only the region that needs refreshing) could be saved to an bitmapped image object in memory, the arrow could be drawn on top of the image rather than the entire dataset each frame. Blitting one image would be far faster than processing the many line segment objects, so this would be a worthwhile modification to consider for future improvements.

The last improvement to be added was the zooming with the scroll wheel. Many other pieces of graphical software, like Google Maps, use scrolling to zoom. Similarly

to Google Maps, the zoom transformation "centers" the frame around the location of the mouse. In other words, zooming in moves "towards" the location of the mouse, and zooming out moves "away." Below is the code that calculates the new clip boundaries. "clip" is the new clip, while "temp" is the old clip from before the zoom event was triggered. Both clips are tuples with 6 elements; the first four are (left, right, top, bottom). "ex" and "ey" are the x and y coordinates of the location of the mouse cursor at the time of the event. "xform" is the Transformer object attached to the ShivCanvas, which calls inv_x() and inv_y() to convert from pixel space to plot space. "scl" is either 0.8 or 1.25 (note that these are reciprocals).

```
clip[0] = self.xform.inv_x(ex - scl * (ex - temp[0]))
clip[1] = self.xform.inv_x(ex + scl * (temp[1] - ex))
clip[2] = self.xform.inv_y(ey - scl * (ey - temp[2]))
clip[3] = self.xform.inv_y(ey + scl * (temp[3] - ey))
```

The code works by finding the distance from the mouse event to each of the four clip boundaries, and multiplying that difference by scl. The result is added to or subtracted from ex and ey, and then translated to plot coordinates. This means that in the new clip, the point (ex, ey) is in exactly the same place both in plot and pixel space. The scaling factor is a constant proportion, so the left, right, top and bottom were all adjusted equally. The result of this process looks good; it is symmetrical and intuitive.

## 3. Conclusions.

ALEGRA is a very powerful tool for modeling physical interactions. To take full advantage of the software, both developers and end users must pay attention to the way output files are created and used. The RMIN problem is an example of a situation where the entire data set must be examined carefully for validity. At first glance, everything may look correct, but there may still be issues with the data on a small scale. It is impossible for ALEGRA to handle every abnormal situation perfectly, so projects like the RMIN documentation are important for ensuring that the user is aware of pitfalls like the negative $dP/d\rho$.

A long-term plan for ALEGRA is to add automatic fetching of RMIN data for each EOS. The tables produced by the RMIN documentation project provide the basis for implementing this feature. Until then, the tables will serve as a readily available source of information for users' simulations.

Another way in which ALEGRA's supporting material was made more useful was by some interface tweaks to the Specialized HIstory Viewer (SHIV), the Python program used to visualize Equation of State tables. Using SHIV during the RMIN documentation provided the perspective of an end user, prompting the consideration of what changes other users would find helpful. The goal was to add features that didn't crowd the interace or make things more confusing. All SHIV needed was a few minor additions that made navigating large EOS tables easier.

REFERENCES

[1] *SESAME: The Los Alamos National Laboratory Equation of State Database"*, Tech. Rep. LA–UR–92–3407, Los Alamos National Laboratory, 1992.

[2] A. C. Robinson et al., *ALEGRA: An arbitrary Lagrangian-Eulerian multimaterial, multiphysics code*, in Proceedings of the 46th AIAA Aerospace Sciences Meeting, Reno, NV, January 2008. AIAA-2008-1235.

# INVERSION UNDER UNCERTAINTY FOR TRACE GASES USING CONVECTION-DIFFUSION-REACTION

HARRIET LI *, BART G. VAN BLOEMEN WAANDERS †, AND TIM M. WILDEY ‡

**Abstract.**

**1. Introduction.** The characterization of trace-gas sources is important to help control pollutants in the atmosphere. Carbon-dioxide is one of several species that has been linked to the increase of average global temperatures and understanding the overall dynamics of these trace-gases depends on knowing the spatial distribution and magnitudes of carbon-dioxide fluxes. Exact characterizations could for instance support a Green House Gas Information System (GHGIS), which would be responsible for monitoring and managing the overall production of greenhouse gases. The determination of locations and characters of sources is complicated by multiple factors: the multiple spatial distributions, extreme sparsity of the measurements, temporal variations, uncertainty of natural $CO_2$ source and sinks, and the many uncertainties associated with data and model parameters (in particular the velocity field that needs to be calculated from atmospheric/weather models). In this work, we investigate two critical aspects of this inversion. First we explore an efficient inversion under uncertainty scheme that leverages concepts from stochastic optimization. We account for uncertainties assosicated with the velocity fields. Second, we investigate the inversion of trace gas source terms by considering multiple trace gase measurements.

The inversion of source terms motivates a large optimization problem in which the goal is to reconcile the differences between sparse observations and numerical predictions of convection-diffusion-reaction dynamics by manipulating magnitudes of source terms as target inversion parameters. To eventually develop methodolgies that can reconstruct source terms in sufficient detail, many inversion variables need to be considered, potentially at every computational discretization point. Black box approaches in which gradients of the objective function are determined through finite difference methods or local interrogation (non-gradient based) methods quickly become computationally intractable in addition to suffereing from quality issues as a result of for instance selecting an appropriate finite difference step. To address both the computational expense and the accuracy of the gradient, adjoint-based sensitivities need to be implemented. This however poses several implementation challenges associated with parallelism and stabilized finite element discretization. Furthermore, first order optimization methods, such as steepest descent and non-linear conjugate gradient (CG), are not efficient and potentially not sufficently accurate. Second order approaches, such as Newton and Quasi-Newton methods, may be required which may introduce additional implementation challenges.

The determination of accurate trace-gas dynamics in atmospheric flows introduces uncertainties ranging from inaccurate velocity fields at fine spatial scales to the variability of the temporal signals for both anthropogenic and biological source terms. Efficient methods must be considered to manage uncertainties without compromising our ability to invert for large number of source terms while managing stochastic model parameters.

*Massachusetts Institute of Technology, kameeko@mit.edu
†Sandia National Laboratories, bartv@sandia.gov
‡Sandia National Laboratories, tmwilde@sandia.gov

In this work, we first present a large scale optimization approach that leverages adjoint-based sensitivities. Our optimization methods are implemented in a separate package, called Rapid Optimization Library (ROL) within the Trilinos framework, which features a range of algorithms including first and second order methods, line search and trust region globalization, and the ability to accomodate inexact gradient and objective function evaluations. The finite element approach is used to discretize convection-diffusion-reaction physics. For high Peclet numbers, a Stabilized Upwind Petrov Galerkin (SUPG) stabilization method is implemented in both the forward and adjoint operators. The Jacobians in the forward simulator are calculated with automatic differentiation through $C^{++}$ template overloading. Parallelization is achieved through the Epetra package in Trilinos. We leverage concepts from stochastic optimization to manage model uncertainties and strive to derive robust solutions. A "risk measure" is introduced in the objective function and then discretized with collocation methods. Although a range of risk measures can be considered, we limit our approach to several popular ones and to risk measures that can be easily mapped from the financial to the engineering world. In particular we consider an expected value and a coherent value at risk, which are related to risk-neutral and probabilty-of-failure measures, respectively. This approach was first developed in an optimal control problem, and although one might prefer a stochastic inverse solution, the formulation is identical and extends to inverse solution with the computational advantages of the large scale deterministic methods.

**2. Mathematical Formulation.** This section describes the optimization problem that we solve in inferring for model parameters from data, and derives the formula by which the gradient of the objective function is calculated. Several types of risk measures are described, and the reasoning for choosing one is explained.

**2.1. Optimization problem formulation.** The physics of the test cases are described by the convection-diffusion-reaction equations for two species with concentration states $\phi_1$ and $\phi_2$, denoted together by $\bar{\phi}$. Although our target is to invert for source terms $f$, we also consider inversion of the diffusion coefficients $\mu$. Among other unknowns in the model, the velocity field $\vec{v}(\zeta)$ is one of the more important sources of uncertainty. We do not try to infer it but instead assume a stochastic description is available where the velocity term is a function of a random variables $\zeta$ with an appropriate distribution. Our mathematical formulation for the inverse problem is given as follows:

$$\min_d \mathcal{J}(\phi, d) = \sigma\left[\frac{1}{2}\int_T\int_\Omega \left(\bar{\phi}-\phi^\star\right)^2 \delta(x-x^*, t-t^*)\mathrm{d}\Omega\mathrm{d}t - \frac{\beta}{2}\int_\Omega \|d\|^2\right]$$

$$\text{where } \phi \text{ solves } F(\phi, d) = \frac{\partial\phi}{\partial t} - \nabla\cdot(\mu(x)\nabla\phi) + \vec{v}(\zeta)\cdot\nabla\phi - r(\phi) - f(x) = 0,$$

$\sigma$ is the risk measure, $\beta$ is the Tikhonov regularization parameter, which controls the magnitude of the penalty term and depends on the quantity and quality of data. $\sigma$ is the risk measure, and in the case of a risk-neutral measure, it can be replaced with an expected value. The optimization parameter vector $d$ can be either $\mu$ or $f$. To solve this optimization problem, a trust-region method is used, with the use of a truncated conjugate-gradient method to solve the trust-region subproblem; the gradient required by this method is calculated using an adjoint approach, described in [1].

The gradient can be calculated by differentiating the objective function and making use of the chain rule:

$$\frac{D\mathcal{J}}{Dd} = \frac{\partial \mathcal{J}}{\partial \phi}\frac{\partial \phi}{\partial d} + \frac{\partial \mathcal{J}}{\partial d}.$$

Since $\phi$ and $d$ are constrained by $F(\phi, d) = 0$, the direct sensitivity matrix can be expressed as

$$\frac{\partial \phi}{\partial d} = -\frac{\partial F}{\partial \phi}^{-1}\frac{\partial F}{\partial d},$$

which when placed in the gradient equation gives

$$\frac{D\mathcal{J}}{Dd} = -\frac{\partial \mathcal{J}}{\partial \phi}\frac{\partial F}{\partial \phi}^{-1}\frac{\partial F}{\partial d} + \frac{\partial \mathcal{J}}{\partial d}.$$

To avoid solving for the direct sensitivity matrix, which for $n_\phi$ states requires solving a linear system with Jacobian $\frac{\partial F}{\partial \phi}$ for each of the $n_\phi$ columns of $\frac{\partial F}{\partial d}$, we reorder the calculation:

$$\frac{D\mathcal{J}}{Dd} = -\frac{\partial F}{\partial \phi}^{-T}\frac{\partial \mathcal{J}}{\partial \phi}\frac{\partial F}{\partial d} + \frac{\partial \mathcal{J}}{\partial d}.$$

where an adjoint solution arises:

$$\frac{\partial F}{\partial \phi}^{T}\lambda = \frac{\partial \mathcal{J}}{\partial \phi}^{T}.$$

The gradient can then be calculated:

$$\frac{D\mathcal{J}}{Dd} = -\lambda\frac{\partial F}{\partial d} + \frac{\partial \mathcal{J}}{\partial d}.$$

**2.2. Risk Measures.** The motivation for augmenting the objective function with a risk measure is to account for some model based uncertainty in an attempt to provide a robust solution. The risk measure is a concept from stochastic optimization and often applied to the management of financial portfolios. The risk measure allows for a mechanism to achieve a range of objectives given the unknown future of the economy. For uncertain market conditions a bank may use risk measures to decide how much currency to keep in reserve, or a business may use them to decide how much to produce.

Generally, a risk measure is a mapping from a set of random variables to the real numbers. In actual applications the risk measure is applied to a probability distribution of losses. Given a loss distribution, the measure should encapsulate the risk associated with it. The choice of risk measures depends on what is considered risky; perhaps a risky investment is one with great variation in its possible returns, or perhaps it is one with very great loss expected in the worst case scenarios. In the case of inverse problems we consider, the "loss" as the observational mismatch combined with the regularization terms.

Risk measures from the financial world are not all easily mapped to engineering applications but there are a few common ones that can be described in the context of engineering targets, including expected value, standard deviation or variance, and Conditional Value-at-Risk (CVaR), The expected value risk measure for function of a random variable $f(X)$, where $X$ has probability distribution $\rho(X)$, is

$$\sigma_{EV}(f(X)) = \mathbb{E}[f(X)] = \int_{-\infty}^{\infty} f(X)\rho(X)\mathrm{d}X.$$

In finance, the expected value is considered "risk-neutral"; a decision maker who is risk neutral cares only about the expected returns or losses. In engineering, the expected value is used when creating a design that is robust to uncertainties[3] and has a favorable mean response. In our context, the loss distribution would correspond to a distribution of penalized errors, composed from observation mismatch and regularization penalties; to minimize the expected loss would be to find a parameter estimate that is robust to uncertainties in the model form.

If neutral to risk is not desired, one might use the standard deviation and variance risk measures, which are defined

$$\sigma_{SD}(f(X)) = \mathbb{E}[f(X)] + w\sqrt[p]{\int_{-\infty}^{\infty} |f(X) - \mathbb{E}[f(X)]|^p \, \rho(X)\mathrm{d}X},$$

$$\sigma_{V}(f(X)) = \mathbb{E}[f(X)] + w\int_{-\infty}^{\infty} |f(X) - \mathbb{E}[f(X)]|^p \, \rho(X)\mathrm{d}X,$$

where usually $p = 2$. These risk measures are used with the assumption that the higher the variance of a variable, such as a portfolio, the more risky. Semideviation and semivariance risk measures, defined by

$$\sigma_{SemiD}(f(X)) = \mathbb{E}[f(X)] + w\sqrt[p]{\int_{-\infty}^{\infty} (\max\{0, f(X) - \mathbb{E}[f(X)]\})^p \, \rho(X)\mathrm{d}X},$$

$$\sigma_{SemiV}(f(X)) = \mathbb{E}[f(X)] + w\int_{-\infty}^{\infty} (\max\{0, f(X) - \mathbb{E}[f(X)]\})^p \, \rho(X)\mathrm{d}X,$$

can be used if only deviation towards the worse side (in the definition above, the more positive side) of the mean is considered risky. For a portfolio, these risk measures might represent a tradeoff between expected returns and the risk one associates with the uncertainty in these returns; in engineering, they might represent a tradeoff between expected performance and uncertainty in the performance that can actually be achieved. The choice of weighting, however, reflects one's own personal aversion to variability.

If one is instead concerned about worst-case scenarios, then one might use the CVaR risk meansure. For a chosen confidence level $0 \leq \alpha \leq 1$ and continuous distribution $\rho(X)$, the CVaR risk measure is defined by

$$\sigma_{CVaR}(f(X), \alpha) = \frac{1}{1-\alpha}\int_{Q_\alpha}^{\infty} f(X)\rho(X)\mathrm{d}X,$$

where $Q_\alpha = \sup\{x \in \mathbb{R} | P(X \leq x) \leq \alpha\}$ is the $\alpha$-quantile. The CVaR risk measure represents the expected loss in the worst $100(1-\alpha)\%$ of the distribution, and as a mean is less sensitive to sampling error than the quantile risk measure $Q_\alpha$ by itself.[4] The CVaR risk measure is related to that used in reliability-based design formulations, in which one wishes to minimize the probability of exceeding a certain threshold. When there is less of a clear line between an acceptable outcome and catastrophic failure, the CVaR risk measure can be used instead to minimize the expected outcome of the worst-case scenarios. What value of $\alpha$ is chosen to represent the most extreme outcomes is again a matter of one's personal degree of aversion to this risk.

In this study, the expected value risk measure is chosen to obtain a parameter estimate that is robust to uncertainties in the model form and least dependent on any personal preference for or against risk.

**3. Implementation.** To solve the optimization problem, we use a trust-region method; the trust-region subproblem is solved using the truncated conjugate-gradient method. Each of the source coefficient parameters is bounded above and below to prevent the sources from becoming sinks and to keep the concentration states from becoming so large that the forward solve does not converge. The risk measure is evaluated by sampling the three-dimensional stochastic space using a seven-point sparse grid generated from the Gauss-Hermite quadrature rule.

The forward model PDE is solved using a Galerkin finite element method, with backwards Euler for timestepping and a Newton method to solve the nonlinear system at each timestep. Since a convection-dominated problem solved by the standard Galerkin finite element method can produce eroneous oscillatory solutions if the Peclet number is too high, the streamline upwind Petrov Galerkin (SUPG) method is used to stabilize the solution. For weight function $w$, the local residual for the weak form of the convection-diffusion-reaction problem is

$$R = \frac{\partial \phi}{\partial t} + \mu \nabla \phi \cdot \nabla w + (\vec{v} \cdot \nabla \phi) w - r(\phi) w - f w.$$

With SUPG stabilization, the local residual becomes

$$R = \frac{\partial \phi}{\partial t} + \mu \nabla \phi \cdot \nabla w + (\vec{v} \cdot \nabla \phi) w - r(\phi) w - f w + \tau (\vec{v} \cdot \nabla \phi - f)(\vec{v} \cdot \nabla w),$$

where

$$\tau = \left( \frac{C_1 k}{h^2} + \frac{C_2 \|\vec{v}\|}{h} \right)^{-1}$$

and $h$ is the size of the element with $C_1 = 4.0$ and $C_2 = 2.0$. The stabilized problem is no longer adjoint consistent, so taking the adjoint (transpose) of the discretized forward system (discretize-then-optimize) is no longer equivalent to discretizing the continuous adjoint system (optimize-then-discretize)[2]. The discretize-then-optimize approach was selected for ease of implementation and ultimately the optimization requires the discrete form of the adjoint.

**4. Numerical Results.** This section presents numerical experimentation results to demonstrate our algorithmic approach. We first consider the deterministic inverse problem when data are available from many sensors, for progressively more complex inferences, then add uncertainty to the model and reduce the number of sensors.

We start by presenting estimation results for diffusivity and source parameters, individually and simultaneously, for a linear convection-diffusion model with data from numerous sensors are given. In these cases, the data is sufficiently informative to accurately recover the true parameter values. The simultaneous estimation of diffusivity and source parameters is repeated for a nonlinear two-species convection-diffusion-reaction model, with data about either one or both species available. For this more complex physics, having a large amoung of data about just one species is not quite enough to recover all the source terms; this can be remedied by adding data about the second species.

We then consider the case where there is uncertainty in the model, and a simultaneous estimation of diffusivity and source parameters for a nonlinear two-species convection-diffusion-reaction model is again performed, for different levels of uncertainty; more uncertainty in the model resulted in parameter estimates that deviated further from the truth values. Lastly, source parameters are estimated given data from sparse sensors; it is shown that adding data about one species can help estimate the source terms of the other.

In all these cases, the computational domain is $\Omega = [0,1] \times [0,1]$ with zero intial conditions, discretized by $40 \times 40$ elements with linear nodal bases. To avoid an inversion crime, data is generated from a finer mesh $80 \times 80$ and then contaminated with Gaussian white noise.

**4.1. Deterministic Convection-Diffusion.** In this section, results are given for the estimation of various parameters given numerous sensors and a convection-diffusion model

$$\frac{\partial \phi}{\partial t} - \nabla \cdot (\mu \nabla \phi) + \vec{v} \cdot \nabla \phi = f$$

for a single species $\phi$. The species is allowed to evolve over 32 timesteps from $t_0 = 0.0$ to $t_f = 1.0$ with homogeneous Dirichlet boundary conditions imposed along the top and bottom of the domain. Data is obtained from 64 sensors and placed in a square grid throughout the domain, taking measurements every $\Delta t = 0.1875$. Since data were available from a large number of sensors at frequent measurements, it was qualitatively decided that regularization was unecessary for the estimation of a handful of parameters in this case with linear physics.

**4.1.1. Diffusivity Coefficient Inversion.** As an initial phase, data from 64 sensors is used to estimate just a single parameter: a constant diffusivity coefficient $\mu$. The known source $f$ is described by

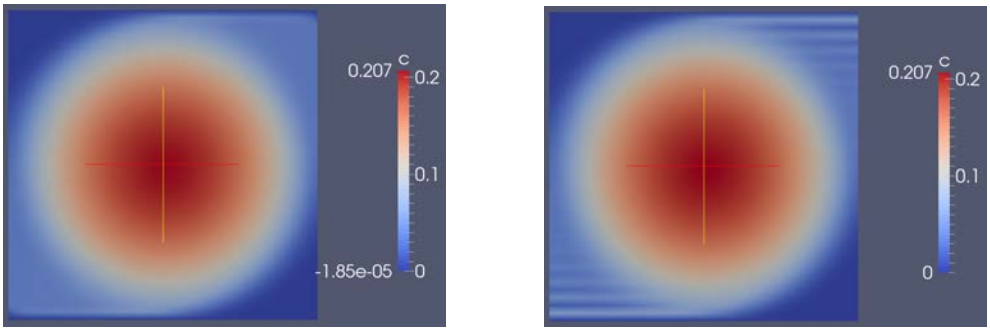
$$f = 10 \exp(-10((x - 0.25)^2 + (y - 0.25)^2)).$$

The known velocity $\vec{v} = (u, v)$ is an irrotational vortex described by

$$u = -1000(y - 0.5),\ v = 1000(x - 0.5).$$

For the given velocity field and element size, and the range of diffusivity coefficients considered, the Peclet number is high enough to warrent the use of a stabilization method to avoid oscillations in the simulated concentration field, as shown in Figure 4.1.

To avoid an inversion crime, Gaussian white noise with standard deviation $\sigma = 10^{-3}$ is added to the data. Although there is a large amount of data available for the estimation of a single parameter, the standard deviation of the noise is only an order




(a) with stabilization (b) without stabilization

Fig. 4.1: Final state, with and without SUPG

of magnitude smaller than the pure measurements, and it would be expected that this relatively high noise level would interfere with what should otherwise be a very accurate estimate of the parameter. This expectation is borne out in the resulting parameter estimate which, as shown in Table 4.1, is much closer to the truth than the initial guess but not as close as might be expected from such a large amount of data.

| Initial guess | Estimated | Truth |
|---|---|---|
| 2.0 | 0.9826992 | 1.0 |

Table 4.1: Estimated diffusivity coefficient - convection-diffusion, no model uncertainty

**4.1.2. Source Inversion.** Next we consider a case where more parameters need to be estimated from the same number of data points; although the resulting optimization problem is simpler than the previous in that it is convex quadratic it is more complex because it is more inversion parameters.

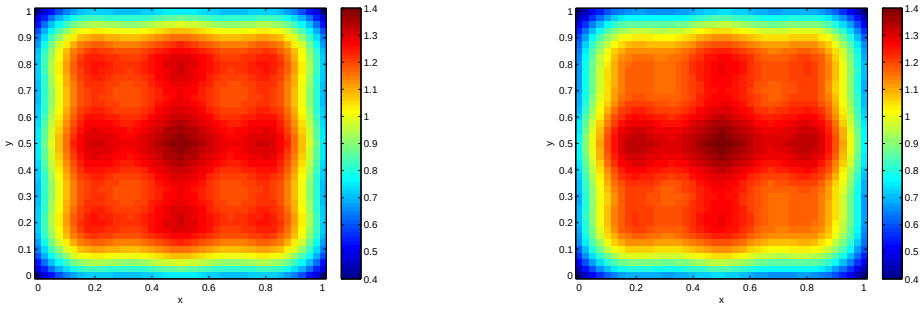
In this case, the diffusivity $k = 1$ is known and the parameters $\vec{C}$ are to be estimated from an algebraic parameterization of the source terms:

$$\begin{aligned} f = {} & C_1 \exp(-20((x-0.15)^2 + (y-0.85)^2)) + C_2 \exp(-20((x-0.5)^2 + (y-0.85)^2)) \\ & + C_3 \exp(-20((x-0.85)^2 + (y-0.85)^2)) + C_4 \exp(-20((x-0.15)^2 + (y-0.5)^2)) \\ & + C_5 \exp(-20((x-0.5)^2 + (y-0.5)^2)) + C_6 \exp(-20((x-0.85)^2 + (y-0.5)^2)) \\ & + C_7 \exp(-20((x-0.15)^2 + (y-0.15)^2)) + C_8 \exp(-20((x-0.5)^2 + (y-0.15)^2)) \\ & + C_9 \exp(-20((x-0.85)^2 + (y-0.15)^2)). \end{aligned}$$

The known velocity $\vec{v} = (u, v)$ is an irrotational vortex described by

$$u = -42(y - 0.5),\ v = 42(x - 0.5).$$

Again, Gaussian white noise with standard deviation $\sigma = 10^{-3}$ is added to the data, and the estimated source terms are shown in Table 4.2 and Figure 4.2. As in the



(a) true $f$ (b) inferred $f$

Fig. 4.2: True and inferred sources - convection-diffusion, no model uncertainty

previous case, the estimated parameters are much closer to the truth than the initial guesses were, but the noise level in the data limited the accuracy of the parameter inversion.

| Parameter | Initial guess | Estimated | Truth |
|---|---|---|---|
| $C_1$ | 2.0 | 0.9164218 | 1.0 |
| $C_2$ | 2.0 | 0.9727920 | 1.0 |
| $C_3$ | 2.0 | 0.9596691 | 1.0 |
| $C_4$ | 2.0 | 1.037823 | 1.0 |
| $C_5$ | 2.0 | 1.014904 | 1.0 |
| $C_6$ | 2.0 | 1.037821 | 1.0 |
| $C_7$ | 2.0 | 0.9596682 | 1.0 |
| $C_8$ | 2.0 | 0.9727950 | 1.0 |
| $C_9$ | 2.0 | 0.9164226 | 1.0 |

Table 4.2: Estimated source coefficients - convection-diffusion, no model uncertainty

**4.1.3. Simultaneous Source and Diffusivity Inversion.** Here we consider an inference problem that combines the difficulties of the nonlinear optimality conditions of the first case with the larger parameter space of the second. Neither the diffusivity nor the source strengths are known. The diffusivity field is modeled as piecewise constant, with the first four parameters representing the diffusivty in four quadrants of the domain. The remaining five parameters describe the source term

$$\begin{aligned} f = {} & C_5 \exp(-20((x-0.5)^2 + (y-0.75)^2)) + C_6 \exp(-20((x-0.75)^2 + (y-0.75)^2)) \\ & + C_7 \exp(-20((x-0.5)^2 + (y-0.5)^2)) + C_8 \exp(-20((x-0.75)^2 + (y-0.5)^2)) \\ & + C_9 \exp(-20((x-0.25)^2 + (y-0.25)^2)). \end{aligned}$$

The velocity field is the same as in the previous case. The standard deviation of the Gaussian white noise added to the data is reduced to $\sigma = 10^{-4}$, and the estimated parameters are shown in Table 4.3. Compared to the previous case, the accuracy of the inferred parameters is improved, reflecting the reduced noise level in the data.

| Parameter | Initial guess | Estimated | Truth |
|---|---|---|---|
| $C_1$ | 2.0 | 1.000415 | 1.0 |
| $C_2$ | 2.0 | 0.9965526 | 1.0 |
| $C_3$ | 2.0 | 1.002004 | 1.0 |
| $C_4$ | 2.0 | 0.9990992 | 1.0 |
| $C_5$ | 2.0 | 0.9859356 | 1.0 |
| $C_6$ | 2.0 | 1.008036 | 1.0 |
| $C_7$ | 2.0 | 1.007769 | 1.0 |
| $C_8$ | 2.0 | 0.9917184 | 1.0 |
| $C_9$ | 2.0 | 0.9968621 | 1.0 |

Table 4.3: Estimated parameters - convection-diffusion, no model uncertainty

**4.2. Deterministic Convection-Diffusion-Reaction.** In this case, diffusivity and source coefficients are again simultaneously estimated, but with a nonlinear two-species convection-diffusion-reaction model. The state equations are

$$\frac{\partial \phi_1}{\partial t} - \nabla \cdot (\mu \nabla \phi_1) + \vec{v} \cdot \nabla \phi_1 = \alpha \phi_2 + f_1$$

$$\frac{\partial \phi_2}{\partial t} - \nabla \cdot (\mu \nabla \phi_2) + \vec{v} \cdot \nabla \phi_2 = \alpha \phi_1 + f_2$$

where $\phi_1$ and $\phi_2$ are the concentration states of the two species and $\alpha = 1.0$ is the reaction coefficient. The model is run from $t_0 = 0.0$ to $t_f = 1.0$ in 32 timesteps, and homogeneous Dirichlet boundary conditions are imposed along the top and bottom of the domain. There are nine parameters to estimate, the first being the constant diffusivity and the rest describing the source terms

$$\begin{aligned} f_1 &= C_2 \exp(-20((x-0.2)^2 + (y-0.8)^2)) + C_3 \exp(-20((x-0.4)^2 + (y-0.8)^2)) \\ &\quad + C_4 \exp(-20((x-0.6)^2 + (y-0.8)^2)) + C_5 \exp(-20((x-0.8)^2 + (y-0.8)^2)) \\ f_2 &= + C_6 \exp(-20((x-0.2)^2 + (y-0.2)^2)) + C_7 \exp(-20((x-0.4)^2 + (y-0.2)^2)) \\ &\quad + C_8 \exp(-20((x-0.6)^2 + (y-0.2)^2)) + C_9 \exp(-20((x-0.8)^2 + (y-0.2)^2)). \end{aligned}$$

The known velocity $\vec{v} = (u, v)$ is an irrotational vortex described by

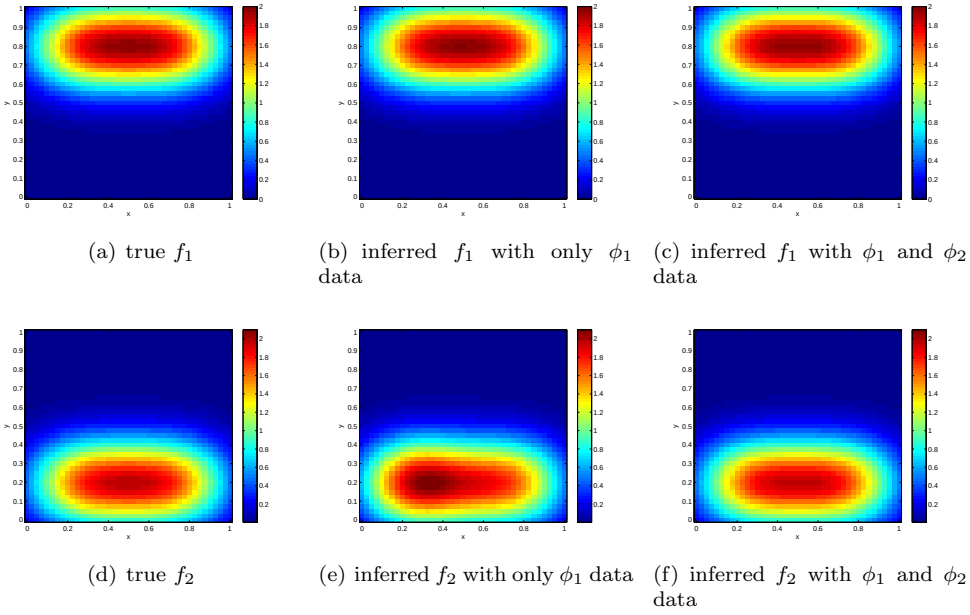
$$u = -42(y - 0.5),\ v = 42(x - 0.5),$$

and since the diffusivity varies as the parameters space is explored, SUPG stabilization is used to avoid possible oscillations. Data contaminated with Gaussian white noise with standard deviation $\sigma = 10^{-3}$ is taken from 64 sensors every $\Delta t = 0.1875$. The estimated parameters are shown in Table 4.4 and the estimated source terms are shown in Figure 4.3.

The data is sufficient to obtain close estimates of the diffusivity and source parameters for the first species, and although data of only the first species is available, the interaction of the two species through the reaction term, along with the large number of sensors, allows for a close estimate of two of the four source parameters for the second species as well. Of course, if each sensor could provide data for both species, the parameter estimate is much improved.

| Parameter | Initial guess | Data from $\phi_1$ only | Data from $\phi_1$ and $\phi_2$ | Truth |
|---|---|---|---|---|
| $C_1$ | 2.0 | 1.001678 | 0.9952705 | 1.0 |
| $C_2$ | 2.0 | 1.014302 | 1.035935 | 1.0 |
| $C_3$ | 2.0 | 1.002739 | 0.9887638 | 1.0 |
| $C_4$ | 2.0 | 0.9935702 | 0.9863631 | 1.0 |
| $C_5$ | 2.0 | 1.014242 | 1.038581 | 1.0 |
| $C_6$ | 2.0 | 1.359563 | 1.038581 | 1.0 |
| $C_7$ | 2.0 | 1.022204 | 0.9863532 | 1.0 |
| $C_8$ | 2.0 | 0.8746213 | 0.9887834 | 1.0 |
| $C_9$ | 2.0 | 1.097095 | 1.035925 | 1.0 |

Table 4.4: Estimated parameters - convection-diffusion-reaction, no model uncertainty



(a) true $f_1$ (b) inferred $f_1$ with only $\phi_1$ data (c) inferred $f_1$ with $\phi_1$ and $\phi_2$ data

(d) true $f_2$ (e) inferred $f_2$ with only $\phi_1$ data (f) inferred $f_2$ with $\phi_1$ and $\phi_2$ data

Fig. 4.3: True and inferred sources - convection-diffusion-reaction, no model uncertainty

**4.3. Convection-Diffusion-Reaction with model uncertainty.** In this section, uncertainty is added to the convection-diffusion-reaction model and the traditional deterministic objective function augmented with the expected value risk measure. First, a simultaneous estimation of diffusivity and source paramaters is performed with data from numerous sensors available and in the presence of different degrees of uncertainty. Then a case is examined in which source parameters are estimated given sparse sensors.

**4.3.1. Numerous Sensors.** The optimization problem can be formulated by:

$$\min_d \mathcal{J}(\phi, d) = \mathbb{E}\left[\frac{1}{2}\int_T \int_\Omega \left(\bar{\phi} - \phi^\star\right)^2 \delta(x - x^*, t - t^*) \mathrm{d}\Omega \mathrm{d}t - \frac{\beta}{2}\int_\Omega \|d\|^2\right]$$

where $\phi$ solves:

$$\frac{\partial \phi_1}{\partial t} - \nabla \cdot (\mu \nabla \phi_1) + \vec{v} \cdot \nabla \phi_1 = \alpha \phi_1^2 + f_1$$

$$\frac{\partial \phi_2}{\partial t} - \nabla \cdot (\mu \nabla \phi_2) + \vec{v} \cdot \nabla \phi_2 = \alpha \phi_1^2 + f_2$$

with $\alpha = 1.0$; the reaction term is based on the reaction rate $r = k[NO_2]^2$ of the reaction

$$CO_{(g)} + NO_{2(g)} \rightarrow CO_{2(g)} + NO_{(g)},$$

with $\phi_1 = [NO_2]$ and $\phi_2 = [CO]$. The same timesteps and boundary conditions are used as in the previous case. There are three parameters to estimate, the first being the constant diffusivity and the other two describing the source terms for the first and second species, respectively:

$$f_1 = C_2 \exp(-20((x - 0.3)^2 + (y - 0.5)^2))$$
$$f_2 = C_3 \exp(-20((x - 0.7)^2 + (y - 0.5)^2)).$$

The velocity field $\vec{v} = (u, v)$ is again an irrotational vortex, but there is uncertainty in its magnitude described by

$$u = -20\zeta(y - 0.5),\ v = 20\zeta(x - 0.5),$$

where $\zeta \sim \mathcal{N}(1, \sigma_\zeta^2)$ and the mean was used to generate the data. Data with Gaussian white noise with standard deviation $\sigma = 10^{-4}$ is available from 36 sensors taking measurements of both species every $\Delta t = 0.1875$, so no regularization is used. As in the previous case, since the diffusivity varies as the parameters space is explored, stabilization is used to avoid possible oscillations. The stochastic space was sampled using a sparse grid built form the Gauss-Hermite quadrature rules.

The parameter estimates obtained in the presence of an uncertain velocity field are shown in Table 4.5, for $\sigma_\zeta = 0.005$ and $\sigma_\zeta = 0.5$. The data is sufficient to obtain a good estimate the parameter values when the velocity field is known, but as the uncertainty in the velocity field increases, the estimates increasingly deviate from the truth, as would be expected.

**4.3.2. Sparse Sensors.** The physics of interest are described by the convection-diffusion-reaction equations for two species since the diffusivity varies as the parameters space is explored.

$$\frac{\partial \phi_1}{\partial t} - \nabla \cdot (k \nabla \phi_1) + \vec{v} \cdot \nabla \phi_1 = \frac{\alpha}{2} \phi_1^2 \phi_2 + f_1$$

$$\frac{\partial \phi_2}{\partial t} - \nabla \cdot (k \nabla \phi_2) + \vec{v} \cdot \nabla \phi_2 = \alpha \phi_1^2 \phi_2 + f_2$$

| Parameter | Initial guess | $\sigma_\zeta = 0.5$ | $\sigma_\zeta = 0.005$ | Truth |
|---|---|---|---|---|
| $C_1$ | 2.0 | 1.1582 | 1.02741 | 1.0 |
| $C_2$ | 2.0 | 1.4644 | 1.33455 | 1.3 |
| $C_3$ | 2.0 | 1.8022 | 1.63228 | 1.6 |

Table 4.5: Estimated parameters - convection-diffusion-reaction with model uncertainty

where $\phi_1$ and $\phi_2$ are the concentration states of the two species, $k = 0.01$ is the diffusivity coefficient, and $\alpha = 2.0$ is the reaction coefficient. The reaction term is based on the reaction rate $r = \alpha[NO]^2[Cl_2]$ of the reaction

$$Cl_{2(g)} + 2NO_{(g)} \rightarrow 2NOCl_{(g)},$$

with $\phi_1 = [NO]$ and $\phi_2 = [Cl_2]$. Nitric oxide is a byproduct of combustion in the presence of nitrogen, which is the main component of air, and chlorine gas has commercial and industrial applications as a disinfectant and for water treatment.

The species concentrations are allowed to evolve over 32 timesteps from $t_0 = 0.0$ to $t_f = 1.0$ with natural boundary conditions. In this simple test case, we assume that we know the locations of the sources producing species 1, but wish to invert for their magnitudes; the source $f_1$ is described by

$$\begin{aligned} f_1 = C_1 \exp(-20((x-0.1)^2 + (y-0.6)^2)) \\ + C_2 \exp(-20((x-0.25)^2 + (y-0.7)^2)) \\ + C_3 \exp(-20((x-0.5)^2 + (y-0.8)^2)) \\ + C_4 \exp(-20((x-0.7)^2 + (y-0.85)^2)) \\ + C_5 \exp(-20((x-0.8)^2 + (y-0.9)^2)), \end{aligned}$$

where $\vec{C} = (C_1, C_2, C_3, C_4, C_5)$ are the parameters we try to estimate from the data. The source $f_2$ also has a known location, but its magnitude is treated as an uncertainty in the model form rather than a parameter:

$$f_2 = \zeta_3 \exp(-10((x-0.6)^2 + (y-0.4)^2)),$$

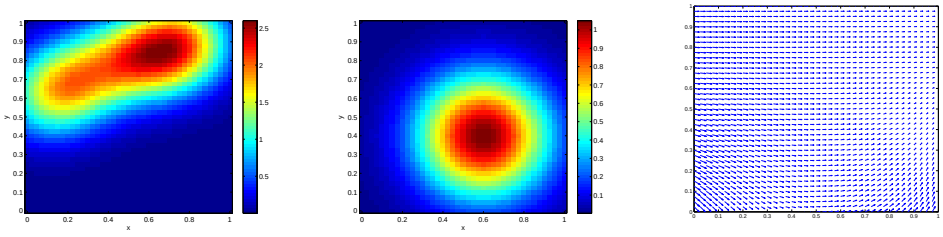
where $\zeta_3 \sim \mathcal{N}(1, (0.01)^2)$ is a random variable. The velocity field $\vec{v} = (u, v)$,

$$u = 1.0 + \zeta_1 \frac{y^2 - (x+0.5)^2}{y^2 + (x+0.5)^2} - \zeta_2 \frac{y^2 - (x-1.5)^2}{y^2 + (x-1.5)^2}$$

$$v = -\zeta_1 \frac{y^2 - (x+0.5)^2}{y^2 + (x+0.5)^2} + \zeta_2 \frac{y^2 - (x-1.5)^2}{y^2 + (x-1.5)^2},$$

is also a source of uncertainty in the model form, with $\zeta_1, \zeta_2 \sim \mathcal{N}(1, (0.1)^2)$. The "true" values of these random variables that are used to produce synthetic data are $\zeta^\star = (1.05, 1.05, 1.05)$; the true source coefficients are $C^\star = (1.0, 1.2, 1.4, 1.2, 1.0)$. The data is perturbed by normally distributed white noise with standard deviation $\sigma = 10^{-4}$. The true sources and velocity field are shown in Figure 4.4.

The data comes from two sensors placed at $(0.3, 0.2)$ and $(0.3, 0.7)$, one near the sources and one at a location that the first species was expected to be convected



(a) true $f_1$ (b) true $f_2$ (c) true velocity

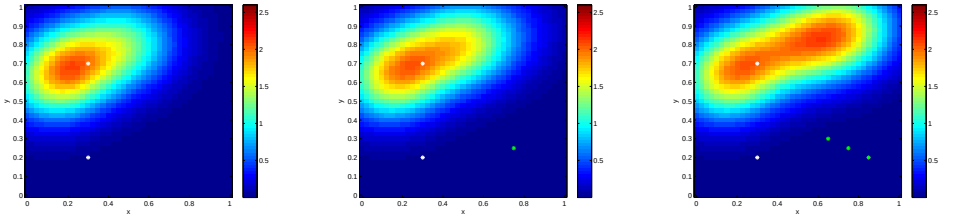
Fig. 4.4: True sources and velocity field

| Parameter | Initial guess | 2 $\phi_1$ sensors 0 $\phi_2$ sensors | 2 $\phi_1$ sensors 1 $\phi_2$ sensor | 2 $\phi_1$ sensors 3 $\phi_2$ sensors | Truth |
|---|---|---|---|---|---|
| $C_1$ | 3.0 | 0.966013 | 0.990649 | 0.992926 | 1.0 |
| $C_2$ | 3.0 | 1.31042 | 1.22505 | 1.22334 | 1.2 |
| $C_3$ | 3.0 | 1.01494 | 1.27215 | 1.23333 | 1.4 |
| $C_4$ | 3.0 | 0.065891 | 0.276508 | 0.93828 | 1.2 |
| $C_5$ | 3.0 | 0.00815815 | 0.111768 | 0.700139 | 1.0 |

Table 4.6: Estimated source coefficients

through, based on the mean velocity field; each sensor took measurements of $\phi_1$ every $\Delta t = 0.1875$. Given the sparse sensors, Tikhonov regularization with $\beta = 10^{-4}$ is used. The resulting estimated source coefficients are compared with that obtained if additional data is available from a sensor at $(0.75, 0.25)$, taking measurements of $\phi_2$ at the same timesteps; this additional sensor is located near the center of $f_2$ and thus where $\phi_2$ was expected to be high.

The estimated source coefficient parameters are summarized in Table 4.6. Using only the measurements of $\phi_1$ from two sensors gives an estimate of $f_1$ shown in Figure 4.5(a). The two sensors are only able to provide enough information for a fair estimate of the source components they are closest to; the ones further away are mostly informed by the regularization term. Using measurements of $\phi_2$ from an additional sensor provides information on the state and thus source of the first species, improving the estimates of the source coefficients. The improved source estimate is shown in Figure 4.5(b). Including data from two more sensors of $\phi_2$ at $(0.65, 0.3)$ and $(0.85, 0.2)$, also located near the center of $f_2$ and thus where $\phi_2$ is expected to be higher and the reaction term larger, further improves the source estimate, shown in Figure 4.5(c).



(a) two $\phi_1$ sensors (white) (b) two $\phi_1$ sensors (white) and one $\phi_2$ sensor (green) (c) two $\phi_1$ sensors (white) and three $\phi_2$ sensors (green)

Fig. 4.5: Inferred source - convection-diffusion-reaction, with model uncertainty

**5. Conclusions.** We present the efficient solution of a parameter estimation problem that is robust to model uncertainties, taking advantage of stochastic optimization algorithms. Both the inversion of diffusivity and source coefficients are investigated for numerous and sparse sensors, utilizing Thikonov regularization for the latter. Convection-diffusion-reaction physics with SUPG stabilization is used to evaluate the use of two reacting species in the presence of uncertainty in the velocity field. It is shown that additional information from another trace-gas species improves the reconstruction of the other trace-gas source term coefficient. Furthermore, robust inversion solutions are obtained in the face of incertainty, expoiting an expected value in the objective function to reflect a risk neutral measure.

REFERENCES

[1] S. S. Collis, R. A. Bartlett, T. M. Smith, M. Heinkenschloss, L. C. Wilcox, J. C. Hill, O. Ghattas, M. O. Berggren, V. Akcelik, C. C. Ober, B. G. van Bloemen Waanders, and E. R. Keiter, *Sensitivity technologies for large-scale simulation*, Tech. Rep. SAND2004-6574, 2004. http://prod.sandia.gov/techlib/access-control.cgi/2004/046574.pdf.

[2] S. S. Collis and M. Heinkenschloss, *Analysis of the streamline upwind/Petrov Galerkin method applied to the solution of optimal control problems*, Tech. Rep. TR02–01, Department of Computational and Applied Mathematics, Rice University, Houston, TX 77005–1892, 2002. http://www.caam.rice.edu/∼heinken.

[3] A. A. Guinta, M. S. Eldred, L. P. Swiler, T. G. Trucano, and S. F. Wojtkiewicz, *Perspectives on optimization under uncertainty: Algorithms and applications*, in Proceedings of 10th AIAA/ISSMO Multidisciplinary Analysi and Optimization Conference, Albany, NY, 2004.

[4] M. R. Hardy, *An introduction to risk measures for actuarial applications*, tech. rep., Education and Examination Committee of the Society of Actuaries, 2006. http://www.utstat.utoronto.ca/sam/coorses/act466/rmn.pdf.

# ANGLE DEPENDENCE OF OBLIQUE IMPACT PROBLEMS IN ALEGRA

JARED M. STATEN * AND ALLEN C. ROBINSON †

**Abstract.** Alegra is a powerful multiphysics Arbitrary Langrangian Eulerian simulation code. To ensure its reliability, inconsistencies and errors must continually be investigated and solved. Evidence has been found of potential angle dependence in oblique impact problems run with Alegra. Extensive testing and obeservation was done in this project to identify the cause of the problem. The project results indicate that the remap method is directionally independent (as designed), but the Johnson Cook fracture model is a significant source of grid-related error. Using the information gathered by this project, the next step is to determine why the Johnson Cook fracture model has a grid-dependence issue and fix the issue altogether.

**1. Introduction.** High velocity impact problems are difficult to test with physical experiments. They can also be very expensive, especially for large scale problems. For this reason, a simulation tool such as Alegra is invaluable for studying the mechanics of high velocity impact. Unfortunately, due to the nature of computer simulations, what is gained in efficiency is compromised in the accuracy of the testing. Some problems, when changed slightly, result in catastrophic errors when the problem should run smoothly. The more common case, though, is variation in the material ejected from the impact point.

Due to the complexity of Alegra, the inconsistency could be caused by a number of different factors. Suspected to be an error in either remap or the material model, the issue required extensive testing before any conclusions could be made. The purpose of this project was to collect data concerning the inconsistencies of oblique impact problems to support future investigations of the problem.

Section 2 describes the problem setup and naming system used for all problems in this project. Section 3 details the first set of tests, used to investigate whether remap was the source of the issue. Section 4 investigates the possibility that the inconsistency was due to the material model in use.

**2. Computational Setup.** The problems were run in a two-dimensional domain nine inches in the Y direction by ten inches in the X direction. The steel target was seven inches long by two inches thick, and was rotated about the center of the domain by angle $\phi$. The copper projectile was four inches long by ten millimeters thick, meaning it had a length-to-thickness ratio of about 10. It was rotated, also about the center of the domain, by angle $\alpha$. The projectile was given a velocity of 1200 m/s.

All problems were named using the same system: ufem_$\phi$_$\alpha$ for problems run with regular Alegra, and xfem_$\phi$_$\alpha$ for problems run using Alegra-XFEM. $\phi$ represents the angle of offset of the target from vertical and $\alpha$ represents the angle of offset of the projectile from horizontal. Problems used to experiment with refinement were named according to the level of refinement used. For example, ufem_$\phi$_$\alpha$_hi is a high-resolution problem, and ufem_$\phi$_$\alpha$_lo is a low-resolution problem. High-resolution problems had cell size 0.5 mm, mid-resolution problems had cell size 1 mm, and low-resolution problems had cell size 2 mm.

Each problem was run to a termination time of 10.0e-5 seconds, taking roughly 1200 timesteps for a mid-resolution simulation.

*La Cueva High School, jmstate@sandia.gov
†Sandia National Laboratories, acrobin@sandia.gov

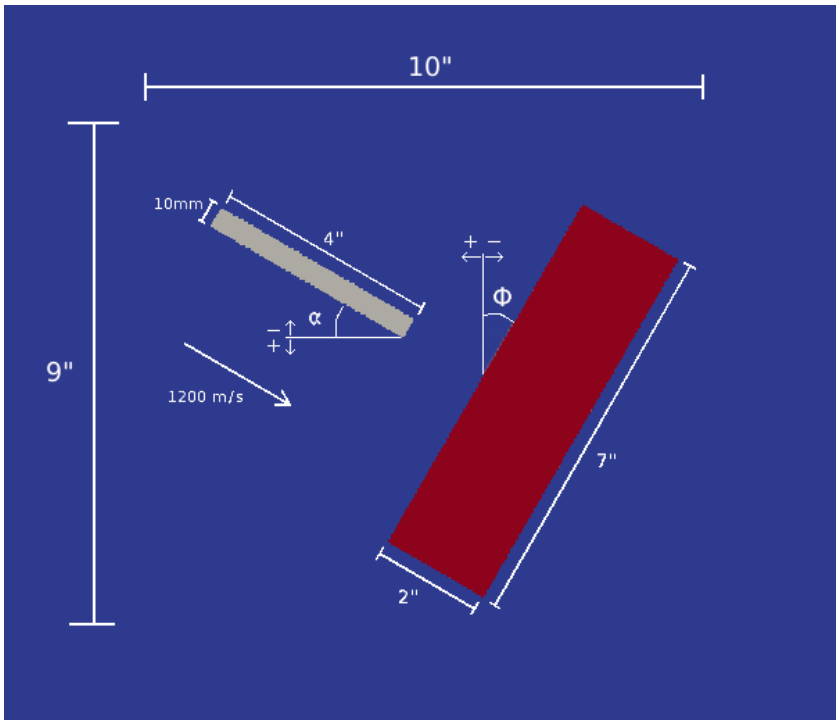



Fig. 2.1: Example problem setup

As indicated, a positive $\phi$ represents a counterclockwise rotation of the steel target, and a negative $\phi$ represents a clockwise rotation. Similarly, a positive $\alpha$ indicates a counterclockwise rotation of the copper projectile and a negative $\alpha$ indicates a clockwise rotation. No signs are used in the names of the problems, however. In every test, the target was rotated clockwise and the projectile was oriented 0, 15, 30, or 45 degrees from it. For example, the angles in the example problem above are negative, ($\phi$=-30,$\alpha$=-30) but the name of the problem would be ufem_30_30 (or xfem_30_30, depending on how the problem is run), not ufem_-30_-30.

**2.1. Methods of Execution.** The majority of problems in this project were run with regular Alegra, which uses a two-step swept-face remap to process and move material. The two-step algorithm alternately sweeps material through X and Y faces of the grid elements, so it had to be eliminated first as a source of angular dependence.

A handful of problems were run with XFEM. XFEM, unlike regular Alegra, uses a single-step intersection remap algorithm, which should be unaffected by the orientation of the problem relative to the grid.

**3. Advection Tests.** Seven different angle sets were used for advection testing, and four of them (0_0, 30_30, 0_30, and 30_0) were used to experiment with mesh refinement. In addition to including a mesh refinement factor, another set of variables was added to track total normal momentum (MOMNORM) and total tangential momentum (MOMTAN), using the frame of the projectile. This addition was necessary because momentum is a vector quantity, and comparison must be made using a

consistent frame of reference.

The normal impact problems were run and observed first. No difference was seen from visualization of the solution, even when the high-resolution results were compared, but small variations were observed in the global history variables. At all three resolutions, problems with higher offset angles lost total energy fastest, as shown in Figure 3.1. The difference is negligible, though, roughly 0.12 percent of the initial total energy of the system (ETOT).

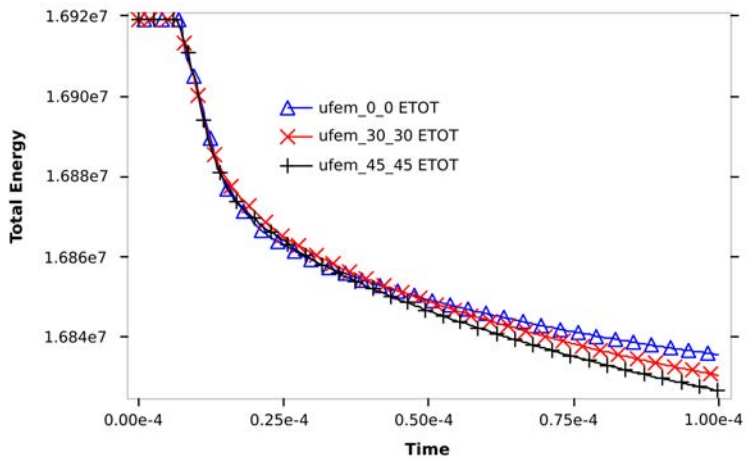



Fig. 3.1: As displayed by the Energy-Time graph, normal impact problems with more offset from the normal orientation lost more total energy.

The 30 degree impact problems proved to be the interesting cases of the advection testing and the rest of the project. 30 degree impact problems at each of the three resolutions are shown in Figure 3.2. The low and medium resolution simulations showed the same types of variations (between 0_30 and 30_0) in ParaView, with the ejected material in the 0_30 problem tending to curve toward the target and the material in the 30_0 problem pointing further from the target. As resolution increased, the two problems resembled one another more closely, indicating that convergence solved the issue. Similarly, for the low and medium resolution simulations, as observed in shivr, 0_30 lost ETOT faster than 30_0. (Figure 3.3) The high resulution simulation however, indicated that the two problems had much closer values for ETOT. Differences between 0_30 and 30_0 reduced significantly in the high resolution simulations, indicating that the problem is converging to a grid-independent solution.

The high-resolution ETOT plot for the 30 degree problem, however, showed interesting results when material ejected by the impact left the domain. This is represented by sudden drops in ETOT. (Figure 3.4) Because ETOT is calculated based on the amount of material in the problem, lost material means lost energy. Thus, drops in ETOT line up with increases in MATLOSS. (Figure 3.4)

The 45 degree oblique impact problems showed the same differences in Alegra as the 30 degree oblique impact problems. (Figure 3.5) The ejected material in the 45_0

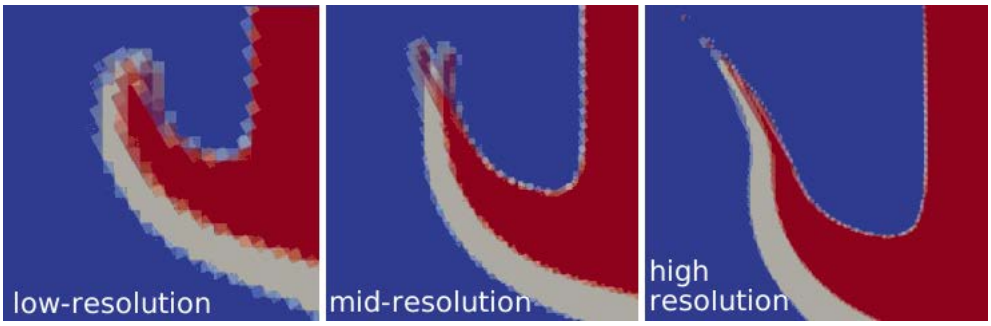



Fig. 3.2: Low resolution testing showed great differences between the 0_30 and 30_0 problems. As resolution increased, (moving right) the difference between the problems was fixed.

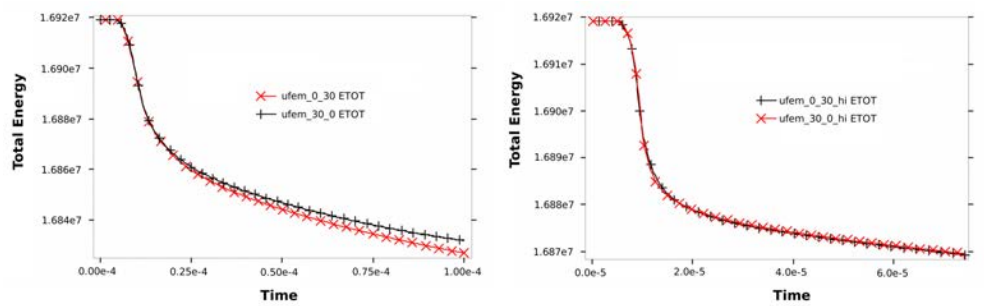



Fig. 3.3: (left) The 0_30 simulation loses ETOT faster than the 30_0 in the mid-resolution simulation. (right) Both 0_30 and 30_0 lose ETOT at the same rate when the problem is refined.

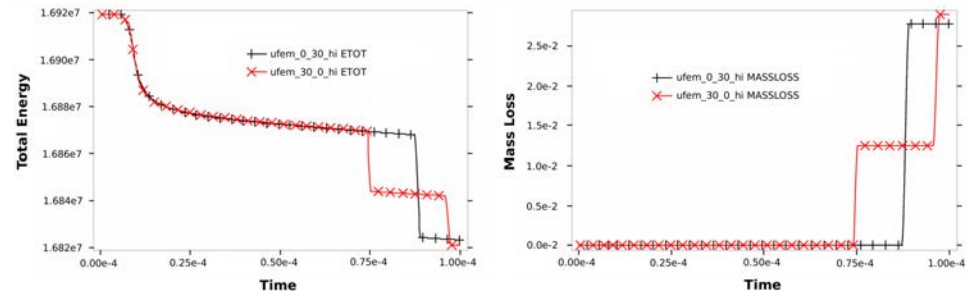



Fig. 3.4: Drops in ETOT are associated with jumps in MATLOSS. Differences between 0_30 and 30_0 are due to differences in boundary placement relative to the problem.

problem curved toward the target and the same material in the 0_45 problem tended away from the target.

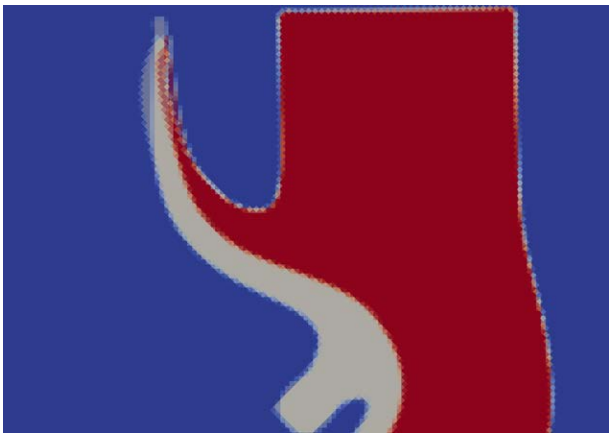


Fig. 3.5: 0_45 and 45_0 simulations are overlaid to show the difference observed in the ejection region. Solid colors represent area filled by both simulations, while translucent areas are only filled by one simulation.

Interestingly, energy and momentum values (ETOT and MOMNORMAL) for the 45 degree oblique impact problems were very similar. (Figure 3.6) ETOT values were visually identical and MOMNORMAL values were numerically identical.

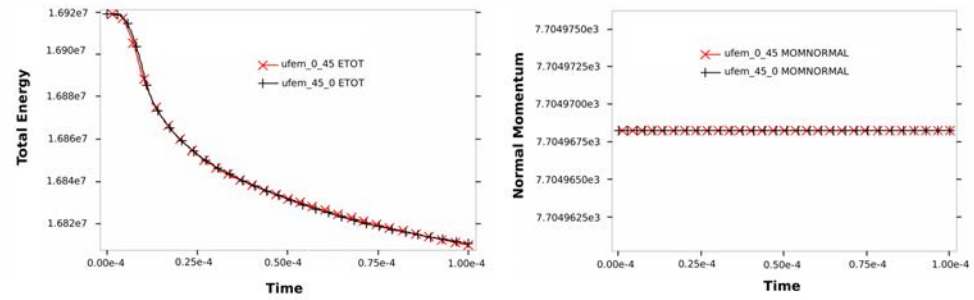



Fig. 3.6: ETOT and MOMNORM values for the 0_45 and 45_0 simulations are practically identical

The advection tests served as a baseline in this project. They were used to investigate whether remap was the source of the problem. As was expected, convergence solved the error, indicating that something else was the source of the issue.

**4. Material Model Tests.** Because the advection tests of the previous study showed that the issues were solved with refinement, it was evident that advection was not affected by obliquity with simple material models. The next step was to add an algorithm suspected to have mesh dependence: the Johnson-Cook fracture model. Two new variables were used to track the behavior of the fracture model: Damage and Failure Fraction. The results of the initial testing prompted a number of additional tests to clarify where and under what conditions the problem was occurring.

**4.1. Regular Alegra tests.** First, a set of seven problems was run using regular Alegra. This set included three normal impact problems, two 30 degree oblique impact problems, and two 45 degree oblique impact problems.

All three normal impact problems ran as expected and were observed to be virtually identical. Notably, the 45 degree oblique impact problems were nearly identical as well.

The two 30 degree oblique impact problems shown in Figure 4.1, however, (0_30, 30_0) differed much more then the others. The problems were run at higher and lower resolutions to see if mesh refinement had any effect. The emitted-particle problem persisted, indicating an issue with the model; refinement did not solve the issue as it had before.

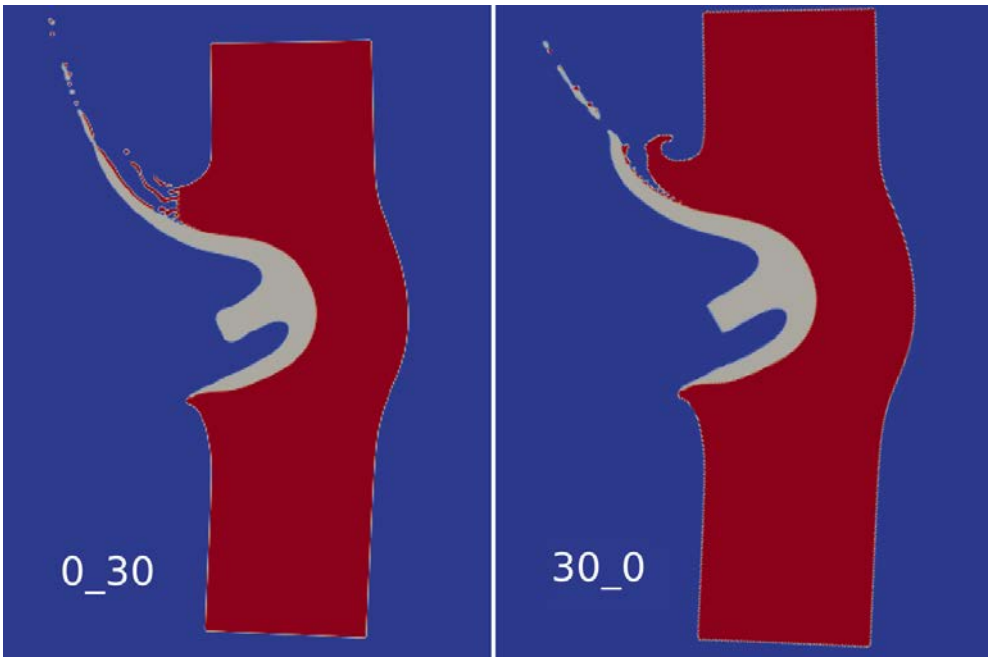



Fig. 4.1: The 0_30 and 30_0 simulations are quite different, even at high resolution. These simulations use the Johnson-Cook fracture model.

While the ejected material of the 30_0 problem shows the expected results, the 0_30 exhibits very peculiar behavior. (Figure 4.1) The ejected material originating from the target stops at a seemingly arbitrary point, and only a small portion of the material continues to move. Meanwhile, the ejected material originating from the projectile behaves as expected.

Figure 4.2 shows the set of new tests that was introduced to investigate the 30 degree angle of impact at varying angles of obliquity: 0_30, 15_15, 20_10, 30_0, 40_10, 60_30, 90_60, 120_90

After running the problems and viewing the results of the new tests, most of the new tests were identical to the 0_30 problem, except for te 120_90 case, which was almost identical to the 30_0 case. The two problems shared a 30 degree displacement

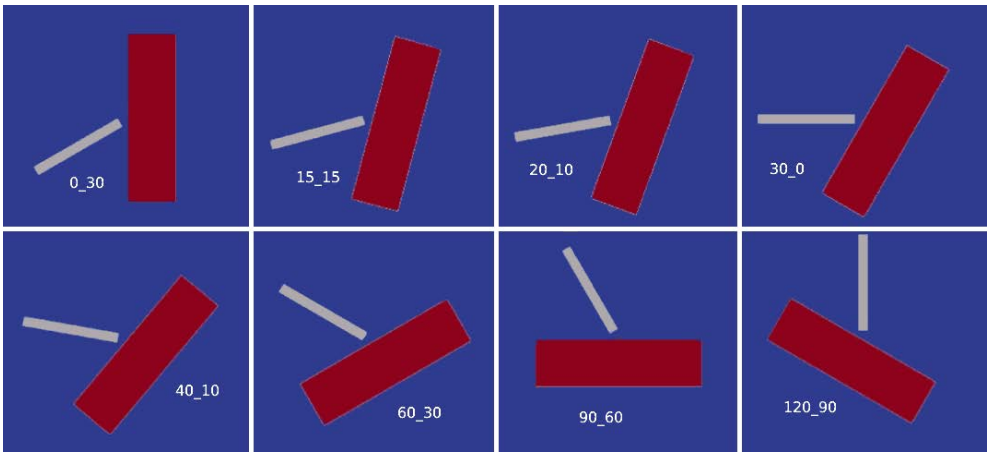



Fig. 4.2: Full set of 30 degree impact problems

from the grid. The addition of 30 degree impact test problems demonstrated X/Y invariance in that each of the test problems was identical to the problem oriented 90 degrees from it. (0_30 was identical to 90_60, 30_0 was identical to 120_90, etc.)

**4.1.1. Damage and Failure Fraction.** To conduct further comparison on the 0_30 and 30_0 problems, the damage and Failure Fraction variables were viewed in ParaView. Based on the results, the 30_0 problem seemed to have taken slightly more damage around the edge of the block. More of the damage in the 0_30 problem was concentrated on the material ejected from the impact point. (Figure 4.3)

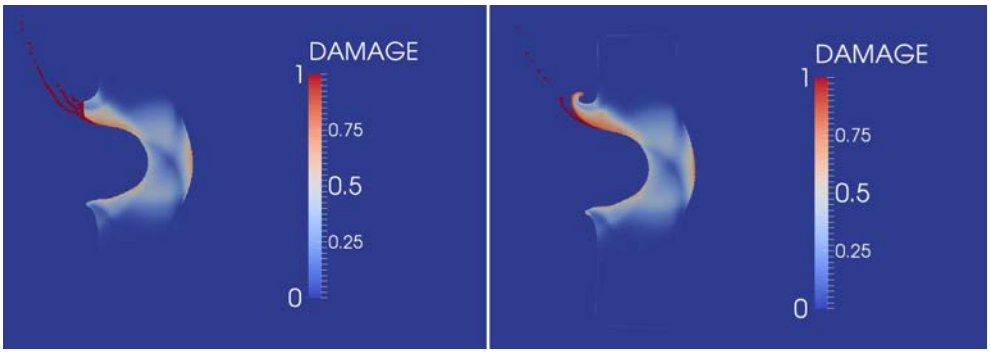



Fig. 4.3: Damage of 30_0 simulation (left) and 0_30 simulation (right)

Failure Fraction was found more consistently in the ejected material of the 0_30 problem and was less present in the 30_0 problem. (Figure 4.4)

The Damage and Failure Fraction visualizations show the same peculiar behavior as the earlier problem. (Figure 4.5) At the same seemingly-arbitrary point where the material stopped in the initial visualization, both Damage and Failure Fraction are

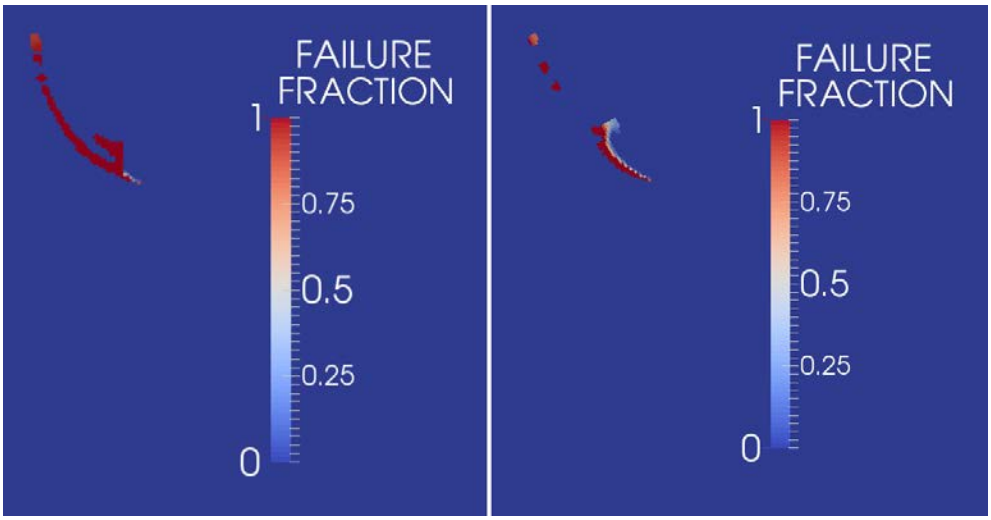



Fig. 4.4: Failure Fraction of 0_30 simulation (left) and 30_0 simulation (right)

suddenly equal to one. In addition, the behavior only occurs in the 0_30 simulation; values for Damage and Failure Fraction are normal in the 30_0 visualizations.

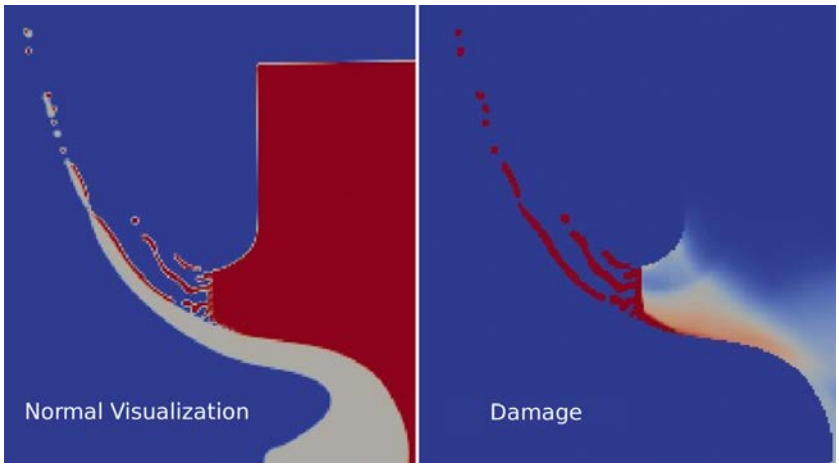



Fig. 4.5: Both visualizations show unusual behavior at the same arbitrary point.

Note: Damage and Failure Fraction variables represent damage sustained by the material and whether or not the material has failed at a given point, respectively. At points where Damage reaches its maximum, the material is damaged to a point that it fails, and Failure Fraction is present there.

**4.2. Alegra-XFEM Tests.** The Alegra-XFEM simulations provided new means to obtain useful information for comparison with the regular Alegra tests. Alegra-XFEM simulations capture exactly where each material lies, rather than an average

of the material present in each element. (Figure 4.6) Use of the VTK visualization with the Alegra-XFEM simulation provided a sharper image of the problem that will be useful in future examination.

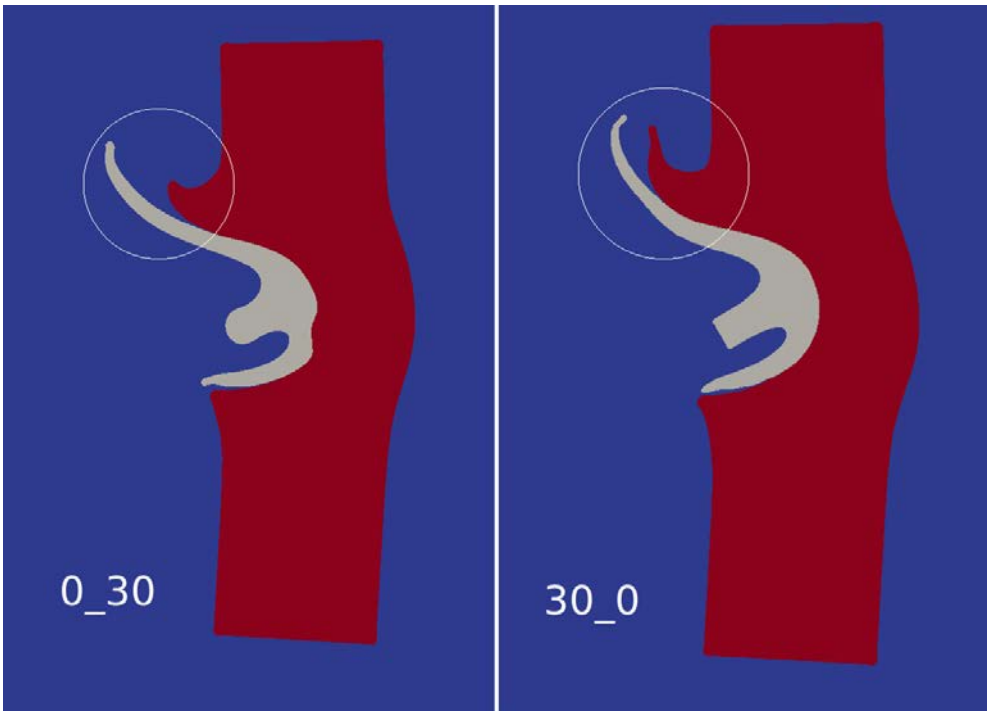



Fig. 4.6: VTK visualization output from an Alegra-XFEM simulation. Notice the critical differences in the circled areas.

Note that in the 0_30 simulation (left) the tail of the projectile has been rounded out. (Figure 4.6) This is because it does not line up with the grid and has been increasingly distorted with each step. The 30_0 simulation, however, shows that the tail of the projectile has remained square. (Figure 4.6) This is because it is in line with the mesh. This shows grid dependence, but it is an expected inaccuracy and is not the focus of this project.

**4.2.1. Damage and Failure Fraction.** The results of the Alegra-XFEM problems were displayed with a different scale than those of the earlier tests, ranging only from 0 to 0.879 for Damage and from 0 to 0.0312 for Failure Fraction. This is because 0.879 and 0.0312 are the highest values for Damage and Failure Fraction in the problem, respectively. The fact that both problems have the same scale means that they both have the same maximum values for Damage and Failure Fraction, which is a good sign.

In both output files, the damage variable showed that the 0_30 simulation sustained more damage at the impact point while the 30_0 sustained more in the ejected material. (Figure 4.7) This is somewhat in agreement with the damage variable results of the normal-Alegra testing, shown in figure 4.3

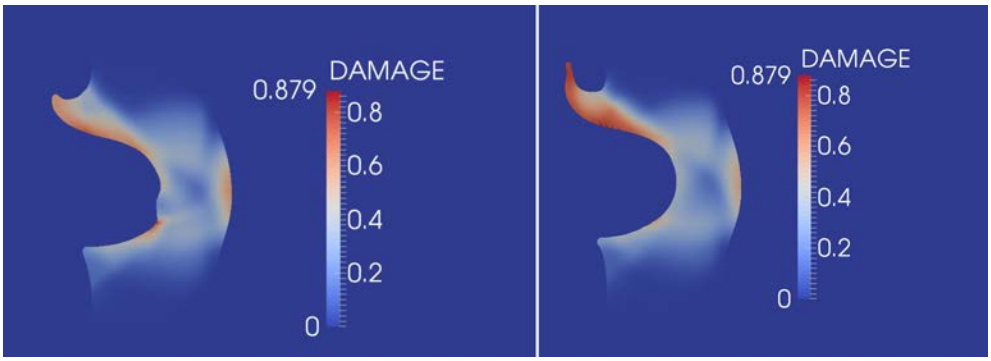



Fig. 4.7: Damage of 0_30 simulation (left) and 30_0 simulation (right)

The Failure Fraction variable more failure in the ejected particles for the 0_30 problem, and very little Failure Fraction presence in the 30_0 problem. (Figure 4.8) This is curious because, though the regular-Alegra simulations showed higher Failure Fraction in the ejected material also, the Failure Fraction in the 0_30 simulations are in entirely different places. Because both the Exodus and VTK visualization from the Alegra-XFEM simulation showed Failure Fraction in the same place, this was clearly a difference between regular-Alegra simulations and Alegra-XFEM simulations.

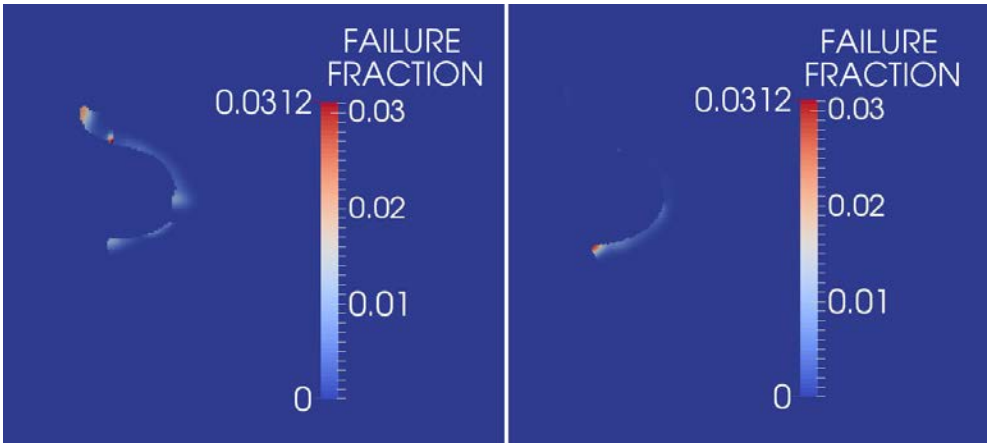



Fig. 4.8: Failure Fraction of 0_30 simulation (left) and 30_0 simulation (right)

The aforementioned unusual behavior seen in all of the regular-Alegra tests that used the Johnson Cook fracture model did not appear in any of the Alegra-XFEM simulations.

**5. Conclusion.** Due to the nature of computer simulations, small bugs and inconsistencies can cause significant problems when the program is used. This project worked to identify the source of an error in Alegra oblique impact problems. It included extensive testing of an extensive set of problems and analyzation of the results

of each. Based on the initial results, Alegra simulations' reliance on grid alignment is lessened with mesh refinement. When a new model (Johnson-Cook Fracture Model) was used, refinement did not fix the issue. This indicates that the inconsistency is caused by the material model. Overall, this project has provided preliminary investigation concerining an inconsistency in Alegra, in addition to creating a bank of data to be used in later investigations of the same issue.

# SQUADGEN GRID GENERATION: A TOOL TO FURTHER IMPROVE THE ATMOSPHERIC GENERAL CIRCULATION MODELS

SHARON M. SULLIVAN* AND MARK A. TAYLOR†

**Abstract.** SQuadGen (Spherical Quadrilateral Grid Generator) is an open source code that makes grids to be used by the Community Atmosphere Model (CAM) to create high-resolution grids to mathematically represent the complex nonlinear interactions within the climate system to change aspects of the interactions. Emphasis is made on tropical cyclones, Arctic storms, and high-intensity orographic precipitation events.

**1. SQuadGen.** The climate system is characterized by complex nonlinear interactions on different temporal and spatial scales [1]. Global atmosphere models use numerical methods to solve the coupled set of partial differential equations describing fluid motion to resolve large-scale (dynamical) atmospheric flow with coupled, ocean, land, and sea ice components [4]. There is emphasis in developing high-order numerical methods for climate modeling, such as the spectral element dynamical core and uncertainty quantification to improve the simulation of extreme events and productivity of global climate models. A variable cubed-sphere grid (shown in Figure 1.1)can be used by the Community Earth System Model (CESM) to focus on conforming quadrilateral meshes to areas of in variable resolutions [2]. Grid construction utilized the open source code, SQuadGen, which generates meshes that can be used by the Community Atmosphere Model (CAM) [4].

Fig. 1.1: The cubed-sphere grid

CAM uses regional mesh refinement to provide a framework for atmospheric flow and high-resolution measurements of climatic variables in specific regions. The Community Atmosphere Model should be able to investigate the role of seasonal and long-term climate variability, applications to environment and policy, and define the number of years that the climate system can be simulated [4]. This computationally effective tool satisfies the need of variable resolution global climate modeling, increased

*University of New Mexico, ssully10@unm.edu
†Sandia National Laboratories, mataylo@sandia.gov

resolution over defined regions, and can be used to improve forecasts through a determination of how these multi-scales interact and to mathematically represent these interactions in climate models [1]. In this paper, we will document the procedure to generate and rotate grids in SQuadGen.

**2. Global Atmosphere Models vs. Regional High-Resolution Models.** Global atmosphere models are a relatively new invention dating to about the time of World War I and World War II. Global atmosphere models allow one to chop up the atmosphere into grid cells. It is performed on a numerical basis with variables that satisfy the laws of physics that propagate through time and simulate reality. Each cell has one value of temperature, pressure, humidity, and winds in the $u$ and $v$ directions. These are called the "dynamics" (fluids) or resolved variables, where partial differential equations governing the resolved large-scale atmospheric flow are solved. The timestep matches up with the scale of variability. The physics of the model are for statistically parameterized variables that represent the unresolved small-scale components and must be solved in terms of the resolved variables [2]. Clouds and precipitation are parameterized because we cannot have one grid cell value for either of these variables.

According to Chaos Theory, errors in the initial model data will propagate into time and contaminate the forecast. The dynamics of atmospheric models rely on wind values to carry the model through time (or the so-called chaos of the equations). Most global atmospheric models rely on equations that describe the conservation of energy, mass, and momentum for each grid cell. The set of primitive equations is essentially a balance of equations. Newton's Second Law describes the components of motion associated with momentum changes upwind, the Coriolis force, and friction at the surface ($\frac{\partial u}{\partial t}$ are winds in the east-west direction and $\frac{\partial v}{\partial t}$ are the winds in the north-south). The First Law of Thermodynamics ($\frac{\partial T}{\partial t}$) describes the rate of change in energy over some timestep. These equations work best at locations in the atmosphere where resolved variables fit reality (i.e. at mid-tropospheric levels not at the surface).

Global models today are on the size of approximately 100 km and are best to show the two-way interaction of global synoptic atmospheric flows. In addition, global models are physically consistent. With each grid cell, decreasing the size of the grid cell by half requires computing power of $2^3$. We can use regional high-resolution nested models to account for surface features within a coarse global model having orders of magnitude fewer elements that can focus in specifically on one area [3]. As opposed to global models, regional models can look at higher-resolution areas, show the vertical structure of the atmosphere, and preserve topographically-forced dynamics. The timestep is globally restricted to the finest grid scale [5]. The computing power results in a 15-20x speedup with variable-resolution vs. uniform grids based on the number of elements [5]. For the same cost of a global uniform/ quasi-uniform grid, we can get higher-regional resolution, additional ensemble simulations, and longer model runs using regional meshes. Regionally-refined meshes with fewer degrees of freedom can produce the same local results as a global uniform grid [6].

**3. Procedure for New Grid in CESM.** SQuadGen must be first downloaded from the High-Order Methods Modeling Environment (HOMME )to make a grid file. The grid file is imported into GIMP (GNU Image Manipulation Program) containing two image layers that can be edited. The first layer shows the uniform grid with an average $1^\circ$ horizontal resolution. The $1.0^\circ$ resolution has strong biases and proposes to use the inexpensive regional resolution within the global model. This grid is referred

to as NE30, since there are 30x30 elements in each cube face. The second layer then can be used to show the elements that have been marked for refinement.The elements themselves must be quadrilateral and the refinement must be conforming (meaning that every edge is shared by exactly two elements) [2]. For the equatorial grids, this region of refinement can be a circle, square, or any combination of a circle/square that is top-bottom symmetric and centered in the equatorial region. Equatorial grids do not always have to be symmetric. For this particular study involving atmospheric forcing, symmetric grids were used but is not a requirement for all equatorial grids.

An image containing a white shape is created in GIMP using the white shape to mark the area of refinement and the cubed-sphere template as a guide. The image created by GIMP is saved as a `.png` file, which the SQuadGen code uses to generate the refined mesh. Parameter values can be modified for the file type, how many levels of refinement, resolution, output, refinement type (CUBIT or LOWCONN), and smoothing parameters. Cubed sphere grid refinements with certain parameters create the refined grid. The LOWCONN (low connectivity) refinement can only be used on grids with an even number of elements and also has less distortion than the CUBIT-type transition regions with angles closer to 90°. The refinement file is the name of the `.png` that was marked and saved for refinement. The refinement level can any integer based on how many levels of refinement are necessary. Two or three levels of refinement are usually used with the 1.0° and 2.0° grids. A 1.0° $\rightarrow$ 1/4° refinement means that there are two levels of refinement. The resolution is the value of the lowest refinement and can be NE15, NE16, or NE30. The NE15 resolution grid has a grid spacing of 222 km analogous to 2°x2°, while the NE30 grid has a grid spacing of 111 km. Two levels of refinement implies that a refinement level of 2 is applied twice and subdivides the grid four times. Likewise, a refinement level of 3 will have a refinement factor of 8. Smoothing parameters are used to turn on grid smoothing, which is applied in the transition region to reduce the distortion to squares. The smoothing iteration tells how many times the smoothing loop will run. Fifteen has been used for the smoothing iteration in this case. An example code for the NE120 series grid is shown:

```
./SQuadGen --refine_file ne120_equatorial.png
--refine_level 2 --resolution 30
--refine_type CUBIT --output NE30.g
--smooth_type SPRING --smooth_dist 3 --smooth_iter 15
```

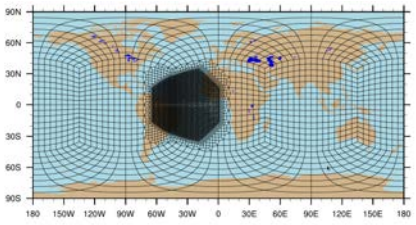
The National Center for Atmospheric Research (NCAR) in Boulder, Colorado created NCL, the NCAR Command Language, which is designed specifically for scientific data processing and visualization through the use of command line operations. The NCL script, `plot_exodus.ncl`, plots the elements and allows the new grid plot to be viewed.

**4. Developments.** The major breakthroughs of this project were the shape that was used, which template (NE60 or NE64 templates) to guide drawing the shape, and how rotations work in SQuadGen. A trial and error approach was used in the beginning and many different approaches were covered. The first attempt was a square with rounded edges ("squircle") that was hand-drawn in GIMP, but there were notches in the bottom and top of the shape when it was refined. Notches in the region of refinement make it difficult to for symmetry to be noticed. Next, a shape that consisted of a basic overlay of two circles of different sizes was used. A layer

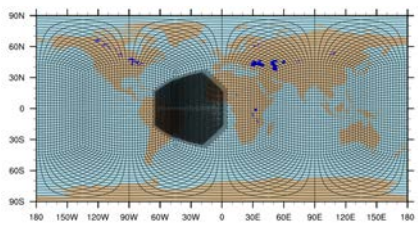
called "Greywhite" forced SQuadGen to generate two levels of refinement by having a white rectangle nested inside a larger grey rectangle. SQuadGen flips the colors that GIMP uses, so white is the area of high-level refinement and the grey would be medium-level refinement. A script in NCL was written to convert the refined mesh into a `.png` file that can be loaded back into GIMP. This allowed the asymmetric areas to be filled in for refinement. A squared-diamond shape was also created. It was symmetric and covered the correct area of refinement, but was not used due to the grid not being a favorite of the group.

For the first grid, the specifications of a 60° equatorial grid had to be met. An irregular-shaped disk was designed that spreads 60° in width. The shape follows the gridlines (straight on the vertical edges and a 45° diagonal along the curved portions of the grid where the gridlines meet). The irregular shape may not be visually appealing, but it is top-bottom symmetric and accurately takes into account flow in and out of the high-resolution nests [5]. The symmetry was particularly important to meet the first grid's criteria.

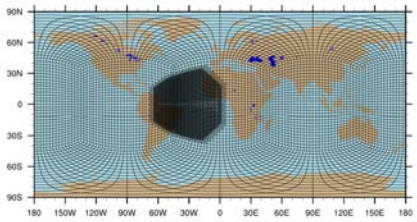
To find which template the region of refinement should be drawn on, find the resolution of the fine region and use the template with a one-half factor of the fine resolution. A 1/8° refinement is an NE16 grid with three levels of refinement. For the NE16 grid series, the region of refinement can be drawn in GIMP using the NE64 image template. Three sets of grids were generated from the irregular disk: NE16 → NE128 with 3 levels of refinement (Figure 4.1(a)) and NE32 → NE128 (Figure 4.1(b)) and NE30 → NE120 (Figure 4.1(c)) each with 2 levels of refinement respectively. The NE15 grid with three levels of refinement can be generated, but it is not an even grid. The NE60 and NE64 cannot be used interchangeably due to placement of the gridded elements.




(a) NE16 → NE128 equatorial grid




(b) NE32 → NE128 equatorial grid

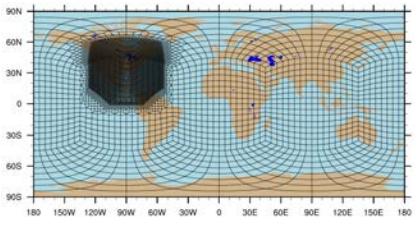



(c) NE30 → NE120 equatorial grid

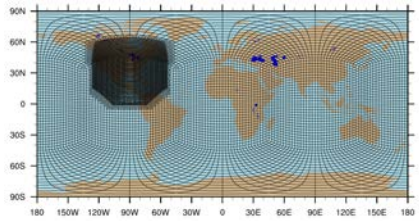
Fig. 4.1: Equatorial grid series

In order to center the region of refinement over the United States , a midlatitude rotation must be performed. SQuadGen has four parameters that can be used for

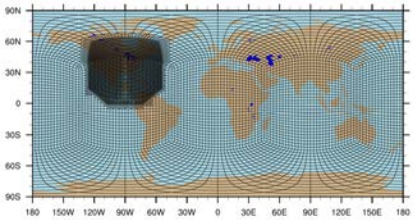
rotations: *lon_base*, *lon_shift*, *x_rotate*, and *y_rotate*. The parameters are order-dependent meaning that if y is coded before x, a rotation about the x-axis will be performed first. A midlatitude rotation is simple if a y-rotation and a z-axis are given. Another name for the z-axis in SQuadGen is *lon_base*. X- and y-rotations were used in 90° increments, where points on the cubed-sphere are always in the same location. There are sixteen different combinations of x-rotations and y-rotations from 0° to 270°, with 6 of those being rotations in the Northern Hemisphere. A 90° rotation in x rotates the region of refinement to the left 90° over the southern Atlantic Ocean from 60W to 60E and 0 to 60S. If an axis of rotation was placed through Ghana, the globe would rotate counterclockwise in the x-direction. A 90° rotation in y stretches the region of refinement from 180W to 60E and 120E to180E from 30N to 90N. It was found that the axis of rotation in the y-direction is a clockwise rotation that goes from the Galapagos Islands at 90W through the Bay of Bengal off the eastern coast of India. The combination that produces the midlatitude region of refinement is a *lon_base* of -90° and *y_rotate* 270°. Figures 4.2(a), 4.2(b), and 4.2(c) below show the midlatitude grid series that was generated from these rotations.




(a) NE16 → NE128 midlatitude grid




(b) NE32 → NE128 midlatitude grid




(c) NE30 → NE120 midlatitude grid

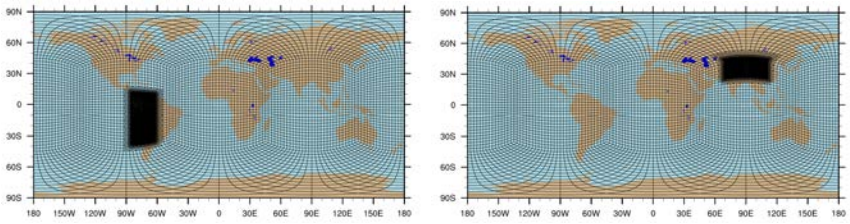
Fig. 4.2: Midlatitude grids

This step-by-step method of generating grids was an improvement over the trial and error method approached previously and seems to generate good grids. The image and .pdf files were compiled for use in the regionally refined project.

**5. Applications.** Modeling sub-synoptic scale features requires high horizontal resolution [5]. Often, high resolution data is only necessary over a small region: for making a local forecast, comparing the model to statistics gathered locally, or using the local statistics to tune the global high-resolution model based on the feedback in the refined region, to name a few reasons [2]. Some examples of these features include tropical cyclones, Arctic storms, high-intensity precipitation events, high-resolution measurements of clouds, localized air quality events, and other variables in specific regions [5]. These measurements are useful for tuning a global model, even though

the data itself is not global [2]. The goal of variable-resolution Global Circulation Models (GCM's) is to facilitate the move towards seamless prediction, a one-model fits all approach for numerical weather prediction and climate application [5].

One of these applications of regional grids allows us to place grids over areas of higher topography. In regions of rough topography, such as over the Andes and Himalayas, models often over predict precipitation due to numerical issues associated with terrain following coordinates. This bias in precipitation is often reduced in global high resolution models [4]. This has been seen if similar improvements can be obtained with selective regional refinement. We thus created a suite of regionally refined grids with a high-resolution region placed over the Andes and Himalayas (shown in Figure 5.1(a) and 5.1(b)).




(a) NE30 → NE240 Andes grid (b) NE30 → NE240 Himalaya grid

Fig. 5.1: NE60 series grids for Andes and Himalaya orographic precipitation

Another application that can be used with variable resolution is Arctic storms. The Arctic storm tracking algorithm uses a modification of the tropical storm algorithm using vorticity and pressure to identify storms and plots the tracks with NCL. The storms that are of concern are the numerous tracks of storms that would be considered a Category 1 on the Saffir-Simpson Hurricane Scale. Once the tracking algorithm is working properly to detect Arctic storms, the latitude of these storms can be determined through storm tracks and visual characteristics of the storms to suggest where to place the refined grid. The latitude of these storms suggest that a refined grid be placed in a belt-like band between 40N and 60N around the globe (Figure 5.2), as suggested by the storm tracks in reanalysis data. The refinement goes from $1\,^{\circ} \rightarrow 1/8^{\circ}$ at the poles. The refined Arctic grid contains 53,000 elements, which falls within the 10% range of the maximum computing power required to refine the elements.

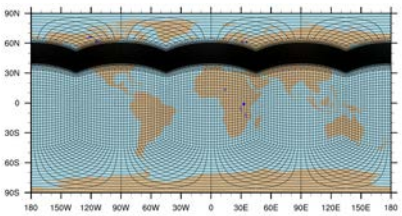



Fig. 5.2: Preliminary Arctic grid

**6. Discussion.** The equatorial disk started out being centered at 30W, but a rotation was required for the refined region to be centered over the United States. Using GIMP, it is difficult to draw the region of refinement by hand. Even with symmetric shapes as the input, the refined grid might not always come out symmetric. Once a region of refinement is created, it is difficult to move the region without some set of rotations. Coloring over the line of a grid cell will refine the next element as well.

Initially, SQuadGen would ignore the smoothing parameters completely, but the bug was eventually fixed. A shape that is not designed for visual effects, but is top-bottom symmetric accurately takes into account flow in and out of the high-resolution nests [5]. The irregular equatorial disk had a longer eastward component allows the flow to be stronger going out of the model, where equatorial pressure gradients are stronger and therefore the trade winds (northeast in the Northern Hemisphere and southeast flow in the Southern Hemisphere) are also stronger in this vicinity.

**7. Conclusion.** Cubed-sphere grids and multi-resolution, highly-scalable dynamical cores may provide opportunities to improve numerical weather prediction and regional climate forecasts [5]. In general, the model tends to oversimulate what is observable in satellite imagery. There are systematic errors in the cyclone track and intensity on $1/8°$. The parameterization comes across performance issues for bands of precipitation, allowing too much light precipitation to be simulated by the model.

The variable-resolution approach focuses on cubed-sphere computational meshes that have the potential to become a standard in future Global Circulation Models (GCMs) [1]. The regional refinements show improved resolution in limited regions without requiring a fine grid resolution throughout the entire domain. Using a refined mesh provides a method to model a high-resolution climate in a particular area of interest while producing a low-resolution simulation outside the region, reducing the computational cost of running CAM [2]. The region of refinement can be tailored towards the research problem associated with atmospheric model simulations. The variable-resolution approach bridges the scale discrepancies in key features in climate modeling between local, regional, and global phenomenon.

## REFERENCES

[1] C. Jablonowski, P. Lauritzen, M. Taylor, R. Nair, and M. Levy, *Adaptive mesh refinement and variable-resolution grids for atmospheric simulations.* http://aoss-research.engin.umich.edu/groups/admg/projects.php, 2012.

[2] M. Levy, J. Overfelt, and M. Taylor, *A variable resolution spectral element dynamical core in the community atmosphere model*, Tech. Rep. SAND: 2013-0697J, Sandia National Laboratories, 2013.

[3] M. Taylor, *The spectral finite element dynamical core in the community atmosphere model*, Proc. of KIAPS International Symposium, Seoul, CESM Atmosphere Model Working Group (2013), pp. 1–33.

[4] M. Taylor, O. Guba, and P. Ullrich, *Cam-se variable resolution capability.* http:www.cesm.ucr.edu/eventsws.2013/presentations/AMWG.neale.pdf, 2013.

[5] C. Zarzycki, C. Jablonowski, and M. Taylor, *Application of variable-resolution cam-se to simulate weather events in a global model.* Lecture at University of California Davis, http://www-personal.umich.edu/~zarzycki/presentations/zarzycki_2013_davis_ppt.pdf, 2013.

[6] ———, *Using variable resolution meshes to model tropical cyclones in the community atmosphere model*, Monthly Weather Review, 142.3 (2014), pp. 1221–1239.

# OPTIMIZATION UNDER UNCERTAINTY FOR THE SHOCKLEY AND THE DRIFT-DIFFUSION MODELS OF A DIODE

T.A. TAKHTAGANOV*, D.P. KOURI†, D. RIDZAL‡, AND E. KEITER§

**Abstract.** We solve stochastic optimal control problems involving numerical models of electrical diodes, where diode parameters are subject to uncertainty. We consider two models of a diode, an algebraic model given by the Shockley equation, and a partial differential equation model given by the drift-diffusion equations. The random parameters that define the stochastic optimal control problem are computed by solving a sequence of deterministic inverse problems based on known distributions of auxiliary variables, such as the current measurements. We solve the inverse and the optimal control problems using gradient-based optimization methods from the Rapid Optimization Library (ROL), Trilinos. We present numerical studies of the stochastic optimal control problem for a variety of risk measures implemented in ROL.

**1. Introduction.** Our goal is to formulate and solve optimization problems with uncertain parameters for the electrical circuit consisting of a voltage source $V^{src}$ and a diode connected in series, as shown in Figure 1.1. The diode is modeled as an intrinsic diode in series with a resistive load $R_S$.

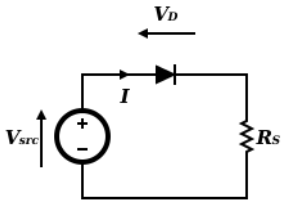



Fig. 1.1: Voltage source $V^{src}$ in series with a diode and resistor. [1]

First we consider an algebraic model of a p-n junction diode given by the Shockley diode equation. This model relates the current $I$ through a diode to the voltage $V_D$ accross it. This relationship, known as an $I$-$V$ characteristic of a diode, leads naturally to a parameter estimation problem for the intrinsic diode parameters that appear in the Shockley equation. By solving the inverse problem for the diode parameters, and a range of $I$-$V$ curves, we obtain a distribution of these parameters. We subsequently use this distribution as a sample space for the stochastic control problem. Specifically, we are interested in computing the optimal voltage source to achieve a prescribed current under uncertainty in the diode parameters.

Next we consider a partial differential equation (PDE) model of a p-n junction diode given by the drift-diffusion (DD) equations. Optimal design of semiconductors using the DD model is studied extensively in the literature [2, 4, 3]. In this work we formulate and solve an inverse problem for the doping profile of the diode using an approach similar to [4]. As in the case of the algebraic model, we then formulate a stochastic optimal control problem, with randomness introduced in the shape of the doping profile of the diode.

*Rice University, tat3@rice.edu
†Sandia National Laboratories, dpkouri@sandia.gov
‡Sandia National Laboratories, dridzal@sandia.gov
§Sandia National Laboratories, erkeite@sandia.gov

To the best of our knowledge, optimal control problems governed by diode models with uncertain parameters have not been studied previously.

This paper is structured as follows. In section 2 we describe in detail the algebraic and the PDE models. In section 3 we formulate inverse and optimal control problems of interest. We discuss various risk measures used to formulate the stochastic optimal control problem. In section 4 we describe the numerical results obtained using the Rapid Optimization Library (ROL), Trilinos. In section 5 we draw conclusions and discuss future directions.

**2. Diode modelling.** We consider two models of a p-n junction diode: an algebraic model given by Shockley diode equation and a PDE model described by the drift-diffusion system of equations. In both cases the current through the diode is the same as in the whole circuit.

**2.1. Shockley diode equation.** The Shockley diode equation or the diode law provides a formula for the diode current $I$ given voltage drop $V_D$ accross the diode:

$$I = I_S \left( \exp\left( \frac{V_D}{\eta V^{th}} \right) - 1 \right). \tag{2.1}$$

Here $I_S$ is the saturation current, $V^{th}$ is the thermal voltage ($\approx$ 26 mV at room temperature) and $\eta$ is the ideality factor that depends on the manufacturing process and ranges between 1 and 2 (for our purposes set to 1).

We use Kirchhoff's voltage law to obtain $V_D$:

$$V_D = V^{src} - IR_S. \tag{2.2}$$

Substituting (2.2) in (2.1) we obtain a formula for the circuit current given the source voltage:

$$I = I_S \left( \exp\left( \frac{V^{src} - IR_S}{\eta V^{th}} \right) - 1 \right). \tag{2.3}$$

We solve this nonlinear equation for the current using Newton's method. Two important parameters of the diode are the saturation current $I_S$ and the ohmic resistance $R_S$ (as we assume the resistor to be a part of the diode).

**2.2. Drift-diffusion formulation.** The PDE model of the diode is based on the drift-diffusion formulation [5]. This formulation consists of three coupled PDEs: a Poisson equation for the electrostatic potential ($\phi(x)$) and two continuity equations, one for the concentration of electrons, $n(x)$, and one for the concentration of holes, $p(x)$. The resulting system is as follows:

$$-\nabla \cdot \left( \frac{\epsilon}{q} \nabla \phi(x) \right) = p(x) - n(x) + C(x) \tag{2.4a}$$

$$\nabla \cdot J_n(x) = R(x) \tag{2.4b}$$

$$\nabla \cdot J_p(x) = -R(x). \tag{2.4c}$$

Here $\epsilon$ is the permittivity of the material, $q$ is the magnitude of the electron charge, and $C(x)$ is the total doping concentration which can be represented as

$$C(x) = N_D^+(x) - N_A^-(x), \tag{2.5}$$

where $N_D^+(x)$ is the concentration of positively ionized donors and $N_A^-(x)$ is the concentration of negatively ionized acceptors.

Furthermore, $R(x)$ is the generation/recombination rate for both species (electrons and holes), $J_n(x)$ and $J_p(x)$ are the current densities of electrons and holes, repectively. In the drift-diffusion model the quantities $J_n(x)$ and $J_p(x)$ are determined from

$$\begin{aligned} J_n(x) &= -\mu_n(x)n(x)\nabla\phi(x) + D_n\nabla n(x) \\ J_p(x) &= -\mu_p(x)p(x)\nabla\phi(x) - D_p\nabla p(x), \end{aligned} \tag{2.6}$$

where $\mu_n(x)$, $\mu_p(x)$ are mobilities for electrons and holes, and $D_n$, $D_p$ are diffusion coefficients. Recombination rate $R(x)$ consists of two terms: the Shockley-Read-Hall term

$$R_{SRH}(x) = \frac{n(x)p(x) - N_{in}^2}{\tau_p(x)(n(x) + N_{in}) + \tau_n(x)(p(x) + N_{in})}, \tag{2.7}$$

where $\tau_n(x)$ and $\tau_p(x)$ are the carrier lifetimes, and $N_{in}$ is the intrinsic carrier density (constant), and the Auger term

$$R_{AUG}(x) = (C_n n(x) + C_p p(x))(n(x)p(x) - N_{in}^2), \tag{2.8}$$

where $C_n$, $C_p$ are constants dependent on the material.

The boundary of the domain is split into two disjoint parts $\Gamma_N$ and $\Gamma_D$. The first one corresponds to an insulated boundary of the device, while the second corresponds to the ohmic contacts (electrodes). The boundary data is

$$\begin{aligned} n = n_D,\ p = p_D,\ \phi = \phi_D \ \text{ on } \ \Gamma_D \\ \nabla n \cdot \nu = \nabla p \cdot \nu = \nabla\phi \cdot \nu = 0 \ \text{ on } \ \Gamma_N, \end{aligned} \tag{2.9}$$

where $\nu$ is unit outward normal to the boundary and

$$\begin{aligned} n_D &= \frac{1}{2}(\sqrt{C_D^2 + 4N_{in}^2} + C_D) \\ p_D &= \frac{1}{2}(\sqrt{C_D^2 + 4N_{in}^2} - C_D) \\ \phi_D &= V^{th}\log\left(\frac{N_D}{N_{in}}\right) + V^{src}, \end{aligned} \tag{2.10}$$

with $C_D$ and $N_D$ the values of the doping and the maximum donor concentration at the Dirichlet boundary $\Gamma_D$, respectively.

For this work we consider a one-dimensional finite-difference discretization of the DD equations. Furthermore, we use the stabilization method by Scharfetter and Gummel (SG) to avoid oscillations in the solution [7]. The SG method simplifies the expressions (2.6) for the current densities by assuming Einstein relations $D_n = V^{th}\mu_n$ and $D_p = V^{th}\mu_p$, which are valid if the variables are scaled correctly (2.14). Current densities are then computed at the midpoints between mesh nodes as described on page 15 in [5]. The solution process, following the implementation in Sandia's circuit simulator Xyce, has three stages:

1. Start with the equilibrium approximation for the electron and hole concentrations and the electrostatic potential (see [5]), calculated using information about the doping profile;

2. Use the equilibrium potential as an initial guess for the nonlinear Poisson equation, given by

$$-\nabla\cdot\left(\frac{\epsilon}{q}\nabla\phi(x)\right) = N_A\exp\left(\frac{\phi^{min}-\phi(x)}{V^{th}}\right) - N_D\exp\left(\frac{\phi(x)-\phi^{max}}{V^{th}}\right)+C(x), \tag{2.11}$$

which is a modification of (2.4a) with $n(x)$ and $p(x)$ approximated by the nonlinear functions of $\phi(x)$ (here $N_A$ and $N_D$ are the maximum acceptor and donor concentrations); and

3. Solve the drift-diffusion system (2.4) using as the initial guess the solution of the nonlinear Poisson equation, $\phi_{NLP}(x)$, along with the hole and electron densities $p_{NLP}(x)$ and $n_{NLP}(x)$ computed using the expressions in (2.11).

Upon solving the drift-diffusion system we compute the electron and hole current densities $J_n(x)$ and $J_p(x)$. The total current density is given by $J(x) = J_n(x) + J_p(x)$. The current flow $I$ over a contact $\Gamma \in \Gamma_D$ is given by

$$I = \int_\Gamma J\cdot\nu ds. \tag{2.12}$$

Evaluating this integral at two contacts (electrodes), which in the one-dimensional case amounts to scaling the values of the current density $J$ at the endpoints of the interval by the "area" constant, we obtain the currents $I_{cathode}$ and $I_{anode}$. Finally, we take the average of the current flows at the electrodes to be the current in the diode (note that at one electrode the current is positive, and negative at the other):

$$I = \frac{1}{2}(I_{anode} - I_{cathode}). \tag{2.13}$$

If $N_X$ points are used to discretize the space, there are $3\times N_X$ unknown variables in the drift-diffusion system—the potential, $\phi$, the electron density, $n$, and the hole density, $p$, for each of the mesh points. To better capture the position of the p-n junction the middle interval is refined further, bringing the number of variables to $3\times(N_X+N_R)$, where $N_R$ is the number of points in the refined region. The mesh points are indexed as follows: $x_1, \ldots, x_{\frac{N_X}{2}}, x_{\frac{N_X}{2}+1}, \ldots, x_{\frac{N_X}{2}+N_R}, \ldots, x_{N_X+N_R}$.

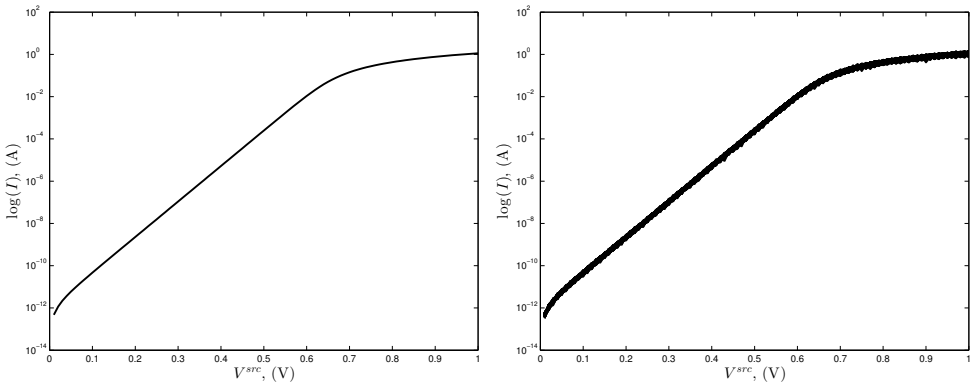
In the simulation the variables are scaled as follows:

$$\begin{aligned} N_A, N_D, N_{in} &\to \frac{N_A}{C_0}, \frac{N_D}{C_0}, \frac{N_{in}}{C_0} \\ \phi, \phi_D, n, n_D, p, p_D &\to \frac{\phi}{V_0}, \frac{\phi_D}{V_0}, \frac{n}{C_0}, \frac{n_D}{C_0}, \frac{p}{C_0}, \frac{p_D}{C_0} \\ x, \tau_n, \tau_p &\to \frac{x}{x_0}, \frac{\tau_n}{\tau_0}, \frac{\tau_p}{\tau_0}, \end{aligned} \tag{2.14}$$

where the scaling constants are defined in Table 4.5.

**3. Optimization Problems.** With the models of the diode circuit defined we are ready to formulate optimization problems.

**3.1. Parameter Estimation.** The behavior of the diode in the circuit is described by its current-voltage characteristics, or the $I$-$V$ curve (Figure 3.1(a)). We will use the following notation to denote this curve: $(I_n^{meas}, V_n^{src})_{n=1}^N$. Here $N$ is the number of measurements taken, $V_n^{src}$ is the value of the source voltage for the $n$-th measurement and $I_n^{meas}$ is the corresponding current.




(a) I-V curve (b) 1000 I-V curves with 10% noise

Fig. 3.1: $I$-$V$ curve without and with noise

A natural question to ask is "What can we infer about the diode parameters from its $I$-$V$ characterization?".

For the case of an algebraic model we are interested in recovering the saturation current of the diode $I_S$ and the resistance of the internal resistor $R_S$. For the case of a PDE model we would like to know the doping profile of the diode $C(x)$ and estimate the position of the p-n junction. In particular, we are interested in the doping values close to the center of the diode, therefore, we keep the boundary values of the doping concentration fixed.

We consider the following optimization problem:

$$\min \frac{1}{2} \sum_{n=1}^{N} (I_n - I_n^{meas})^2 \tag{3.1}$$

where $I_n = I_n(V_n^{src})$ is either the solution to the equation (2.3) or obtained from the solution of the drift-diffusion system (2.4) with $V_n^{src}$ as an input (boundary condition). For the algebraic model we minimize with respect to $I_S$ and $R_S$, while for the PDE model we minimize with respect to $C(x_i)$ with $i = 2, \ldots, N_X + N_R - 1$. Moreover, for the PDE model, in order for the resulting doping profile to be reasonably smooth, we add to the objective (3.1) a regularization term of the form

$$\frac{\gamma}{2} \int_{x_1}^{x_{N_X+N_R}} \left( \frac{\partial C}{\partial x} \right)^2 dx, \tag{3.2}$$

where the parameter $\gamma$ is chosen experimentally.

In order to use gradient-based methods from ROL we need to provide not only the objective function but also its gradient and preferrably the Hessian as well. This means we need derivatives of the current with respect to the inversion parameters. In the case of the algebraic model it is straightforward to compute the first and second order derivatives of the current $I$ with respect to $I_S$ and $R_S$. In the case of the PDE model the task becomes more complicated, as many of the parameters of the system (2.4) depend on the doping profile, including the mobilities, $\mu_n(x)$

and $\mu_p(x)$, the lifetimes, $\tau_n(x)$ and $\tau_p(x)$, and the recombination rate, $R(x)$. In our implementation we use automatic differentiation to obtain the derivatives of the current $I$ with respect to the ($\phi$,$n$,$p$) variables, and the derivatives of the ($\phi$,$n$,$p$) variables with respect to the doping variable. We use the Jacobian of the system to find the derivative of the current with respect to the doping. Formally this can be expressed as

$$\frac{\partial I}{\partial C} = \frac{\partial I}{\partial X}^T \left(\frac{\partial F}{\partial X}\right)^{-1} \frac{\partial F}{\partial C}, \tag{3.3}$$

where $X$ denotes the vector of variables ($\phi$,$n$,$p$), and the system (2.4) is represented by $F(X, C) = 0$. Furthermore, we use a "Gauss-Newton approximation" to the Hessian of the objective function. This approximation avoids the computation of the second derivatives of the governing nonlinear equation, by assuming a linear model.

To keep the optimization variables within a meaningful physical range we incorporate lower and upper bounds in the formulation. For the algebraic model we use $10^{-18} \leq I_S \leq 0.1$ and $10^{-4} \leq R_S \leq 100$. For the PDE model the constraints are placed on the values of the doping at the boundary, $-10^{15} \leq C(x_i) \leq 10^{15}$ for $i = 2, \ldots, N_X + N_R - 1$.

**3.2. Optimal Control.** The typical goal of optimal semiconductor design is to obtain the best current flow at one of the contacts [4]. We formulate a different problem, of obtaining the desired current in the diode circuit by choosing an optimal voltage source:

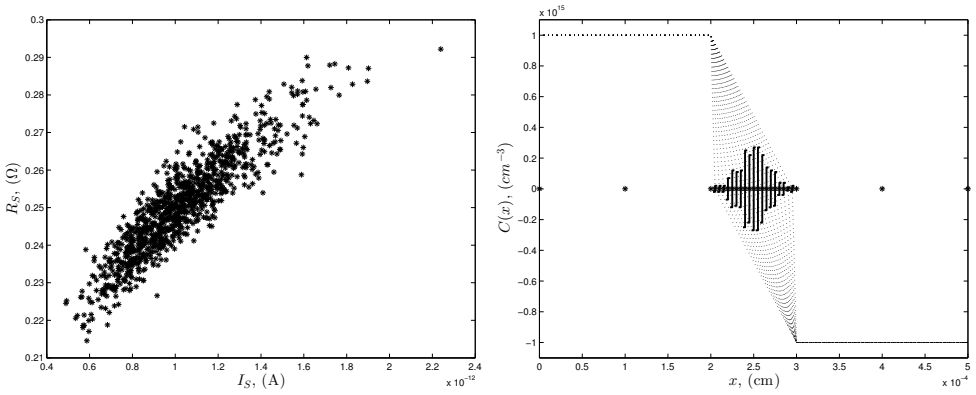
$$\min_{V^{src}} \frac{1}{2}(I - I^{target})^2 + \frac{\alpha}{2}(V^{src})^2. \tag{3.4}$$

Again, here $I = I(V^{src})$ is either the solution to the equation (2.3) or is the function of the solution to the drift-diffusion system (2.4). In the first case we compute analytic derivatives of the current with respect to the $V^{src}$. For the PDE model we differentiate the weak form, since the source voltage $V^{src}$ appears as an essential (Dirichlet) boundary condition, and use an expression similar to (3.3) with $\frac{\partial F}{\partial C}$ replaced by $\frac{\partial F}{\partial V^{src}}$.

The solution of the optimal control problem (3.4) is nearly trivial, assuming efficient solvers for the diode equations. The problem (3.4) becomes much more interesting if we introduce uncertainty in the diode parameters.

**3.3. Uncertainty and Risk Measures.** We introduce uncertainty in the optimal control problem by taking specified diode parameters to be random variables with some given distribution. For the algebraic model we choose these parameters to be the saturation current $I_S$ and the ohmic resistance $R_S$, while for the PDE model we introduce randomness in the shape of the doping profile.

To obtain the distributions of our random variables we do the following. For the algebraic model we first add random Gaussian noise to the measured currents $I_n^{meas}$ to obtain the range of I-V curves (Figure 3.1(b)), then solve inverse problems for the $I_S$ and $R_S$ parameters, for each of the I-V curves. The results of the inversion serve as the sample space for the stochastic optimal control problem, Figure 3.2(a). For the PDE model we create a random variable $\xi$ normally distibuted in the interval $[x_{\frac{N_X}{2}+1}, x_{\frac{N_X}{2}+N_R}]$, that is, between the two middle points of the coarse mesh. We then take the zero level of the doping to be at the point $x_i$ closest to the value of $\xi$




(a) Possible $I_S$, $R_S$ values.

(b) Possible doping profiles: stars are the mesh points, dotted lines correspond to the doping profiles, vertical bars show relative distribution of the profiles (here $N_X$=6, $N_R$=19).

Fig. 3.2: Sample spaces: (a) values of $I_S$, $R_S$; (b) doping profiles.

and interpolate the doping profile between the value at $x_{\frac{N_X}{2}}$ and $x_i$, and between $x_i$ and the value at $x_{\frac{N_X}{2}+N_R+1}$. We thus obtain a range of doping profiles, Figure 3.2(b).

As the parameters of our models become random variables, the solutions to (2.3) and (2.4) become random variables as well. Thus, to make sense of formulation (3.4) we need to introduce some measure of randomness. This is done by introducing risk measures. We rewrite (3.4) as

$$\min_{V^{src}} \frac{1}{2}\sigma((I - I^{target})^2) + \frac{\alpha}{2}(V^{src})^2 \tag{3.5}$$

where $\sigma$ is a risk measure and $I = I(V^{src};\xi)$ is the solution to (2.3) or (2.4) with uncertain parameter $\xi$. In the case of the algebraic formulation, $\xi$ is two-dimensional and represents possible values of $(I_S, R_S)$. In the case of the PDE formulation we take $\xi$ to be a one-dimensional uniformly distributed variable representing the position of the zero level of the doping profile.

For a given random variable $Y$ we consider the following risk measures:

1. expected value, $\sigma(Y) = \mathbb{E}[Y]$;
2. mean plus deviation of order $q$, $\sigma(Y) = \mathbb{E}[Y] + \mathbb{E}\left[[Y - \mathbb{E}[Y]]^q\right]^{1/q}$;
3. mean plus semi-deviation of order $q$, $\sigma(Y) = \mathbb{E}[Y] + \mathbb{E}\left[[Y - \mathbb{E}[Y]]_+^q\right]^{1/q}$;
4. conditional value-at-risk, $\sigma(Y) = \min_{t\in\mathbb{R}}\{t + c\,\mathbb{E}[Y - t]_+\}$.

Here "$x_+$" denotes $\max\{0, x\}$. With the first risk measure we solve for the optimal control that on average minimizes the misfit between the current in the circuit and the target current. The other risk measures provide additional safety margins [6].

**4. Numerical Results.** The inverse problem and the optimal control problem under uncertainty are implemented in the Rapid Optimization Library framework. Several optimization algorithms from ROL are tested. We obtain consistently good results using Newton-Krylov descent type methods; either the truncated Conjugate

Gradient (CG) method with trust regions ot the Newton-CG method with a line search.

For the deterministic inverse problem we generated the data $(I_n^{meas}, V_n^{src})_{n=1}^N$ by specifying either the ($I_S$,$R_S$) parameters or the doping profile $C(x)$, and running the forward simulation.

For the algebraic model the "true" values of $I_S$ and $R_S$ are chosen to be $10^{-12}$ and 0.25, respectively, and the source voltage is swept between 0 and 1 with a step of 0.01, thus producing 101 measurement points. The results of the inversion for ($I_S$,$R_S$), even for remote starting points, are almost exact with relative errors being on the order of machine precision, see Table 4.1.

| Initial guess | Obtained solution | # of iterations | Rel. error |
|---|---|---|---|
| $10^{-13}$, 0.1 | $10^{-12}$, 0.25 | 17 | 0.0 |
| $10^{-10}$, 0.5 | $10^{-12}$, 0.25 | 21 | $2.2 \times 10^{-16}$ |
| $10^{-8}$, 1.0 | $10^{-12}$, 0.25 | 38 | $1.1 \times 10^{-16}$ |

Table 4.1: Results of the inversion for $I_S$, $R_S$ using Newton-CG with line search.

For the PDE model the "true" doping profile was taken to be constant with opposite signs in the $p$ and $n$ regions with linear interpolation in the middle region of the diode. Inverting with respect to the doping profile proved to be more challenging due to the highly ill-conditioned nature of the problem.

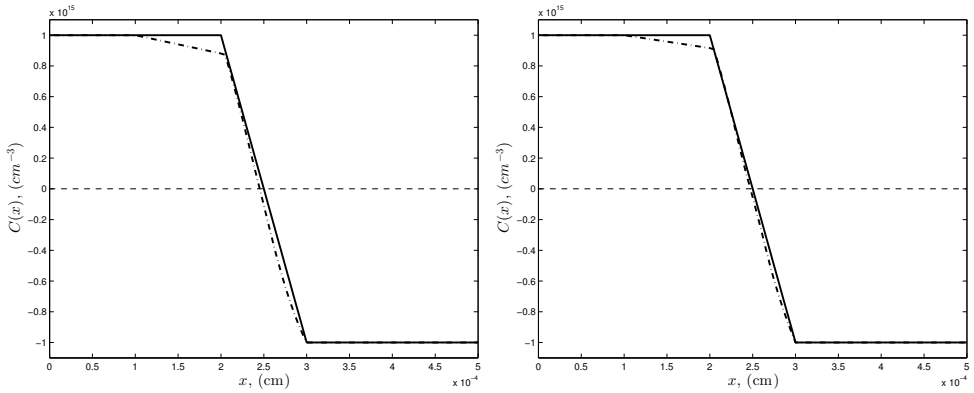
The following setup was used to test the PDE model. The length of the device was set to $5 \times 10^{-4}$. The coarse uniform discretization ($N_X$ points) was used together with a finer mesh in the middle part of the device ($N_R$ points). The values of the doping concentration were fixed at the endpoints of the domain, specifically, $C(x_1)$ was set to $10^{15}$ and $C(x_{(N_X+N_R)})$ to $-10^{15}$. The initial guess for the doping at the mesh points $x_i$, $i = 2, \ldots, N_X + N_R - 1$, was chosen to be zero. The results of the inversion for $N_X = 6$ and $N_R = 19$ and different values of $\gamma$ are plotted in the Figures 4.1(a)-4.1(d).

For the stochastic optimal control problem in the case of the algebraic model we used 1000 samples obtained from the solutions of the inverse problem (Figure3.2(a)). The goal was to then solve the problem (3.5) for each of the four risk measures described in section 3.3.

The results of running ROL with the initial guess $V_0^{src} = 0.5$, $I^{target} = 0.595083889$ (corresponding to $V^{src} = 0.85$ when $I_S = 10^{-12}$, $R_S = 0.25$), and using the truncated CG trust-region method with constraints $0.01 < Vsrc < 1.0$ are presented in Table 4.2.

| Risk measure | Resulting $V^{src}$ | Objective value |
|---|---|---|
| $\mathbb{E}[Y]$ | 0.849952396 | $6.697007 \times 10^{-5}$ |
| $\mathbb{E}[Y] + \mathbb{E}\left[[Y - \mathbb{E}[Y]]^2\right]^{1/2}$ | 0.849952396 | $6.697007 \times 10^{-5}$ |
| $\mathbb{E}[Y] + \mathbb{E}\left[[Y - \mathbb{E}[Y]]_+^2\right]^{1/2}$ | 0.849955417 | $5.366958 \times 10^{-4}$ |
| $\min_{t \in \mathbb{R}} \{t + 5\,\mathbb{E}[Y - t]_+\}$ | 0.849959059 | $2.583095 \times 10^{-3}$ |

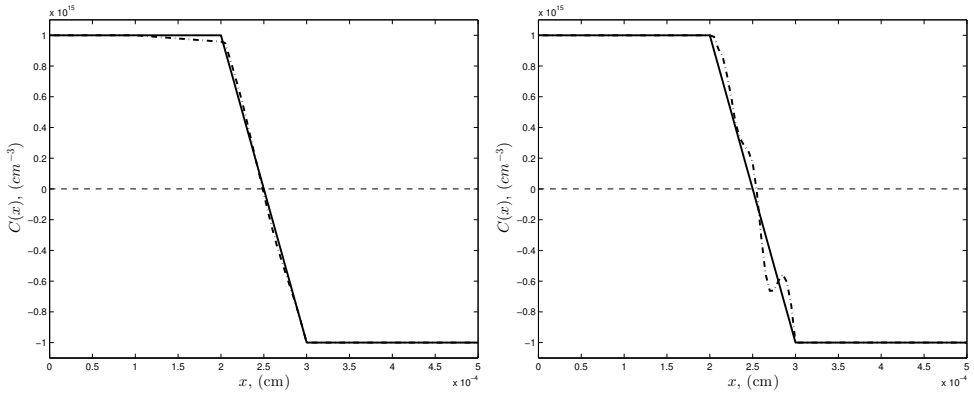
Table 4.2: Results of solving (3.5) for algebraic model




(a) $\gamma = 10^{-25}$, rel. error = 0.26




(b) $\gamma = 10^{-26}$, rel. error = 0.21




(c) $\gamma = 10^{-27}$, rel. error = 0.16




(d) $\gamma = 10^{-28}$, rel. error = 0.2

Fig. 4.1: True doping profile (solid line) and inverted doping profile (dash-dotted line)

In case of the PDE model we used 20 samples of the position of the zero-level of the doping profile (Figure 3.2(b)).

The results of running ROL with initial guess $V_0^{src} = -0.5$, $I^{target} = 1.70076740e{-}09$ (corresponding to $V^{src} = -0.14$ when $N_R = 19$, $N_X = 25$), and using Newton-CG method without constraints are presented in Table 4.3.

| Risk measure | Resulting $V^{src}$ | Objective value |
|---|---|---|
| $\mathbb{E}[Y]$ | -0.134854562 | $4.709111 \times 10^{-23}$ |
| $\mathbb{E}[Y] + \mathbb{E}\left[[Y - \mathbb{E}[Y]]^2\right]^{1/2}$ | -0.134854562 | $4.709111 \times 10^{-23}$ |
| $\mathbb{E}[Y] + \mathbb{E}\left[[Y - \mathbb{E}[Y]]_+^2\right]^{1/2}$ | -0.134611985 | $4.687500 \times 10^{-11}$ |
| $\min_{t \in \mathbb{R}} \{t + 5\,\mathbb{E}[Y - t]_+\}$ | -0.135460491 | $2.513903 \times 10^{-10}$ |

Table 4.3: Results of solving (3.5) for DD model

| Parameter | Physical meaning | Numerical value |
|---|---|---|
| q | Electric charge | $1.602176565 \times 10^{-19}$ $(A \cdot s)$ |
| $N_{in}$ | Intrinsic density | 6263659575.67454 $(cm^{-3})$ |
| $V^{th}$ | Thermal voltage | 0.02586492315346 $(V)$ |
| $\epsilon$ | Relative permittivity | 11.8 |

Table 4.4: Physical parameters for silicon

| Parameter | Description | Value |
|---|---|---|
| $C_0$ | Doping | $10^{14}(cm^{-3})$ |
| $V_0$ | Potential | $= V^{th}$ |
| $x_0$ | Length | $5 \times 10^{-4}$ |

Table 4.5: Scaling variables

**5. Conclusions.** In this work we considered the new task of solving stochastic optimal control problems involving algebraic and PDE models of diodes. We studied the Shockley diode model and the drift-diffusion model. We formulated optimal control problems using a variety of risk measures and studied their solutions. This work was done using the Rapid Optimization Library (ROL), Trilinos.

REFERENCES

[1] A. Bekhit. `https://en.wikipedia.org/wiki/File:Diode_Modelling_Image2.jpg`. [Online; accessed 17-Septermber-2014].

[2] M. Burger and R. Pinnau, *Fast optimal design of semiconductor devices*, SIAM Journal on Applied Mathematics, 64 (2003), pp. 108–126.

[3] M. Hinze and R. Pinnau, *An optimal control approach to semiconductor design*, Mathematical Models and Methods in Applied Sciences, 12 (2002), pp. 89–107.

[4] ———, *Mathematical tools in optimal semiconductor design*, BULLETIN-INSTITUTE OF MATHEMATICS ACADEMIA SINICA, 2 (2007), p. 569.

[5] E. Keiter, S. A. Hutchinson, R. Hoekstra, E. Rankin, T. Russo, and L. Waters, *Computational algorithms for device-circuit coupling*, tech. rep., Tech. Rep. SAND2003-0080, Sandia National Laboratories, Albequerque, NM, 2003.

[6] R. T. Rockafellar, *Coherent approaches to risk in optimization under uncertainty*, Tutorials in operations research, 3 (2007), pp. 38–61.

[7] D. Scharfetter and H. Gummel, *Large-signal analysis of a silicon read diode oscillator*, Electron Devices, IEEE Transactions on, 16 (1969), pp. 64–77.

## Software and High Performance Computing

The articles in this section discuss the implementation of high performance computing software for a variety of applications from high-level linear algebra operations such as solving eigenvalue problems to low-level message passing functions for MPI-based computations. The high-level computational mathematics algorithms discussed in the first four articles require large numbers of reliable operations including arithmetic and system communication. The fifth paper in this section discusses lightweight kernel operating systems, while the final two papers discuss the power issues associated with such communication schemes and present ideas to reduce the associated power requirements.

*Klinvex et al.* provide a user's manual for the TraceMin functionality in Anasazi. The TraceMin algorithm in Anasazi implements a trace minimization solver for symmetric generalized eigenvalue problems. *Peterson and Dunlavy* present a Python-based software tool for tensor computations. This tool wraps implementations of tensor data structures as well as algorithms for tensor manipulation and model fitting from the C++ tensor toolkit SWIG. *Schroots and Moreland* discuss the implementation of parallel algorithms using the Dax Toolkit. The Dax Toolkit implements infrastructure to exploit concurrency in data anaysis and visualization. The authors demonstrate this exploited concurrency through the implementation of parallel merge sort and marching tetrahedra. *Slota et al.* discuss the implementation of graph-based analytics algorithms on manycore systems such as GPUs. The authors present a GPU implementation of the breadth-first search and color propogation subroutines common in graph-based analytics. This implementation uses the Kokkos package and achieves a considerable speed up over the state-of-the-art. *Cabrera and Pedretti* implement a function shipping layer for the Kitten operating system. This function shipping layer offloads I/O from the compute nodes to specialized I/O nodes. Their implementation leverages specific features of high performance computing networks to move the I/O data efficiently. *Groves and Ferreira* hypothesize that one can reduce peak power consumption during system communication operations by adjusting meassage size. The authors evaluate this hypothesis by testing in the MPI framework. They found that a reduction in power consumption is possible, but results in extreme costs to bandwidth and latency. *Jean-Baptiste and Lofstead* discuss the extreme costs of high volume transfers between compute nodes and staging area machines in Integrated Application Workflows and provide a software solution called Delta. Delta provides an algorithm to identify and prune duplicate information prior to transfer and then restore these copies following transfer.

D.P. Kouri
M.L. Parks

December 18, 2014

# ANASAZI TRACEMIN USER MANUAL

ALICIA M. KLINVEX *, MICHEAL A. HEROUX †, MICHEAL L. PARKS ‡, AND KAREN D. DEVINE §

**Abstract.** Sparse symmetric eigenvalue problems arise in many computational science and engineering applications: in structural mechanics, nanoelectronics, spectral reordering, and Google's PageRank link analysis, for example. Often, the large size of these problems requires the development of eigensolvers that scale well on parallel computing platforms. In this document, we will present three such eigensolvers recently developed for the Anasazi package of Trilinos: TraceMin, TraceMin-Davidson, and Jacobi-Davidson. These methods are different from many other eigensolvers in that they do not require accurate linear solves to be performed at each iteration in order to find the smallest eigenvalues and the associated eigenvectors. After reading this document, you should have a reasonable understanding of how these methods work and how to use them in your research.

**1. Introduction.** Our goal is the solution of symmetric generalized eigenvalue problems of the form

$$Ax = \lambda Bx \tag{1.1}$$

where $A$ and $B$ are both symmetric sparse matrices, with $B$ being positive definite. Although $A$ and $B$ are large $n \times n$ matrices, we are only interested in finding the $p$ smallest eigenpairs, where $p \ll n$. Such problems arise in many applications, such as determining automotive acoustics, modeling particles moving through electric fields via the Schrödinger equation, and spectral reordering. Since these problems can be quite large ($n \sim 10,000,000$ or greater), we want code that can run efficiently on a large parallel cluster. To solve this problem, we have implemented several trace-minimization eigensolvers within the Trilinos framework.

The goal of this document is to get you to use our eigensolvers as quickly and painlessly as possible. First, we will give a basic overview of what Trilinos is and how it can benefit you. Then, we will explain the theory behind our eigensolvers and some of the important implementation details. We will also present a few examples demonstrating how to use TraceMin in various situations.

**2. A Brief Overview of Trilinos.** Trilinos is a collection of open source software libraries (Table 2.1) intended to be used as building blocks for the development of scientific applications. We have chosen to implement our solver in the Trilinos framework because it provides so many of the distributed memory kernels we need. Trilinos code is templated to perform efficiently on many different types of architectures, and it has impressive scalability. It is capable of performing operations with matrices that have more than 2 billion rows, and most packages only require the action of a matrix on a vector, rather than having to explicitly store the matrix. Unlike PETSc and the associated eigensolver package SLEPc, Trilinos is capable of handling operations and solves with multiple vectors rather than one at a time. As a result, Trilinos is capable of grouping its communications during a linear solve, and will probably scale better.

*Purdue University, aklinvex@purdue.edu
†Sandia National Laboratories, maherou@sandia.gov
‡Sandia National Laboratories, mlparks@sandia.gov
§Sandia National Laboratories, kddevin@sandia.gov

| Package | Description |
|---|---|
| Anasazi | The Eigensolver Package |
| Belos | The Iterative Solver Package<br>Our trace-minimization solvers require the solution of a linear system at each iteration (when computing the smallest eigenpairs of a matrix or solving a generalized eigenvalue problem). |
| Ifpack<br>Ifpack2 | The Preconditioner Packages<br>Preconditioners can help iterative solvers perform better by improving the spectrum of eigenvalues of the operator. |
| Epetra<br>Tpetra | The Container Packages<br>These packages define sparse matrices, dense matrices, and multivectors. They also contain basic linear algebra operations such as dot products and matvecs. Tpetra is the newer version which has native MPI+X support and support for arbitrary data types. |

Table 2.1: Relevant Packages of Trilinos

**3. The Trace Minimization Algorithms.** In this section, we outline the three trace-minimization eigensolvers of Trilinos: TraceMin (with a static subspace dimension), TraceMin-Davidson (which uses an expanding subspace), and Jacobi-Davidson.

**3.1. TraceMin with a Static Subspace Dimension.** TraceMin is an eigensolver developed by Ahmed Sameh and John Wisniewski in 1982 [5]. It is based on the following observation derived from the Courant-Fischer theorem.

$$\min_{Y^T B Y = I} \text{trace } \left(Y^T A Y\right) = \sum_{i=1}^{p} \lambda_i \tag{3.1}$$

where $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_p < \lambda_{p+1} \leq \cdots \leq \lambda_n$. The $Y \in \mathbb{R}^{n \times p}$ which minimizes that trace is the set of eigenvectors corresponding to the smallest eigenvalues.

Our eigensolver is iterative (as are all eigensolvers), which means that we start with some initial guess and gradually refine it until we converge to the desired eigenpairs. We wish to find a sequence of iterates $Y_{k+1} = (Y_k - \Delta_k) S_k$ where both $Y_k$ and $Y_{k+1}$ form a section of the eigenvalue problem with trace $\left(Y_{k+1}^T A Y_{k+1}\right) <$ trace $\left(Y_k^T A Y_k\right)$. $Y \in \mathbb{R}^{n \times p}$ forms a section of the eigenvalue problem if

$$Y^T A Y = \Sigma \text{ and } Y^T B Y = I \tag{3.2}$$

where $\Sigma$ is a diagonal matrix whose diagonal entries approximate the eigenvalues; these are what we refer to as "Ritz values".

**Computing the Adjustment $\Delta_k$.** We find our update $\Delta_k$ by solving the following constrained minimization problem

$$\min_{Y_k^T B \Delta_k = 0} \text{trace } \left( \left(Y_k - \Delta_k\right)^T A \left(Y_k - \Delta_k\right) \right) \tag{3.3}$$

If $A$ is symmetric positive definite, solving problem 3.3 is equivalent to solving

$$\min_{Y_k^T B d_i = 0} \left( \left(y_i - d_i\right)^T A \left(y_i - d_i\right) \right) \qquad \forall 1 \leq i \leq p \tag{3.4}$$

where $y_i$ is the $i$th column of $Y_k$ and $d_i$ is the $i$th column of $\Delta_k$. Using Lagrange multipliers, we see that problem 3.4 is equivalent to

$$\begin{bmatrix} A & BY_k \\ Y_k^T B & 0 \end{bmatrix} \begin{bmatrix} \Delta_k \\ L_k \end{bmatrix} = \begin{bmatrix} AY_k \\ 0 \end{bmatrix} \tag{3.5}$$

where $2L_k$ represents the Lagrange multipliers.

**Computing $S_k$, i.e. Forming a Section.** Given that

$$V_k = Y_k - \Delta_k \tag{3.6}$$

consider the following small dense eigenvalue problem.

$$\left(V_k^T A V_k\right) S_k = \left(V_k^T B V_k\right) S_k \Omega_k \tag{3.7}$$

It is straightforward to prove that $Y_{k+1} = V_k S_k$ forms a section of our original eigenvalue problem, where $S_k$ is the set of eigenvectors of 3.7. Alternatively, we could orthonormalize $V_k$ via Gram-Schmidt (or any other orthogonal factorization process) and then solve a small standard eigenvalue problem $\left(V_k^T A V_k\right) S_k = S_k \Omega_k$ to obtain the same result.

**Convergence Properties and Ritz Shifts.** TraceMin has global linear convergence. The global part is good, as it means we will always converge to the eigenpairs of interest, given enough time. Linear convergence is not ideal; it means that the residual will decrease a great deal over the first few iterations but will then decrease very slowly over the rest of the iterations. The convergence rate of the $i$th eigenpair is bounded by the quantity

$$\frac{\lambda_i}{\lambda_{s+1}} \tag{3.8}$$

where $s$ is our subspace dimension, i.e. the number of vectors in $V$.

The convergence rate of TraceMin is very good if the eigenvalues are well separated and near the origin, as in $\frac{\lambda_i}{\lambda_{s+1}} \approx 0$. If the eigenvalues are poorly separated and far from the origin, $\frac{\lambda_i}{\lambda_{s+1}} \approx 1$ and the convergence rate is terrible. Fortunately, we can solve this problem by using Ritz shifts. Instead of solving $Ax = \lambda Bx$ with convergence rate $\frac{\lambda_i}{\lambda_{s+1}}$, we will solve

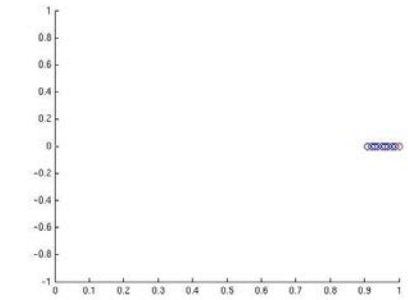
$$(A - \sigma B)\, x = (\lambda - \sigma)\, Bx \tag{3.9}$$

with convergence rate

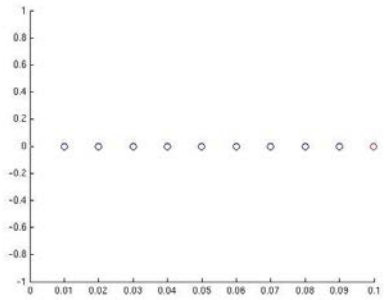
$$\frac{\lambda_i - \sigma}{\lambda_{s+1} - \sigma} \tag{3.10}$$

where $\sigma$ is our chosen shift. If $\sigma$ approximates our smallest eigenvalue, the convergence rate is greatly improved.

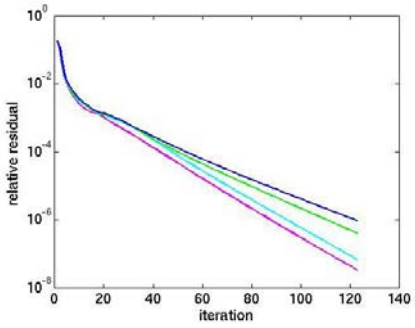
To demonstrate the impact of Ritz shifts, we generated a random sparse matrix in Matlab of order 1000 with eigenvalues equally spaced in the interval $[0.91, 10.9]$. We are seeking the 4 smallest eigenpairs using a subspace dimension of 9. A comparison of the results obtained with a Ritz shift of 0.9 and the results obtained with no shifting can be found in Figure 3.1. Note that TraceMin with shifts took only 10



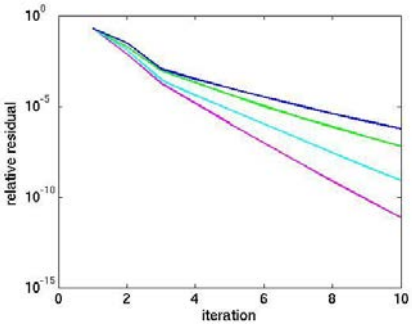
(a) A bad eigenvalue distribution (without shift)



(b) A better eigenvalue distribution (with shift)




(c) Convergence plot (without shift)




(d) Convergence plot (with shift)

Fig. 3.1: A demonstration of the importance of Ritz shifts

iterations to converge to a relative residual of $10^{-6}$, but TraceMin with no shifts took 120 iterations.

We must be careful when selecting these shifts. If our shift is too small, we don't improve our convergence rate much. If our shift is too large, we may destroy TraceMin's global convergence. (TraceMin will find the eigenvalues closest to the shift. If we had chosen 1 as the shift in our example rather than 0.9, TraceMin would *not* have converged to the desired eigenvalues 0.91, 0.92, 0.93, and 0.94 as those are not the closest eigenvalues to 1.) Our Ritz shift selection process is based on the Ritz values, the residuals, and the clustering of the eigenvalues; this process is detailed in [4].

Note that while our example only used one Ritz shift, we can use a separate shift for each right hand side of the saddle point problem to improve convergence. Each shift should approximate the eigenvalue it corresponds to. For instance, the magenta (bottom) line in Figure 3.1(d) shows that the first eigenpair converged after only 5 iterations, but the navy blue (top) line corresponding to the fourth eigenpair took twice as many. That means our shift of 0.9 was a really good choice for the first eigenpair, but the fourth would have been better off with a shift closer to 0.94. Instead

of using the same shift for all eigenpairs, which gives us the saddle point problem

$$\begin{bmatrix} A-\sigma B & BY_k \\ Y_k^T B & 0 \end{bmatrix} \begin{bmatrix} \Delta_k \\ L_k \end{bmatrix} = \begin{bmatrix} (A-\sigma B)\, Y_k \\ 0 \end{bmatrix} \tag{3.11}$$

we can use a separate shift for each and solve this set of problems:

$$\begin{bmatrix} A-\sigma_i B & BY_k \\ Y_k^T B & 0 \end{bmatrix} \begin{bmatrix} d_i \\ l_i \end{bmatrix} = \begin{bmatrix} (A-\sigma_i B)\, y_i \\ 0 \end{bmatrix} \tag{3.12}$$

As a result, the convergence rate of each eigenpair will be dramatically improved rather than just the first one.

The choice of our subspace dimension $s$ is very important. If we choose $s$ to be too small, the convergence rate is terrible. If it is too large, we will converge in a lower number of iterations, but each iteration will be very expensive since it involves solving $k$ linear systems. In practice, $s = 2p$ or $s = 3p$ are usually good choices.

Algorithm 1 summarizes the overall process of computing the smallest eigenpairs.

**Algorithm 1** TraceMin (with a fixed subspace dimension)

1: Choose an initial $V$
2: **repeat**
3: Form a section from $V$

$$Y^T AY = \Sigma \text{ and } Y^T BY = I$$

where $\Sigma$ is a diagonal matrix whose diagonal entries approximate the eigenvalues
4: Compute the residual $R = AY - BY\Sigma$
5: Choose the Ritz shifts $\sigma_i$ based on the Ritz values and residuals
6: Solve the saddle point problems $1 \le i \le s$

$$\begin{bmatrix} A-\sigma_i B & BY \\ Y^T B & 0 \end{bmatrix} \begin{bmatrix} d_i \\ l_i \end{bmatrix} = \begin{bmatrix} (A-\sigma_i B)\, y_i \\ 0 \end{bmatrix} \tag{3.13}$$

7: Update $V = Y - \Delta$
8: **until** converged

**3.1.1. What makes TraceMin so special?.** Many eigensolvers such as Krylov-Schur depend on solving linear systems to a great degree of accuracy (when finding the smallest eigenpairs). The trace-minimization eigensolvers presented in this chapter do *not* require the saddle point problems to be solved very accurately. Even if a matrix is ill-conditioned or too large to be factored, or if we are incapable of constructing a good preconditioner, TraceMin will still converge.

TraceMin has been tested against numerous eigensolvers and has proven to be competitive in terms of robustness, speed, and scalability [1].

**3.2. Jacobi-Davidson.** Jacobi-Davidson is a similar eigensolver developed by Gerard Sleijpen and Henk Van der Vorst in 1996 [6]. There are only two differences between TraceMin and Jacobi-Davidson:

- Jacobi-Davidson uses expanding subspaces unlike TraceMin's constant subspace dimension.

- Jacobi-Davidson uses the Ritz values as shifts at each iteration, whereas TraceMin uses a complicated shift-selection algorithm to preserve convergence.

These differences have been highlighted in Algorithm 2.

**3.2.1. Handling the expanding subspaces.** First, we must introduce some additional constants. Here, we redefine $s$ as the block size, the number of vectors we add to the subspace at each iteration. The block size should always be greater than the multiplicity of the desired eigenvalues, i.e. if one of the eigenvalues you desire has a multiplicity of 4, $s$ should be at least 4, or else Jacobi-Davidson may not capture the correct multiplicity. We define $m$ as the maximum subspace dimension, $c$ as the current subspace dimension, and $r$ as the restart dimension. For simplicity, we will assume $r = s$, though it may be greater. A reasonable value for the maximum subspace dimension $m \geq 2s$ is $10p$, where $p$ is the number of desired eigenvalues.

Like before, we start by forming a section from $V \in \mathbb{R}^{n \times c}$. This involves computing the eigenvectors of $\left(V^T AV\right) X = \left(V^T BV\right) X\Omega$. Previously, we computed $Y = VX$; now, we use only the $s$ eigenvectors corresponding to the smallest eigenvalues. Computing the residual does not change, and neither does solving the saddle point problems (although it should be noted that we only solve $s$ linear systems at each iteration). Instead of taking $V = Y - \Delta$ as we did before, we now expand $V$ by inserting $\Delta$ at the end of it, so $V = [V\ \Delta]$. When $V$ gets to be too large, meaning $c + s > m$, we restart. When we restart, we reinitialize $V = Y$ and continue the iterations, meaning we keep the most meaningful part of the subspace and discard the rest.

**3.2.2. Why Jacobi-Davidson's Ritz shift selection is problematic.** The Ritz values will always be greater than the corresponding eigenvalues (because we are minimizing the trace). For the first few iterations of these eigensolvers, the Ritz values provide an extremely poor estimate for the eigenvalues. If we choose our shifts to be equal to the Ritz values, global convergence is not preserved. That means *Jacobi-Davidson is not a globally convergent eigensolver.* Sometimes, the Ritz shifts will lead Jacobi-Davidson far away from the desired eigenpairs, and while the eigenpairs it returns are in fact valid eigenpairs, they may not be the smallest ones. We decided to make Jacobi-Davidson available in Trilinos because it is a very popular method, but *we do not recommend using it.* Please consider running TraceMin-Davidson instead, which will be described in the next section.

If you have run Jacobi-Davidson in the past, you may be wondering why it worked for you. Some implementations of Jacobi-Davidson, such as the one in SLEPc, do very clever things to try and maintain global convergence. SLEPc's implementation does not use any Ritz shifts until the residuals become very small, which makes their implementation closer to TraceMin-Davidson than the original Jacobi-Davidson method.

**3.3. TraceMin-Davidson.** TraceMin-Davidson is a variant of TraceMin that uses expanding subspaces like Jacobi-Davidson, but is still globally convergent[4]. The only difference between TraceMin-Davidson and Jacobi-Davidson is that we use the same criteria for the Ritz shifts here as we do in TraceMin; we do not use the Ritz values as shifts, meaning we preserve the global convergence. This difference has been highlighted in Algorithm 3.

**Algorithm 2** Jacobi-Davidson

Choose an initial $V$
**repeat**
  Form a section from $V$

$$Y^T AY = \Sigma \text{ and } Y^T BY = I$$

  where $\Sigma$ is a diagonal matrix whose diagonal entries approximate the eigenvalues and $Y \in \mathbb{R}^{n \times s}$
  **if** the subspace is full (meaning $c + s > m$) **then**
    Shrink the subspace, reinitializing $V = Y$
  Compute the residual $R = AY - BY\Sigma$
  Choose the Ritz shifts $\sigma_i$ to be equal to the Ritz values
  Solve the saddle point problems $1 \leq i \leq s$

$$\begin{bmatrix} A - \sigma_i B & BY \\ Y^T B & 0 \end{bmatrix} \begin{bmatrix} d_i \\ l_i \end{bmatrix} = \begin{bmatrix} (A - \sigma_i B)\, y_i \\ 0 \end{bmatrix} \tag{3.14}$$

  Update $V = [V\ \Delta]$
**until** converged

**Algorithm 3** TraceMin-Davidson

Choose an initial $V$
**repeat**
  Form a section from $V$

$$Y^T AY = \Sigma \text{ and } Y^T BY = I$$

  where $\Sigma$ is a diagonal matrix whose diagonal entries approximate the eigenvalues and $Y \in \mathbb{R}^{n \times s}$
  **if** the subspace is full (meaning $c + s > m$) **then**
    Shrink the subspace, reinitializing $V = Y$
  Compute the residual $R = AY - BY\Sigma$
  Choose the Ritz shifts $\sigma_i$ based on the Ritz values and residuals
  Solve the saddle point problems $1 \leq i \leq s$

$$\begin{bmatrix} A - \sigma_i B & BY \\ Y^T B & 0 \end{bmatrix} \begin{bmatrix} d_i \\ l_i \end{bmatrix} = \begin{bmatrix} (A - \sigma_i B)\, y_i \\ 0 \end{bmatrix} \tag{3.15}$$

  Update $V = [V\ \Delta]$
**until** converged

**4. The General Structure of Anasazi.** You may have noticed that for every solver in Anasazi, there is a class called [Solver Name] and one named [Solver Name]SolMgr; for instance, you will find TraceMin and TraceMinSolMgr. The class TraceMin implements a basic TraceMin iteration. It handles the solution of saddle point problems, orthogonalization, and other things of that nature. The solver manager handles the locking of converged eigenpairs (i.e. moving them to a separate location so we don't continue to do unnecessary computations with them) and the

restarting process (for eigensolvers that use expanding subspaces). When you wish to use Trilinos' eigensolvers, you should always create a solver manager, which will manage the solver class for you.

Each solver and solver manager in Anasazi is templated on three parameters: `ScalarType`, `MV`, and `OP` as in

```
TraceMinDavidsonSolMgr< ScalarType, MV, OP >
```

`ScalarType` defines the precision you wish to use; one example is `double`. `MV` is the type of multivector being used to store your eigenvectors, such as `Tpetra::Multivector`. `OP` is the operator type, an example of which is `Tpetra::Operator`.

The solver manager constructors only take two parameters:

```
BlockDavidsonSolMgr(
  const Teuchos::RCP< Eigenproblem< ScalarType, MV, OP > > & problem,
  Teuchos::ParameterList & pl )
```

The first parameter is an RCP, or Reference-Counted Pointer, to an Eigenproblem. You can think of an RCP as a wrapper around a standard C-style pointer. An Eigenproblem contains info about the problem you wish to solve, such as the stiffness and mass matrices of your problem, the preconditioner you want to use (if you want to use one), and the number of eigenvalues you desire. The second item is a ParameterList. A parameter list stores all the optional arguments you might want to give the solver. Each solver in Anasazi has slightly different parameters. The valid parameters of our trace-minimization eigensolvers are described in the next section.

**4.1. TraceMin parameters.** These parameters determine how Anasazi's trace-minimization eigensolvers perform. Don't be intimidated by the long list of options! All of these are optional, and if you're ever unsure which parameters are optimal for your particular problem, you can always contact us. Also, these eigensolvers will check your parameter list to make sure everything is valid before they do anything else. If you passed in an invalid parameter, the solver manager should catch it and provide a clear error message explaining what went wrong and how to fix the problem.

**Which.** Specifies whether we want to find the largest ("LM") or smallest ("SM") eigenpairs. Default: "SM".

Note: For most of the eigensolvers in Anasazi, the "Which" parameter does nothing more than change how the eigenvalues are sorted (and therefore which ones are saved upon restarting). When you ask the trace-minimization eigensolvers for the largest eigenpairs, the solver manager will perform a spectral transformation for you based on the following observation: if $(\lambda_1, x_1)$ is the smallest eigenpair of $Bx = \lambda Ax$, then $\left(\lambda_1^{-1}, x_1\right)$ is the largest eigenpair of $Ax = \lambda Bx$. If you want to find the largest eigenpairs of a standard eigenvalue problem, these eigensolvers will *never* have to solve a single linear system.

Another note: If you want to find the largest eigenpairs of a standard eigenvalue problem, you can not use Ritz shifts. Using Ritz shifts would force us to solve linear systems at each iteration with the operator $I + \sigma A$, which is probably more expensive than it's worth.

**Convergence Tolerance.** Specifies how small the residual norms must be before we consider them to be converged. Default: machine precision.

**Relative Convergence Tolerance.** Specifies whether residual norms should be scaled by the corresponding eigenvalues for the purpose of deciding convergence. Default: true.

**Convergence Norm.** Specifies the norm for convergence testing, either "2" or "M". Default: "2".

**Use Locking.** Specifies whether the algorithm should employ locking of converged eigenpairs. Default: true.

If locking is enabled, once the residual associated with a given eigenpair becomes smaller than a certain threshold, that eigenvector is moved to the set of auxiliary vectors. That prevents us from doing a lot of unnecessary computations with a vector that has already converged. We strongly recommend leaving this option enabled if you are using Ritz shifts; otherwise, it should not greatly impact the performance of your program.

**Max Locked.** Specifies the maximum number of eigenpairs to be locked. Default: the number of desired eigenvalues.

**Locking Quorum.** Specifies the number of eigenpairs that must meet the locking criteria before locking actually occurs. Default: 1.

**Locking Tolerance.** Specifies how accurate the residual norms must be before we lock the associated eigenpair. Default: Convergence Tolerance / 10.

Must be at least as small as the convergence tolerance.

**Relative Locking Tolerance.** Specifies whether residual norms should be scaled by the corresponding eigenvalues for the purpose of deciding whether to lock that eigenpair. Default: true.

Should be the same as Relative Convergence Tolerance.

**Locking Norm.** Specifies the norm for testing whether to lock a given eigenpair, either "2" or "M". Default: "2".

Should be the same as Convergence Norm.

**When To Shift.** Specifies when Ritz shifts should be performed. Options are "Never", "After Trace Levels", and "Always". Default: "Always".

"After Trace Levels" means our eigensolver will only shift after the quantity

$$\frac{\operatorname{trace}\left(Y_{k-1}^T A Y_{k-1}\right) - \operatorname{trace}\left(Y_k^T A Y_k\right)}{\operatorname{trace}\left(Y_{k-1}^T A Y_{k-1}\right)}$$

becomes smaller than a certain threshold, specified by the parameter "Trace Threshold". This "Trace Threshold" has a default value of 0.02.

**How To Choose Shift.** Specifies how to choose the Ritz shifts, assuming Ritz shifts are being used. Options are "Largest Converged", "Adjusted Ritz Values", and "Ritz Values". Default: "Adjusted Ritz Values".

"Largest Converged" means that we will not shift until one of the eigenpairs is locked, and then we will use the largest locked eigenvalue as a shift. This method is safe, but it will be slow.

"Adjusted Ritz Values" means that the Ritz shifts will be chosen based on the Ritz values and their associated residuals in a way that preserves global convergence. This method is described in [4].

"Ritz Values" means that the Ritz shifts will be chosen to be equal to the Ritz values. Note that this does *not* guarantee global convergence. This option is only provided in order to support Jacobi-Davidson.

**Use Multiple Shifts.** Specifies whether to use one or many Ritz shifts (assuming shifting is enabled). Default: true.

**Saddle Solver Type.** Specifies how to solve the saddle point problem arising at each iteration. Current options are "Projected Krylov" and "Schur Complement". Default: "Projected Krylov".

The option "Projected Krylov" transforms our saddle point problem into the following linear system

$$P\left(A-\sigma_i B\right)Pd_i = P\left(A-\sigma_i B\right)y_i \tag{4.1}$$

with

$$P = I - BY\left(Y^T B^2 Y\right)^{-1} Y^T B \tag{4.2}$$

Since Trilinos is templated on operators rather than matrices, we never have to form the large dense matrix $P\left(A-\sigma_i B\right)P$. However, each application of that operator will involve two sets of inner products and a matrix-vector multiplication, so it may not perform very well. Since we are projecting the operator, we must also project our preconditioner. If we wish to use a preconditioner $M \approx A$, $F = (PMP)^{-1}$ must be applied as

$$F = \left[I - M^{-1}BY\left(Y^T BM^{-1}BY\right)^{-1}Y^T B\right]M^{-1} \tag{4.3}$$

as shown in [3].

The option "Schur Complement" solves the Schur complement system by computing $z_i = \left(A-\sigma_i B\right)^{-1}By_i$ using an iterative method, then using Gaussian elimination on the resulting small linear system in $\Delta = Y + Z\left(Y^T BZ\right)^{-1}$. Because we only compute the Schur complement approximately, we do not necessarily preserve the orthogonality between $\Delta$ and $Y$. This option is not very numerically stable as a result. However, since we do not have to perform projections at each iteration of the Krylov solver, this option may yield better performance.

We are currently working on providing additional options for solving the saddle point problem, such as the block diagonal preconditioner of [2].

**Verbosity.** Specifies how much output you want. Accepts a sum of the following MsgTypes: Errors, Warnings, IterationDetails, OrthoDetails, FinalSummary, TimingDetails, StatusTestDetails, and Debug. Default: Errors.

If you want to really understand how TraceMin works and how it selects the Ritz shifts, you should use Debug. If you are running tests where time is important, you should use Errors only.

**Block Size.** For TraceMin-Davidson, this specifies the number of vectors being added to the subspace at each iteration with a default value of 1. Remember that this should always be at least as large as the multiplicity of the eigenvalues you seek.

For TraceMin, this specifies the number of vectors we work with at each iteration (since we're not using expanding subspaces). A larger block size means more work per iteration, but it may also decrease the number of iterations required. The default is 2 * the number of eigenvalues we seek.

**Num Blocks.** Only valid for TraceMin-Davidson, not TraceMin. Specifies the maximum number of blocks in the subspace. After we compute this many blocks, we will restart. Default: 10 * the number of eigenvalues we seek / block size.

**Num Restart Blocks.** Only valid for TraceMin-Davidson, not TraceMin. Specifies how many blocks we keep when restarting. Default: 2 * the number of eigenvalues we seek / block size.

**Maximum Restarts.** Only valid for TraceMin-Davidson, not TraceMin. Specifies the maximum number of restarts to be performed. Default: 50.

**Maximum Iterations.** Only valid for TraceMin, not TraceMin-Davidson. Specifies the maximum number of TraceMin iterations to be performed. Default: 100.

**5. Conclusions.** In this document, we presented a comprehensive overview of three trace-minimization schemes: TraceMin, TraceMin-Davidson, and Jacobi-Davidson. These methods are very appealing in that they do not require accurate linear solves to be performed at each iteration, like many other eigensolvers. We briefly outlined what Trilinos is and why we have chosen to implement these methods in its Anasazi package. Then, we reviewed the basic structure of Anasazi and the valid parameters for our solvers. The full-length manual also reveals how to build and run these solvers, and it explains how our example drivers work as well as how to modify them to solve your own eigenvalue problems.

REFERENCES

[1] A. Klinvex, F. Saied, and A. Sameh, *Parallel implementations of the trace minimization scheme TraceMIN for the sparse symmetric eigenvalue problem*, Computers & Mathematics with Applications, 65 (2013), pp. 460 – 468. Efficient Numerical Methods for Scientific Applications.

[2] M. F. Murphy, G. H. Golub, and A. J. Wathen, *A note on preconditioning for indefinite linear systems*, SIAM J. Sci. Comput., 21 (1999), pp. 1969–1972.

[3] E. Romero and J. Roman, *A parallel implementation of the trace minimization eigensolver*, in High Performance Computing for Computational Science - VECPAR 2008, J. Palma, P. Amestoy, M. Dayd, M. Mattoso, and J. Lopes, eds., vol. 5336 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2008, pp. 255–268.

[4] A. Sameh and Z. Tong, *The trace minimization method for the symmetric generalized eigenvalue problem*, Journal of Computational and Applied Mathematics, 123 (2000), pp. 155–175.

[5] A. Sameh and J. Wisniewski, *A trace minimization algorithm for the generalized eigenvalue problem*, SIAM Journal on Numerical Analysis, 19 (1982), pp. 1243–1259.

[6] G. L. G. Sleijpen and H. A. V. der Vorst, *A Jacobi-Davidson iterative method for linear eigenvalue problems*, SIAM Journal on Matrix Analysis and Applications, 17 (1996), pp. 401–425.

# TENSOR TOOLBOX: WRAPPING TO PYTHON USING SWIG

M.G. PETERSON * AND D.M. DUNLAVY †

**Abstract.** Tensor Toolbox was developed as a Matlab package to allow users to interact with tensors (ie. multidimensional or $N$-way arrays). A subset of this Matlab version was implemented in C++ with the focus on increased speed for computing. This paper discusses how a Python version was implemented using SWIG (a C++ wrapping tool) on the C++ subset of Tensor Toolbox. All three of the versions were compared against each other based on their computing time for the CPAPR (CANDECOMP/PARAFAC Alternating Poisson Regression) algorithm. The C++ version was by far the fastest, the Python version was the second, and the Matlab version third.

**1. Introduction.** Tensors are multidimensional arrays (i.e., $N$-way arrays) and are often used the fields of machine learning, chemometrics, signal processing, graph analysis, and more (see [5] for a survey for more details). Several different implementations of tensor data structures, models, and model fitting algorithms have been developed for data analysis in these fields: e.g., Matlab Tensor Toolbox (TTB) [2], N-Way Toolbox [1], among others. To date, much of the algorithmic work for tensors applied to data analysis has been implemented in Matlab. We present here an implementation of the operations and algorithms for tensor analysis in Python, following the main class structure of TTB and leveraging a C++ version of the data structures and algorithms available in TTB. We discuss some of the challenges in developing a Python version of a limited subset of functionality of TTB, which we call PyTTB. Further, we demonstrate some of the benefits of using PyTTB over TTB in terms of computational speed.

The CANDECOMP/PARAFAC (CP) tensor model can be used to approximate the relationships amongst data represented in a tensor. The CP model is often referred to as a high-order analog of principal components analysis (PCA) without orthogonality constraints on the factors. Like PCA, the CP model is a sum of vector outer products and can be used to identify a reduced-dimension approximation of data relationships. Algorithms for fitting CP models to data include CPALS (CP Alternating Least Squares) for continuous data and CPAPR (CP Alternating Poisson Regression) for count data. These two models have been implemented in the C++ version of TTB and available in PyTTB. CPAPR will be the method used for comparisons of the different implementations of TTB presented in Section 4.

The Matlab Tensor Toolbox is a software package used for constructing tensor objects and models, computing tensor factorizations, and performing data analysis of multiway data using those structures and models. One of the main strengths of TTB is its efficient computations and algorithms for large, sparse data. TTB was originally written in Matlab, and later a subset of the functions were implemented in C++ with the goal of improving speeds for larger tensors. Challenges is using these two implementations include the cost of a Matlab license for TTB and the lack of an interactive environment for the more efficient C++ implementation. PyTTB is an effort to address these two challenges.

PyTTB provides both an interactive environment for "on-the-fly" object modifications via the Python interactive shell and computational efficiency via binding to the C++ implementation mentioned above. Python is free to use and is a very common language used for interactive purposes. Also, including C++ code in Python

*University of New Mexico for Computer Science, mpeterson@unm.edu
†Sandia National Laboratories, dmdunla@sandia.gov

via extensions can be accomplished without requiring changes to the original C++ code using a software tool called SWIG.

**2. Extending C++ Code Using SWIG.** SWIG (Simplified Wrapper and Interface Generator)[1] is a software tool that wraps C++ classes and functions into other languages such as Perl, Java, Python, TCL, and more [3, 4]. Python wrapping using SWIG is the main focus in the sections below.

The SWIG files created by the developer are referred to as interface files. These files are where the developer can make use of the wrapping features that SWIG offers. The most important features available in SWIG and used in PyTTB are *extension blocks*, *python code blocks*, and *typemaps*.

**2.1. Extension Blocks.** Extension blocks allow the developer to add methods to a class without having to modify the original C++ code. This is useful when you have a method that only applies to the specific language you are trying to wrap to and not in the general use cases of the C++ code, such as mapping a custom object's `size()` method to `__len__` in Python.

Extensions are written in C++ and their scope are functions/methods defined in the original C++ code. This means that methods defined SWIG interface files are outside the scope of methods defined in extensions; even if two given methods are inside the same extension block. Despite the scoping constraint extensions prove useful, especially with adding target language built-in functions, such as `__str__` and `__len__` in Python.

**2.2. Python Code Block.** SWIG interface files are written in C++, but with the
`pythoncode` tag the user can write Python code which will be directly inserted into the generated `.py` package. This can be used for adding additional functions to the package or can be written within an extension block to add more methods to a class.

Unlike extension blocks, the scope of pythoncode blocks is the entire Python package. This allows the developer to use methods and functions previously defined elsewhere in the SWIG interface files, including those in extension blocks.

**2.3. Typemaps.** Typemaps are arguably the most convenient feature of SWIG. They are used for converting an object of one type into that of another type, either on input or output. This provides a simple, global way to convert an object from the target language into a C++ object, or vice versa.

For every typemap(in) (ie. a typemap from target language to C++), there needs to be a special typemap defined called a typecheck; this function returns a boolean value. A typecheck is used to determine whether an object can use a given typemap; it returns true if the input object can be converted or false if it cannot be converted.

An example of the impact of typemaps in SWIG on Python wrapping of C++ code is presented in Figure 2.1, where a package is imported and an array object is printed. A typecheck for the typemap used in `MyPackage.printArray()` would check if the input object is a sequence, only contained three items, and that the items contained inside the sequence were all integers. The typecheck would return false if any of those attributes were not true for the input object, otherwise it would run the conversion typemap.

There is a precedence value associated with each typecheck. This value is responsible for creating the order in which a typecheck gets called. The developer can

[1] `http://www.swig.org/`

EXAMPLE_1: (without a typemap)

```
>>> import MyPackage
>>> v = MyPackage.ThreeArray(1,2,3)  # Creates a ThreeArray.
>>> MyPackage.printArray(v)
Printing Array: [1,2,3]
```

EXAMPLE_2: (using a typemap)

```
>>> import MyPackage
>>> MyPackage.printArray([1,2,3])    # Using a Python list.
Printing Array: [1,2,3]
```

Fig. 2.1: MyPackage is a package that was created using SWIG; it contains a class object called ThreeArray which is a container object that holds three integers. MyPackage also has a function called printArray that takes in a ThreeArray as a parameter. EXAMPLE_1 shows how printArray would be used without a typemap. EXAMPLE_2 shows how it could work if a typemap from a Python list to ThreeArray was previously defined.

modify the precedence value of the typecheck that he/she has constructed.

**3. Python Tensor Toolbox.** PyTTB is the Python version of the C++ implementation of Tensor Toolbox and is intended to provide the same capabilities as the Matlab version, TTB. SWIG was used to map all the C++ functions and classes into Python (see Section 2 above). This section provides specific examples of how PyTTB utilizes features provided by SWIG.

**3.1. Class Wrapping.** Within the PyTTB package there are several C++ classes that were wrapped; `Array`, `IndxArray`, `FacMatrix`, `FacMatArray`, `Tensor`, `Sptensor`, and
`Ktensor`. Some of the classes are used primarily as an underlying data structure for the tensor classes. Each class has their own SWIG interface file containing the mapping rules from C++ for the given class.

**3.1.1. Operator and Method Extensions.** All of the classes implemented in PyTTB include an extension block. Within these extension blocks are arithmetic operators and special object methods for that particular class; such as `__add__`, `__sub__`, `__mul__`, `__div__`, `__pow__`, `__eq__`, `__setitem__`, `__getitem__`, `__iter__`, `__str__`, and `__len__`.

Below is an example of calling the special object methods `__add__` (using '+') and `__str__` (using `print`) for `Array`.

```
>>> import tensor_toolbox as ttb
>>> a = ttb.Array([2,1,3])
>>> b = ttb.Array([1,1,1])
>>> print a + b              #Elementwise addition
[3 2 4]
```

**3.1.2. Error Handling.** Within each class there is an exception block which contains a `try/catch` block. This is used for converting C++ errors into Python errors. One thing to note is that if an error is not explicitly thrown within a C++ method it will not be caught by the exception block.

This what an exception would look like:

```
>>> a = ttb.Array([2,3,1])
>>> b = ttb.Array([2,2])
>>> a + b
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/.../python/tensor_toolbox.py", line 332,
  in __add__
    def __add__(self, *args):
    return _tensor_toolbox.Array___add__(self, *args)
StandardError: Sizes of Arrays do not match
```

**3.1.3. Merge Types.** These were functions added to the PyTTB package that are not included in the C++ version. This will merge two objects of the same class into one. For example, with an Array you could think of a merge as a concatenation function. With an Object like a FacMatrix we can merge as rows or columns; in other words it adds the rows from one FacMatrix to another, or adds the columns. An example can be seen in the listing below.

```
>>> import tensor_toolbox as ttb
>>> f1 = ttb.FacMatrix(2,2)
>>> f1.setValue(1)            # 2x2 matrix with all 1's
>>> f2 = ttb.FacMatrix(1,2)
>>> f2.setValue(2)            # 1x2 matrix with all 2's
>>> ttb.mergeAsRows(f1,f2)
_____________________________________
matrix
_____________________________________
Size = [ 3 2 ]
X(0,0) = 1
X(1,0) = 1
X(2,0) = 2
X(0,1) = 1
X(1,1) = 1
X(2,1) = 2
_____________________________________
```

**3.1.4. CPALS and CPAPR.** Although the CPALS and CPAPR algorithms for fitting CP models are provided in the C++ implementation, they are also implemented in PyTTB as well. In Python, they are implemented as an illustration of the use of several C++ classes and methods in a complete algorithm. These implementations are the ones used in experiments discussed in Section 4.

**3.2. Style Choices.** Several style choices were made during the process of developing the PyTTB package. There are other ways to achieve similar results, but these seem to be better in the long run, motivated by the goal to make it simple for developers to add new classes to PyTTB in the future.

**3.2.1. EQUALITY TOLERANCE.** For a few classes there is a method, `isEqual(object,tolerance)`, defined in the original C++ package. This method compares two objects of the same class and determines whether they are equal to each other within some tolerance level. This method seems like a logical choice to map to the `__eq__` operator in Python where the tolerance level would be zero, but there is a small problem. Even if the two objects are exactly the same, the method will return false because the tolerance level is a strict inequality; i.e. since the the comparision result is always greater than or equal to zero, it cannot be strictly less

than zero. So a global variable is defined, `EQUALITY_TOLERANCE`, that is set to `1.0e-8` by default and is used instead of zero for the `__eq__` methods to compensate for the strict inequality issue.

**3.2.2. Python Code Blocks.** The scope of extension blocks are very limited, but the scope of Python code blocks includes the entire package. The underlying data structures in the tensor classes are either an `Array` or a `FacMatrix`. So when defining a mathematical operator for the tensor class, such as `__add__`, a Python code block was used. This gave the develeoper the ability to use the underlying data structure's `__add__` method when defining the tensor's method for `__add__`.

This style was chosen to reduce the amount of error handling that had to be implemented as well as to provide a simple way to modify the code for multiple classes at once.

**3.2.3. Typemaps.** The goal of typemaps were to make the C++ code feel more Pythonic. Typemaps, such as Python list to `Array` and `IndxArray`, were implemented to allow users to easily change parameters on the fly while in the Python interface.

**3.3. IPython Notebooks.** IPython Notebooks[2] are a useful way to design a tutorial for Python code [6]. Notebooks are set up into cells which are classified as Markdown, Python Code, and various Header Fonts. This allows the designer to use a mixture of Python code and HTML when designing a tutorial.

The Python tutorials were designed to look exactly like the TTB tutorials provided in Matlab. The goal of this was to make an easy transition for Matlab users by making the environment as similar as possible.

When the design of the notebook is finished, there are options to convert the notebook to HTML or Python. The HTML version can be open/viewed from a browser and is useful for displaying the tutorial. The Python version will comment out everything except for Python Code cells, which is useful when a user wants to interact with the code directly.

**4. Experiments.** In this section, we present the results of comparing Matlab, C++, and Python implementations of the CPAPR (CP Alternating Poisson Regression) algoritm . Specifically, we focus on the speed of the implementations (from the user point of view), as we have verified that the 3 implementations generate identical iterations and solutions. The data used for testing were random sparse tensors containing approximately 0.1% nonzeros. The rank of the tensors, 10, was the same for all the tests. The dimensions of the tensors were all of size three, where the first two dimensions remained constant but the third varied. The complete set test characteristics can be found in Fig 4.1.

The second data set varied in all three dimensions. For each of the tests the dimensions were scaled by a factor of 2. More information about the data set is described in Fig 4.2.

Timing information were collected using the `time` Linux system call. The time recorded included the time to run the CPAPR algorithm as well as the time it took to read-in the sparse tensor data and the k-tensor initial guess. Comparisons are made using a paired sample t-test of timing information from the results of running CPAPR on ten different initial guesses for each test size.

The experiments were performed on system with 8 Intel i7-3940XM CPUs (3.00 GHz) with 32GB RAM running Ubuntu 12.04 (64-bit).

[2] `http://ipython.org/notebook.html`

| Test Number | Dimensions | Rank | Nonzeroes | Coefficients |
|---|---|---|---|---|
| 1 | [20,30,15] | 10 | 836 | 650 |
| 2 | [20,30,50] | 10 | 2,836 | 1,000 |
| 3 | [20,30,100] | 10 | 5,684 | 1,500 |
| 4 | [20,30,500] | 10 | 28,371 | 5,500 |
| 5 | [20,30,1000] | 10 | 56,569 | 10,500 |
| 6 | [20,30,5000] | 10 | 282,885 | 50,500 |

Fig. 4.1: A brief description of the attributes for each the test data files used to check performance on each of the languages. In this first data set only the third dimension was changed.

| Test Number | Dimensions | Rank | Nonzeroes | Coefficients |
|---|---|---|---|---|
| 1 | [20,30,15] | 10 | 836 | 650 |
| 2 | [40,60,30] | 10 | 6,808 | 1,300 |
| 3 | [80,120,60] | 10 | 54,476 | 2,600 |
| 4 | [160,240,120] | 10 | 434,638 | 5,200 |

Fig. 4.2: A brief description of the attributes for each the test data files used to check performance on each of the languages. In this second data set all dimensions were scaled by a factor of 2.

**4.1. Python vs Matlab.** Figs. 4.3–4.6 present the results of running CPAPR for the test cases described above. These results demonstrate that the Python implementation is much faster than the Matlab version. Also, this conclusion is confirmed by the results of the paired sample t-test results presented in Fig. 4.7 and Fig. 4.8. When Python and Matlab are compared it can be seen that the t-statistic is consistently negative and decreasing. This implies that Matlab computation time is increasing faster than that of the Python implementation as the problem size increases.

| Test | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| C++ | 0.005 | 0.005 | 0.005 | 0.005 | 0.004 | 0.015 |
| Python | 3.042 | 15.098 | 32.823 | 179.425 | 354.016 | 969.710 |
| Matlab | 16.162 | 56.926 | 92.586 | 314.409 | 577.373 | 2,750.293 |

Fig. 4.3: Average timing results (in seconds) of 10 runs of CPAPR on each of the test cases described in Fig. 4.1.

**4.2. Python vs C++.** The C++ implementation was clearly the fastest of the three versions. In Fig. 4.7 one can see that the t-statistic for Python vs C++ is positive and increasing, which implies that the Python version's computation time is increasing at a faster rate than the C++ version. Another item to note is that the t-statistics for Python vs C++ is much larger than that of Matlab vs C++. This is an indication that the Python version's computation time is growing faster than the Matlab's version in respect to the C++ version.

| Test | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| C++ | 0.004 | 0.004 | 0.005 | 0.005 |
| Python | 3.043 | 41.536 | 204.732 | 1,235.935 |
| Matlab | 16.318 | 132.133 | 609.617 | 4,729.493 |

Fig. 4.4: Average timing results (in seconds) of 10 runs of CPAPR on each of the test cases described in Fig. 4.2.

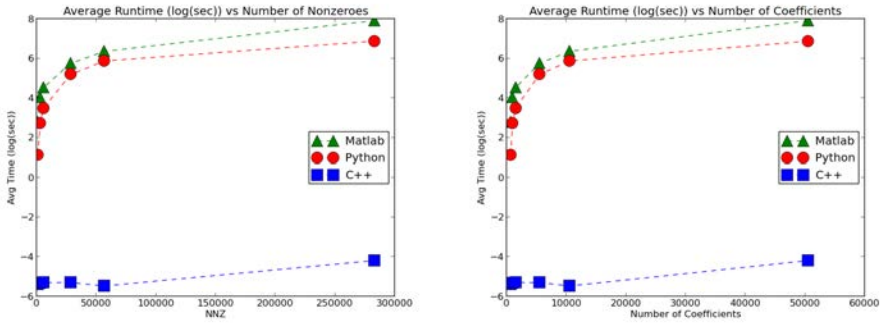



Fig. 4.5: Average timing results (in seconds on a log scale) of 10 runs of CPAPR on each of the test cases described in Fig. 4.1. The x-axes include the number of nozeros, NNZ (left) and number of coefficients in the CPAPR models (right).

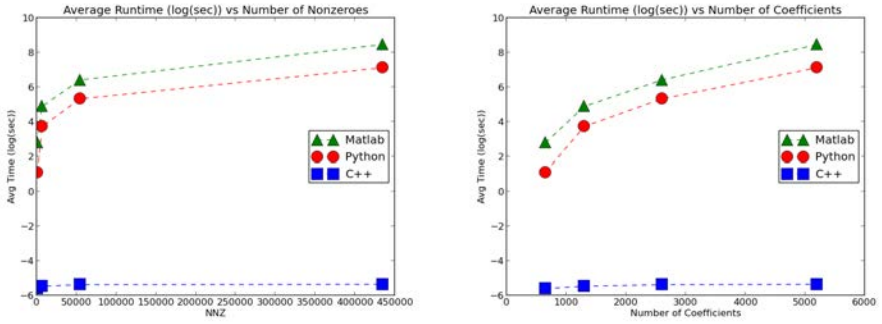



Fig. 4.6: Average timing results (in seconds on a log scale) of 10 runs of CPAPR on each of the test cases described in Fig. 4.2. The x-axes include the number of nozeros, NNZ (left) and number of coefficients in the CPAPR models (right).

| | Python vs Matlab | | Python vs C++ | | Matlab vs C++ | |
|---|---|---|---|---|---|---|
| Test | t-stat | p-value | t-stat | p-value | t-stat | p-value |
| 1 | -4.93 | 1.08e-04 | 5.56 | 2.81e-05 | 6.20 | 7.44e-06 |
| 2 | -5.35 | 4.41e-05 | 7.46 | 6.57e-07 | 7.53 | 5.71e-07 |
| 3 | -7.12 | 1.24e-06 | 11.68 | 7.77e-10 | 11.70 | 7.55e-10 |
| 4 | -9.25 | 2.92e-08 | 47.48 | 2.29e-20 | 22.31 | 1.45e-14 |
| 5 | -12.44 | 2.81e-10 | 80.04 | 1.98e-24 | 36.04 | 3.11e-18 |
| 6 | -28.63 | 1.83e-16 | 206.78 | 7.72e-32 | 44.70 | 6.71e-20 |

Fig. 4.7: Above is a short summary of statsistcal values of the experiments described in Fig. 4.1. A negative t-value indicates the first faster than the second, and a positive t-value implies the first is slower than the second. The p-value is the probability that one would be correct if they reject the hypothesis in regards to the t-value

| | Python vs Matlab | | Python vs C++ | | Matlab vs C++ | |
|---|---|---|---|---|---|---|
| Test | t-stat | p-value | t-stat | p-value | t-stat | p-value |
| 1 | -5.12 | 7.18e-05 | 5.78 | 1.74e-05 | 6.42 | 4.73e-06 |
| 2 | -17.45 | 9.92e-13 | 27.82 | 3.02e-16 | 26.58 | 6.74e-16 |
| 3 | -166.05 | 3.97e-30 | 239.58 | 5.42e-33 | 266.94 | 7.74e-34 |
| 4 | -296.58 | 1.16e-34 | 716.74 | 1.47e-41 | 405.88 | 4.11e-37 |

Fig. 4.8: Above is a short summary of statsistcal values of the experiments described in Fig. 4.2. For information about the meaning of the values please refer to Fig. 4.7.

**5. Conclusion.** The Python version was not as fast as the C++ version, but it was faster than the Matlab version. The goal of PyTTB was to offer an alternative to the Matlab version, while leveraging the efficiency of the C++ implementation when possible. PyTTB currently offers a subset of the Matlab TTB functionality, and this subset can produce the same results in less time.

The style choices chosen in PyTTB help consolidate shared functionality into reusable code. The use of `pythoncode` tags decreased the amount of error checking needed. It provided a coordinated approach to error checking of previously defined functions containing a diverse set of error checking measures. But the choice that impacted the code the most was typemaps.

Typemaps allowed an easy way to use make PyTTB feel more "Pythonic." The methods and functions defined in the C++ version were able to take in a larger variety of arguments by defining a hand full of mappings. Without the typemaps, a majority of the methods would have required overloaded implementations to handle Python objects (e.g., `lists`).

**5.1. Future Work.** As mentioned in Section 3.1.2, there is room for improvement in regards to error handling. There may be a way to use typemaps to handle errors, avoiding explicitly throwing errors. Another option would be to include error `try/catch` blocks in every class extension block; the typemaps would convert C++ errors into python errors, even if the C++ errors were not explicitly thrown.

Lastly, there is the the fact that C++ version is much faster than the Python version. There are some areas in the Python where optimization may be available, primarily where Python handles mathematical operations. In a few of the mathematical methods, a copy of the object is used instead of using the original. Generating a copy makes sense for generic cases, such as `z = a + b`. But if `a += b` is done instead, a copy does not need to be generated and the `a` object can be modified directly. Another option would be to run CProfile to help identify areas in the code that could lead to timing improvements.

REFERENCES

[1] C. A. Andersson and R. Bro, *The N-way toolbox for MATLAB*, Chemometrics & Intelligent Laboratory Systems, 52 (2000), pp. 1–4.
[2] B. W. Bader, T. G. Kolda, et al., *Matlab tensor toolbox version 2.5.* Available online, January 2012.
[3] D. M. Beazley, *Automated scientific software scripting with SWIG*, Future Gener. Comput. Syst., 19 (2003), pp. 599–609.
[4] T. Cottom, *Using SWIG to bind C++ to python*, Computing in Science Engineering, 5 (2003), pp. 88–97.
[5] T. Kolda and B. Bader, *Tensor decompositions and applications*, SIAM Review, 51 (2009), pp. 455–500.
[6] F. Pérez and B. E. Granger, *IPython: a system for interactive scientific computing*, Computing in Science and Engineering, 9 (2007), pp. 21–29.

# IMPLEMENTING PARALLEL ALGORITHMS USING THE DAX TOOLKIT

HENDRIK SCHROOTS[‡] AND KENNETH MORELAND[§]

**Abstract.** With processor speeds no longer increasing, improving computing peformance has been more difficult. Parallel processing offers an avenue though which performance advances can be made, especially in this big data era. However, experts agree that to reach exascale computing even greater amounts of concurrency must be exploited from current and future systems. The Dax Toolkit provides a framework that focuses on exploiting that concurrency for data anlysis and visualization. In this paper we implement two parallel algorithms using Dax. A parallel merge sort shows improved perfromance compared to other sort algorithms, and the implementation of marching tetrahedra provides Dax with the added functionality of handling unstructured grid data for countour surface generation.

**1. Introduction.** As Moore's law continues to apply, computers should be getting faster. However, there has not been a major increase in processor speeds once they reached around 3.2 Ghz back in 2006. The additional transistors that continue to double every eighteen to twenty-four months as per the law have gone into additional cores on a single die. This shift in processor design has encouraged the long held notion that programmers must approach solutions from a parallel aspect if they are to achieve peak performance.

The push for parallel solutions and parallel processors started long before processor speeds stopped increasing and long before multiple cores became commercially available. Seymour Cray did groundbreaking work in the realm of supercomputing back in 1976 with the release of the CRAY-I. This system demonstrated the application of vector processing, which is a concept fundamental to modern parallel processors. Today, a leader in both supercomputing and modern parallel architectures is Nvidia. Their Kepler architecture takes a SIMD approach to processing and thus achieves high throughput. This type of processor is commonly known as a graphics processing unit (GPU). It took some time for GPUs to become viable supercomputing processors since despite their high throughput their overall latency was just too far behind CPUs at the time. However, advances have pushed GPU latency into the realm of practical use while making strides in throughput as well. This dichotomy of low latency processors (CPU) and high throughput processors (GPU) has led to what is known as heterogeneous computing.

Heterogeneous computing is the term used to describe a system that is made up of multiple processors that have different architectures. The advantage of such a system is greater overall performance than that of just using one processor architecture or the other in isolation. However, one of the major drawbacks to such a system is the detailed care that must be taken to have a single program run across multiple architectures. In addition, to achieve peak performance a deep understanding of the underlying hardware for each architecture is required. This creates a divergence of goals for the programmer. Where the goal of programming is often to create software that is general and robust, heterogeneous computing demands software to be written specific to a given architecture's hardware design. One such example is Nvidias CUDA language, which until recently required code to explicitly allocate and transfer memory between the GPU and CPU memory systems. The Dax toolkit, on the other hand,

[‡]University of California, Davis, hschroots@ucdavis.edu
[§]Sandia National Laboratories, kmorel@sandia.gov

is a new framework that takes care of many of the hardware specific details allowing the programmer to focus on writing robust code that can then be run on multiple architectures. Dax targets data analysis and visualization on exascale platforms where large amounts of parallelism are needed in order to achieve peak performance.

As mentioned above, supercomputers today make use of every technique available to achieve maximum computational power. This includes heterogeneous systems, no matter how much detail must be known about the system in order to be programmed. Dax makes developing visualization applications for such large scale computers simpler by abstracting away many of the hardware specific details into an object type named the Device Adapter. The programmer can then develop programs within the constructs of the framework, and Dax will take care to manage the data such that it runs correctly on the device specified.

This paper focuses on extending and testing the usability of the Dax framework. First, a parallel version of the merge sort was implemented in Dax, and its run-time compared to the built-in sort function. As a fundamental building block to more complex algorithms, adding a faster sort algorithm improves Dax's performance. The contouring algorithm marching tetrahedra was also implemented and its performance compared to the marching cubes algorithm. Also a common visualization algorithm, marching tetrahedra adds to Dax's robustness as a framework by being capable of handling an additional type of input data for contouring.

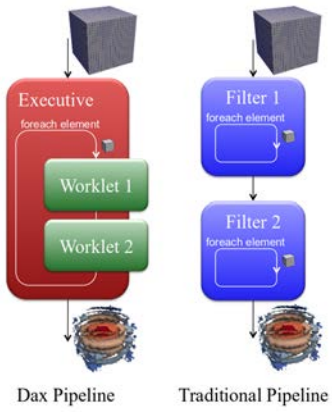



Fig. 2.1: The traditional pipeline must process the entire data set before applying the next filter. The Dax pipeline can execute all worklets in the order specified because data elements are independent.

[8]

**2. Related Work.** As previously stated, the Dax toolkit is for data analysis and visualization on exascale systems. Many visualization applications use what is known as the visualization pipeline to structure a logical flow of data through the system. The visualization pipeline works by modifying the dataset via filters where the output of one filter is the input to the next. While many visualization applications have had success with this construct such as the Visualization Toolkit (VTK) and the Application Visualization System (AVS), one drawback is that between filters the pipeline has to synchronize. Parallelism is abundant within each filter, but the next

filter cannot begin modifying the data until all the threads of the previous filter have completed.

Dax makes use of transformation functions similar to a filter called a worklet. A worklet is a computational unit that works on a single independent data element. Because the processing of data elements is independent, a data element is free to flow through a pipeline of worklets as soon as it is ready and doesnt have the need to synchronize between worklets. Figure 2.1 shows a comparison of the traditional pipeline and a pipeline as implemented using worklets. This approach allows for much more concurrency in a system.

Exascale computing requires larger amounts of concurrency to be exploited [8, 7]. Amdahl's law states that the amount of speed up from parallelizing a program is limited by the amount of work that cannot be parallelized. In other words, given an infinite number of resources a program simply cannot run faster than its slowest non-parallel part [6]. This puts an upper bound on how much speed up parallel processing can provide and in some cases that speed up is not very promising. This observation, however, assumes that the portion of work that is parallel remains constant.

Gustafson-Barsis' law provides a corollary to Amdahl's law. It states that programs can in fact achieve large amounts of speed up as the number of processors increases so long as the amount of parallel work grows with the scaling of processors. With the big data revolution we are in right now there is no doubt that the amount of data we are processing is in fact growing at an astounding rate. This is where Dax steps in.

The Dax toolkit attempts to exploit the necessary parallelism needed for exascale computing. Table 2.1 shows the projected level of concurrency needed for exascale performance. Experts agree that to achieve this amount of concurrency even finer levels of parallelism must be exploited. Table 2.2 shows these levels and in what capacity they are used. The very-fine level parallelism is handled by compilers to a certain degree and ultimately by the underlying hardware to exploit instruction level parallelism. The Dax worklet provides the mechanism necessary to process data at the fine-grain level throughout all data transformations.

Table 2.1: Comparison of characteristics between petascale and projected exascale machines.

| | Jaguar – XT5 | Titan – XK7 | Exascale | Increase |
|---|---|---|---|---|
| Cores | 224,256 | 247,696 | 100 million – 1 billion | 400 – 5,000× |
| Threads | 224,256 way | 70 – 500 million way | 1 – 10 billion way | 4,000 – 50,000× |
| Storage | 300 Terabytes | 13.6 Petabytes | 10 – 128 Petabytes | 30 – 500× |
| Storage Bandwidth | 240 GB/s | 240 GB/s | 2.5 – 5TB/s | 6 – 25× |

Table 2.2: Levels of Parallelism

| Grain Size | Level Name | Implemented at | Parallelized by |
|---|---|---|---|
| Large | Task-Level | Program | Programmer |
| Medium | Control-Lelvel | Function | Programmer |
| Fine | Data-Level | Loop/Instruction block | Programmer/Compiler |
| Very fine | Instruction-Level | Instruction | Processor |

**3. Algorithms.** The first step was to implement parallel merge sort in Dax. Traditionally, the merge sort algorithm is a divide and conquer algorithm. Developed by John Von Neumann in 1945, the divide and conquer approach is inherently parallel. The array of elements to be sorted is recursively divided in two, until at the lowest level there are only sub arrays with one element in them. Each thread then takes a pair of sub arrays (each of which is guaranteed to be sorted) compares the elements of the arrays and merges the two into a sorted array. Figure 3.1 shows a top down approach example of this. An unsorted array at the top is split in half until each thread merges the sub arrays back into a single sorted array.

For a large number of elements one can see that there is plenty of work to do for many threads at the lowest level of subdivision. However, once near the top there is only a single thread in charge of a very big task, which is to compare the elements of the two sorted sub arrays into a single array. The solution for maintaining large amounts of concurrency is a simple binary lookup. Instead of having each thread merge an entire subarray, a thread is spawned for each element in the array. The threads calculate their scatter address by adding their current index location to the number of elements less than its value found in the other array. Figure 3.2 shows this calculation. The search into the other array is a binary search. Now the level of concurrency is maintained throughout the entire algorithm.

The traditional merge sort implementation has a time complexity of $O(n \log n)$. However, for the parallel implementation each element incurs the cost of a binary search which runs in $O(\log n)$. This makes the parallel version run in $O(n \log^2 n)$ time. It is slightly slower, but because the concurrency is maintained the cost is amortized on highly parallel hardware.

The other algorithm implemented in Dax for this paper is marching tetrahedra (MT). Marching tetrahedra is a derivative of the algorithm marching cubes (MC) originally conceived by William Lorensen and Harvey Cline in 1987 [5]. Marching cubes falls into a classification of algorithms that deal with extracting a contour from a field of scalar values. Most people are familiar with the 2D analog of MC, called marching squares. Marching squares generates isolines commonly used in elevation maps.

MC takes regular grid data as input. Because regular grid data is defined at equally spaced intervals, the structure of the data is implicit and the data can be stored as a simple list of vertices. Contouring algorithms extract the surfaces in 3D defined at a given isovalue. Lorensen made the case that just like there are a finite number of ways a line can pass through a square, there are a finite number of ways a curve passes through a cube [5]. To do this each vertex of each cube undergoes a binary classification. Each vertex is first identified as being less than (inside), or greater than or equal to the isovalue (on or outside) the specified isolvalue. This

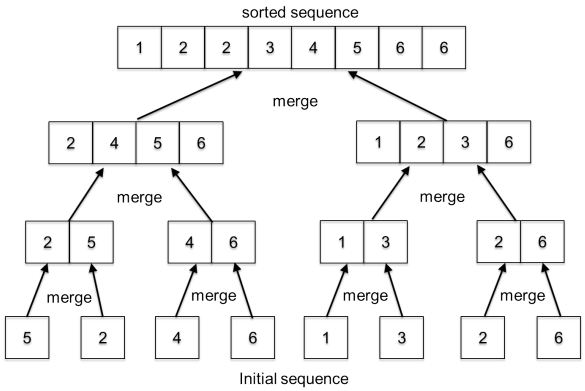



Fig. 3.1: Each thread is responsible for the merging of two subarrays in a traditional merge sort.

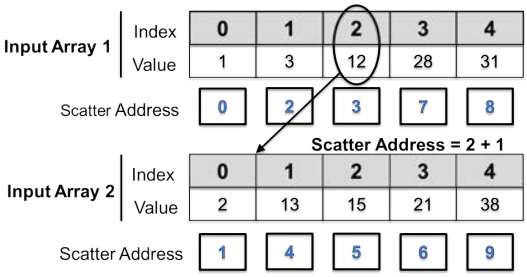



Fig. 3.2: The calculation of the scatter address in a parallel implementation of merge sort.

binary classification creates a finite number of cases for each cube. Since each cube has eight vertices, and each vertex can be either inside or outside of the curve defined at the isovalue, there are $2^8 = 256$ possible ways a curve passes through a cube. Using symmetry these 256 cases are reduced to just 14 unique cases.

It was later found that certain cases are ambiguous and that there are multiple ways a curve passes through these particular cube configurations [11]. Much work has gone into resolving these ambiguities [11, 10, 3, 9]. By adding 44 cases to the original 256 cases these ambiguities are identified and handled to produce a topologically continuous surface [3]. Figure 3.3 shows some of the ambiguous cases and the extra cases that are used to resolve them [11].

Marching tetrahedra takes the same approach MC does, only it uses tetrahedra, not hexahedra, as its cell shape. Using tetrahedra provides one primary benefit over hexahedra. Because each face of the tetrahedron is a triangle, there are no ambiguous cases when identifying how a curve passes through the cell. Also, because each tetrahedron only has four vertices, instead of the eight in hexahedra, there are only $2^4 = 16$ cases to worry about. Figure 3.4 shows the full classification of tetrahedra for this algorithm. This simplification does not come without its draw back though. Since tetrahedra are not a regular shape, their sampling cannot be implicit. Extra data

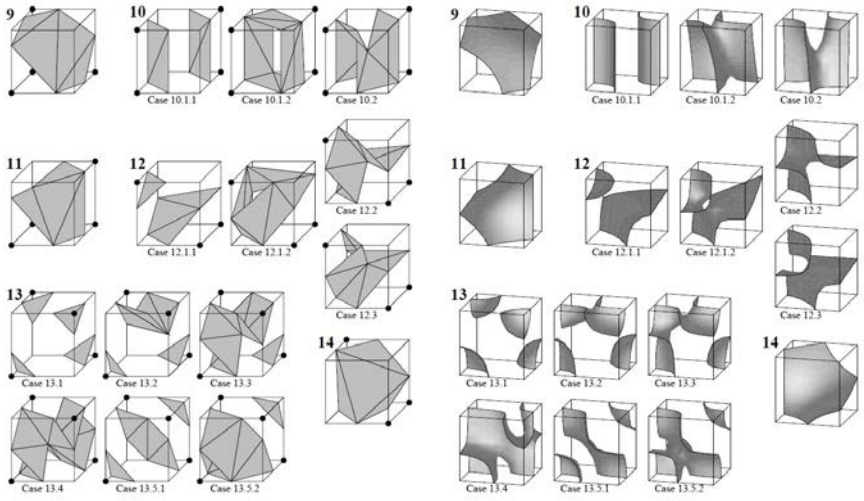



Fig. 3.3: Part of an advanced lookup table to resolve ambiguities in Marching Cubes. [3]

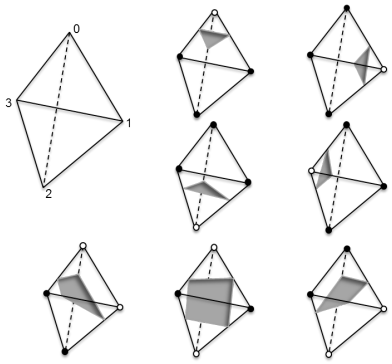



Fig. 3.4: The binary calssification of tetrahedra for Marching Tetrahedra.

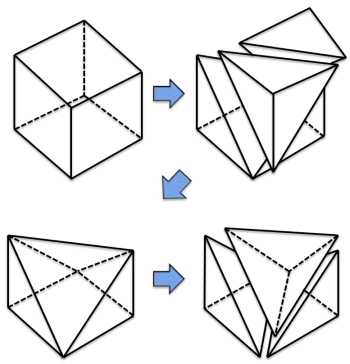


Fig. 3.5: A 5-fold subdivision of a hexahedron into tetrahedra.

must be maintained to describe the structure of the tetrahedra [12]. Another point of motivation for using MT over MC is that not all data is regularly sampled. In fact, a large part of volume rendering deals with unstructured grid data and ultimately many other primary cell shapes can be broken down into tetrahedra [4].

An important step in the MT algorithm is to subdivide the hexahedral cells into tetrahedral cells. Hamish Carr details a complete taxonomy of how to subdivide hexahedra into tetrahedra [2, 1]. This paper uses a 5-fold minimal subdivision shown in Figure 3.5. This not only reduces the amount of processing (compared to subdivisions of 6, 12, 24, or 48 tetrahedra) but maintains the original data since each tetrahedron is defined using the original vertices from the hexahedron. In other words, no new vertices need to be added in order to define the tetrahedral subdivision. It should be noted that a 6-fold Freudenthal subdivision also only uses the original vertices and

would have served the same purpose as the minimal subdivision. However, testing different subdivisions is outside the scope of this paper.

**4. Methods.** The first experiment is designed to test the run-time of the implemented parallel merge sort and compare it the bitonic sort algorithm that is included with Dax. Their run-times were also compared to the radix sort algorithm built into the Thrust libraries. Dax uses Thrust for reasons other than sorting, and it provides a simple point of relative performance with which to compare. Thrust's radix sort is labeled as "Native" in the graph. At the time of the experiment bitonic sort only worked on arrays whose lengths were a power of 2. However, this is an implementation issue. There is nothing about the bitonic sort algorithm that restricts it to power of two length arrays.

Three parameters were controlled to conduct this experiment. First, The values in the arrays were randomly generated integers from 0 to the maximum integer value. Next, the length of the array for bitonic sort started at $2^{14}$ and doubled each iteration for 13 iterations. The maximum array length for bitonic sort was $2^{26}$. For the parallel merge sort and the native sort their array sizes started at $2^{14}$ plus a randomly generated integer offset between 0 and 20,000. For each iteration their array sizes were doubled until their sizes exceeded 73 million elements individually. Last, each sort algorithm was executed 1000 times.

The second experiment is designed to test the run-time of the MT algorithm and compare it to the MC algorithm. The data for this experiment is a simple uniform scalar field whose dimensions are passed in as a parameter. The value at each point in the field is simply its distance from the origin. To this effect, both the algorithms were run on multiple input sizes. Starting with a uniform grid of size $64^3$, the size of the grid is incremented by 8 units each iteration. The final size of the grid is $104^3$. At each input size, the surface extracted was defined at the isovalue of 100. Data on the number of output cells generated was also measured and compared. That is, the number of triangles generated from each algorithm at each input size was counted.

These experiments were run using an Intel Xeon E5-1620 processor with 4 cores at 2x hyper threading per core. The system has 8GB DDR3 1300Mhz RAM and runs Red Hat Enterprise Linux release 6.5. For the parallel experiments an Nvidia Quadro 600 with 1GB on chip memory and a 128-bit memory interface on a PCI express 2.0 interconnect was used.

**5. Results.** The chart in Figure 5.1 shows the different running times of the three sorting algorithms. The parallel bitonic sort ran the slowest as expected. Data could only be collected for array lengths that are a power of 2 so a cubic trend line is used to interpolate between values for the array sizes. Even though bitonic sort, and parallel merge sort have the same time complexity of $\mathrm{O}(n \log^2 n)$ it is clear that the parallel merge sort has a performance advantage over bitonic sort. For the largest array size of $2^{26}$ elements, parallel merge sort ran about 1.3x faster than bitonic sort. Also as expected Thrust's native sorting algorithm ran about 6x faster than parallel merge sort and about 8x faster than bitonic sort. The performance of the native sort however may not be great if floating point values were sorted instead of integer values. Radix sort requires additional computation to handle float point values whereas both bitonic sort and parallel merge sort are expected to run at speeds similar to the ones shown. However, because additional effort must be taken to handle floating point values for Thrust's radix sort, it would not work well with floating point values. This is where parallel merge sort provides benefit since its runtime is not affected whether its values are integers of floating point values.

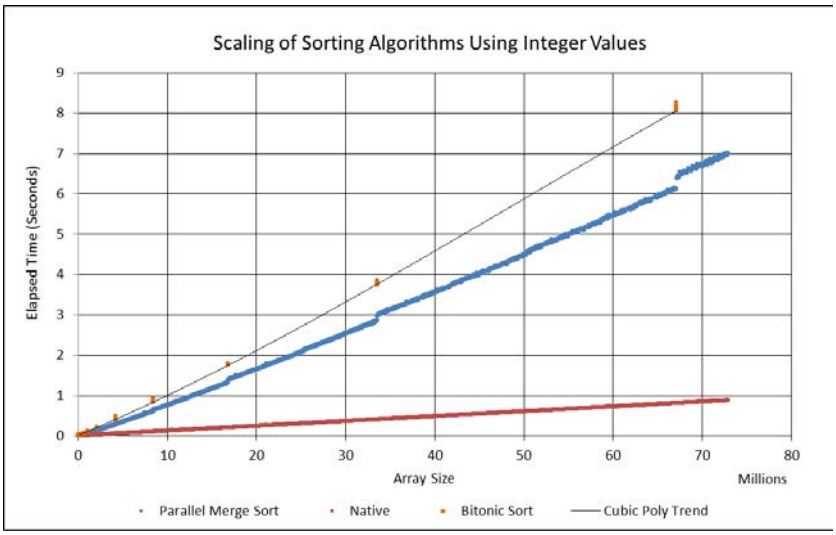



Fig. 5.1: A comparison of the runtimes for different sorting algorithms as the number of elements sorted increases.

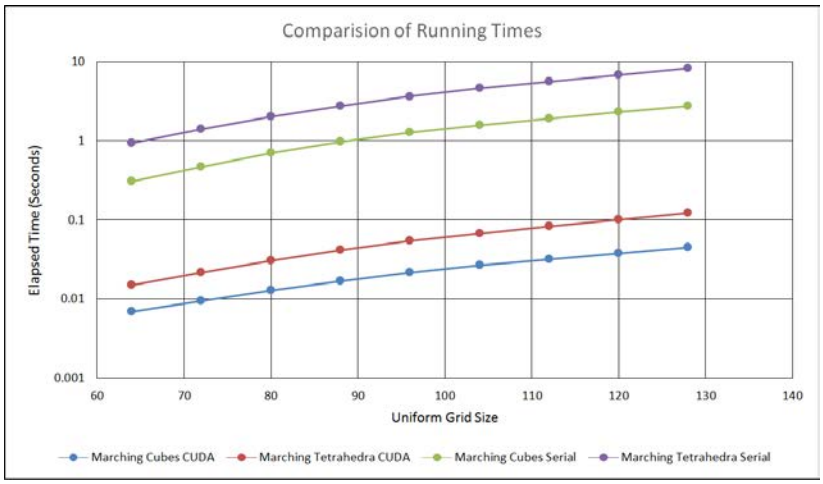



Fig. 5.2: A comparision of the rumtimes of MC and MT run in serial and parallel.

In figure 5.2 we see the runtimes of the MC and the MT algorithms. Each algorithm was run twice for each input size. The first time was run in parallel and the second run in serial. This was done simply to demonstrate the performance speedup that parallel processing has to offer. We see that each algorithm shows about 100x speedup by being run in parallel. The vertical axis is on a logarithmic scale to show better separation of the data for small input problem sizes. Comparing both the serial and the parallel runs between the two algorithms we see that MT closely matches MC in performance but is consistently slower. This is expected, however, because of the 5-fold minimal subdivision generates more output data.

Both algorithms perform two passes over the data. The first pass classifies each cell and identifies how many output triangles will be generated by each cell. For the standard 256 possible cases in MC each cube can generate at most five output triangles. In contrast, each tetrahedron only generates at most two output triangles. However, with each cube decomposed into five tetrahedra there is a potential for 10 output triangles compared to the original 5 from the cube. This increase in output geometry, shown in figure 5.3, explains the difference in runtime. Because MT has a larger output array, more data store instructions are executed. In addition to the decomposition of the cube, both algorithms go through a sort step so that duplicated vertices can be removed. Sorting on a larger output array will also take longer. Lastly, the runtimes of the algorithms in parallel do not include the time to transfer data between the GPU and CPU. Results may not show a 100x increase in performance if data transfer times were taken in to account for both the serial and parallel runs of the algorithms.

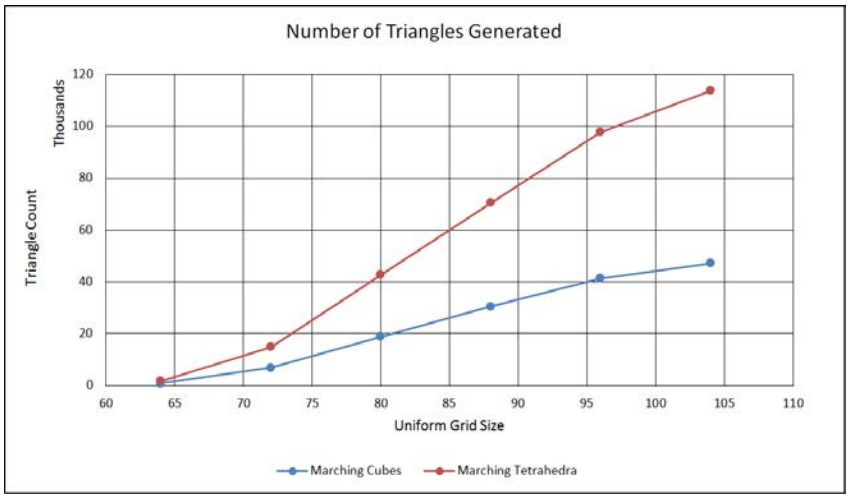



Fig. 5.3: A comparison of the amount of gemoetry generated by MT and MC.

Figure 5.4 shows a comparison of the isosurfaces generated by both MC and MT. There is no visible difference between the two. However, figure 5.5 shows the difference in the geometry between the two algorithms. It is clear that MT has much more geometry generated.

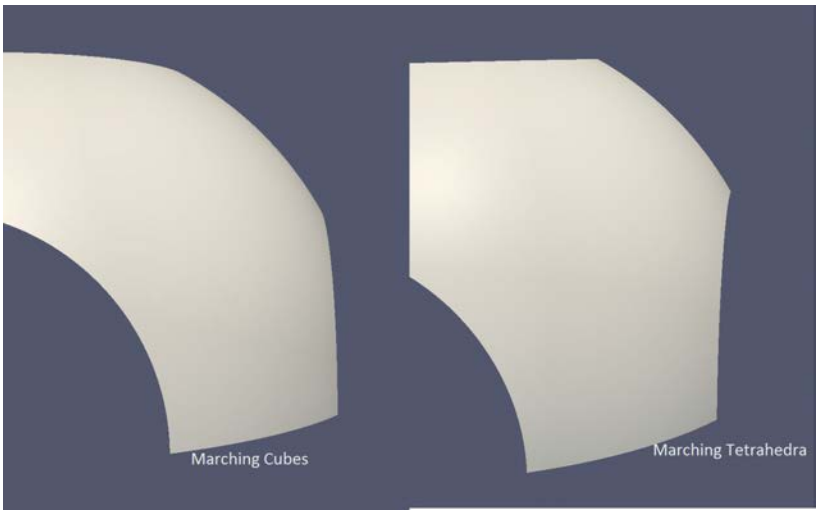



Fig. 5.4: A comparison of the surfaces generated by MC and MT. The surfaces are visibly indistiguishable.

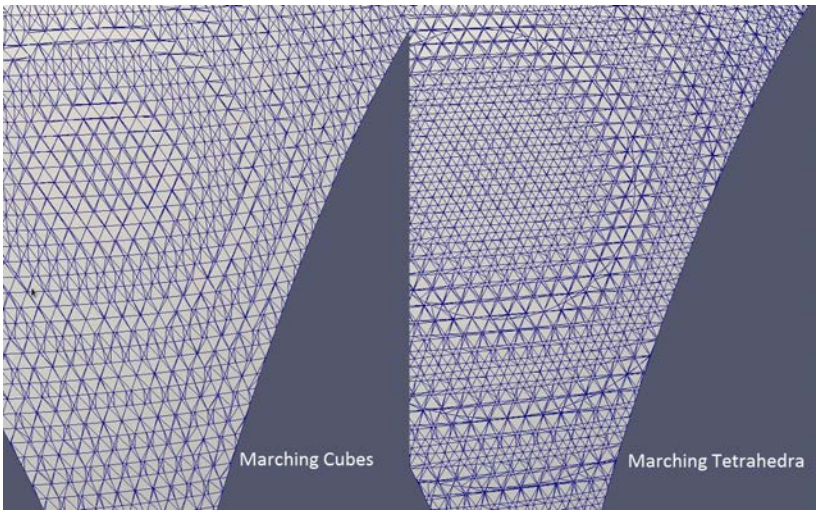



Fig. 5.5: A wireframe image of surfaces generated runnning the MC and MT algorithms.

**6. Conclusions.** This paper implements a parallel version of merge sort and the visualization algorithm marching tetrahedra using the Dax toolkit. A comparison of the runtimes of each of these to related algorithms demonstrates the powerful performance increase realized when parallel processing techniques are used. The Dax toolkit is a framework aimed at data analysis and visualization on exascale systems. As the scientific community pushes the capability of computing today we approach the threshold of exscale computing. The trend so far shows that these systems will be heterogeneous. The ease with which Dax switches between many different architec-

tures makes it a great tool to harness the performance capabilities that heterogeneous systems offer. The implementation of a parallel merge sort helps Dax continue to be a flexible tool capable of high performance. The addition of the MT algorithm adds to Dax's robustness as a tool as it now has a built in algorithm to handle isosurface extraction of unstructured grid data.

There is still work to do in the extension of Dax however. The adoption of it as a visualization tool will depend in part on its ease of use. In addition there many more visualization algorithms and data analysis techniques that can be incorporated into Dax to continually extend its functionality.

## REFERENCES


[1] H. Carr, T. Möller, and J. Snoeyink, *Simplicial subdivisions and sampling artifacts*, in Proceedings of the Conference on Visualization '01, VIS '01, Washington, DC, USA, 2001, IEEE Computer Society, pp. 99–106.

[2] H. Carr, T. Moller, and J. Snoeyink, *Artifacts caused by simplicial subdivision*, IEEE Transactions on Visualization and Computer Graphics, 12 (2006), pp. 231–242.

[3] E. V. Chernyaev, *Marching cubes 33: Construction of topologically correct isosurfaces*, tech. rep., 1995.

[4] J. Dompierre, P. Labbé, M.-G. Vallet, and R. Camarero, *How to subdivide pyramids, prisms, and hexahedra into tetrahedra.*, in IMR, 1999, pp. 195–204.

[5] W. E. Lorensen and H. E. Cline, *Marching cubes: A high resolution 3d surface construction algorithm*, SIGGRAPH Comput. Graph., 21 (1987), pp. 163–169.

[6] M. McCool, J. Reinders, and A. Robison, *Structured Parallel Programming: Patterns for Efficient Computation*, Elsevier Science, 2012.

[7] K. Moreland, *A pervasive parallel framework for visualization: Final report for fwp 10-014707*, tech. rep., January 2014.

[8] K. Moreland, U. Ayachit, B. Geveci, and K.-L. Ma, *Dax toolkit: A proposed framework for data analysis and visualization at extreme scale*, in Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on, Oct 2011, pp. 97–104.

[9] T. S. Newman and H. Yi, *A survey of the marching cubes algorithm*, Computers & Graphics, 30 (2006), pp. 854–879.

[10] G. Nielson, *On marching cubes*, Visualization and Computer Graphics, IEEE Transactions on, 9 (2003), pp. 283–297.

[11] G. M. Nielson and B. Hamann, *The asymptotic decider: Resolving the ambiguity in marching cubes*, in Proceedings of the 2Nd Conference on Visualization '91, VIS '91, Los Alamitos, CA, USA, 1991, IEEE Computer Society Press, pp. 83–91.

[12] W. J. Schroeder, K. Martin, and W. Lorensen, *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics, Third Edition*, Kitware, Inc. (formerly Prentice-Hall), Janurary 2003.

# STRONGLY CONNECTED COMPONENTS ON GPU

GEORGE M. SLOTA*, SIVA RAJAMANICKAM†, AND KAMESH MADDURI‡

**Abstract.** Breadth-first search and color propagation are fundamental subroutines used in many graph-based analytics. With the advent of manycore computational systems such as GPU accelerators, the parallelization of these subroutines is not straightforward, especially on irregular graphs with a skewed degree distributions. The aim of this work is to implement and optimize breadth-first search and color propagation running on GPUs in the context of computing the strongly connected components of large-scale real-world graphs. We utilize the recently developed Kokkos framework to implement known and new algorithms. Our GPU approach shows up to a 1.5× speedup over the present state-of-the-art for CPU computation of strongly-connected components.

**1. Introduction.** The recently developed Kokkos library [8, 7, 4] is an example of a recent paradigm in scientific computing, the development of frameworks for *write-once-run-anywhere* code. Due to the ever-changing nature of computational hardware and large costs associated with updating older code to be performant on modern systems, the need for such frameworks is evident. Ideally, algorithms implemented in such a framework should be resistant to the current trends in hardware (e.g. increasing parallelism and changing memory hierarchies) by exploiting wide parallelization, minimal synchronization, and localized memory accesses whenever possible.

We utilize the Kokkos library in an exploratory fashion, to examine the process of altering existing parallel state-of-the-art multi-core CPU codes to run in a manycore GPU environment. For this, we take the problem of computing strongly connected components (SCCs) in large directed small-world graphs, a common analytic for social networks [15] and a preprocessing step for scientific computing (among other usages) [17]. Using the multicore-optimized Multistep algorithm [18], we demonstrate the process of optimizing the various subroutines utilized by the algorithm for manycore performance.

**1.1. Our Contributions.** Our primary contribution is the implementation of a strongly connected components algorithm that runs up to 1.5× faster than the current state-of-the-art. Additionally, since the primary subroutines comprising our algorithm are breadth-first search (BFS) and color propagation, we demonstrate several routes of optimization for these algorithms on several differing graph topologies. Due to the commonality for both BFS and color propagation, these optimizations are applicable across a wide range of use cases.

**2. Background.**

**2.1. Kokkos Library.** The Kokkos library was originally developed as a backend for linear algebra solvers, but has since been extended to a more general-purpose library for parallel execution. The two primary capabilities of Kokkos include polymorphic multidimensional arrays optimized for varying data access patterns and thread parallel execution that allows for fine-grained data parallelism on manycore devices.

The parallel execution model follows a dispatch model where a single master CPU thread divides some $N$ units of work to be processed in parallel. Each unit of work is executed by a single *thread team*. On GPU, a thread team is comprised of multiple warps each executing on the same multiprocessor. This team of threads operates in

*The Pennsylvania State University, gslota@psu.edu
†Sandia National Laboratories, srajama@sandia.gov
‡The Pennsylvania State University, madduri@cse.psu.edu

a data parallel SIMD fashion, and is able to communicate via shared memory. On CPU, a thread team is comprised of a small number of threads (usually 1) executing on a single CPU core.

**2.2. Parallel Strongly Connected Components Algorithms.** The optimal serial algorithm for detection of strongly connected components is Tarjan's algorithm, which is based on depth-first search (DFS). Due to the lack of parallelism available with a DFS-based approach, several alternative parallel SCC algorithm have been developed based around bread-first search and color propagation.

**2.2.1. Forward-Backward.** The Forward-Backward algorithm (FW-BW) [19] utilizes a recursive approach based on two breadth-first searches from a single *root vertex* to find a single SCC. A BFS from the root following all out-edges will discover all vertices that are reachable from the root. A BFS from the root following in-edges will discover all vertices that are able to reach the root. Because a SCC is defined as the maximal set of vertices that are able to reach and be reached by all other in the set, the union of the two sets of vertices both reachable from the root and able to reach the root forms a single SCC.

Removing this SCC from the graph results in three distinct sets: vertices that were not visited in either search, vertices that were only reachable from the root, and vertices that were not reachable from the root but could reach the root. The SCCs comprising these sets are distinct and able to processed recursively with task-based parallelism. Due to the relatively limited data parallelism available when processing lots of small SCCs and overheads associated with task-based parallelism, this approach tends to suffer on highly parallel systems. Although an efficient tasking model can greatly benefit the performance of the algorithm [10], this approach will still suffer once the number of small nontrivial SCCs gets high enough [18].

**2.2.2. Trimming.** Trimming was originally proposed as a preprocessing step for the Forward-Backward algorithm [13]. It is not a fully independent SCC algorithm on its own since it can only find trivial SCCs. Trimming simply scans over the set of vertices and removes all vertices with a current out-degree or in-degree of zero. It can do this once, recursively, as well as multiple times throughout the running of Forward-Backward or another algorithm. Because of the aforementioned overheads involved with individually processing small SCCs with the forward-backward algorithm, utilizing a trimming procedure can greatly benefit computation time.

**2.2.3. Color Propagation.** The color propagation algorithm [16] follows a similar processing model as FW-BW in that there are both forward and backward phases. During the forward phase, all vertices in the graph are initialized with a unique numeric *color*. All vertices propagate their color to all of their out-neighbors if the vertices' color is greater than that of their neighbors' colors. This continues in an iterative process until no more colors can be propagated.

On the backward phase, we consider all of the unique sets of vertices with the same color. For a single set, we consider the vertex in that set that originally owned the set's color to be the root of a new SCC. From that root, we find all vertices reachable via in-edges that are in the same colored set. This forms an SCC. This stage can be either processed in a task-parallel with individual searches from each root or data parallel by performing what is essentially a constrained multi-source BFS.

Note that a single pass of the forward and backward stages will not remove all SCCs. This requires that color propagation have several iterations. However, due to the fact that multiple SCCs can be removed during each iteration and that there is

considerably higher data-level parallelism available for processing small SCCs, coloring tends to run a lot faster than FW-BW when removing a lot of small but non-trivial SCCs. However, the color propagation portion of the algorithm is highly dependent on the diameter of the graph, and will require a large number of iterations to converge on high diameter graphs.

**2.2.4. Multistep.** For real small-world graphs, it has been noted that there is usually one large central SCC and many smaller ones [3]. Using this observation, the Multistep algorithm was developed [18]. This algorithm runs a single iteration of trimming followed by a single iteration of FW-BW to ideally remove the large SCC in the graph. It then runs coloring iteratively until some minimal number of vertices are left and then completes with a serial algorithm. In this work, we will run coloring until all SCCs are discovered to minimize the proportion of any serial work for our GPU algorithms.

**3. Methodology.** Here we will present the various implementations of breadth-first search used for the forward-backward phase of the Multistep algorithm. Although we explicitly demonstrate only breadth-first search, all of these implementations are easily extensible to color propagation via altering some initializations and the contents of the innermost loop, which will later be demonstrated.

For all of the following, we consider starting a BFS from a given vertex *root* on a graph $G(V, E, E')$ with vertex set $V$, out-edges $E$, and in-edges $E'$. Note that we also do not explicitly create a BFS tree through marking parent vertices or tracking levels, as we only care about reachability from the given *root*.

For Kokkos team-based parallelization, we break up the available work (e.g. the size of the current level queue $Q$) into equal portions of constant size. This work partitioning is performed explicitly by us, with the most performant chunk size being dependent on the algorithm and runtime architecture. From our observations, we see chunk sizes equivalent to about 1 to 8 vertices per team member to be most performant on GPU and chunk sizes of several thousand per thread to be most performant on CPU. Each of these work portions is assigned to a thread team for dispatch and execution across a GPU multiprocessor or CPU core. For simplification of presentation for the following algorithms, we don't explicitly show this work distribution. However, consider this to be implicitly performed before any loop with **in parallel**.

**3.1. Baseline Parallelization.** The baseline parallelization of BFS is given by Algorithm 4. Here, we utilize several common optimizations, including a *visited* array to mark reachability status from *root*, a global current-level queue $Q$, a global next-level queue $N$, and thread-owned queues $Q_t$. Upon each insertion of a new vertex into a thread-owned queue, the current size of the queue is checked and the contents are moved to the global queue if the thread-owned queue is full. The usage of thread-owned queues decreases the amount of global synchronization required while increasing cache utilization. The size of the thread-owned queue is dependent on cache size for performance reasons. E.g. $Qt_SIZE$ on a CPU might be several thousand while on a GPU it might be only 16.

This general approach suffers on GPU with irregular graphs for several reasons. On highly skewed graphs, there is severe under-utilization of the entire warp and Kokkos thread team when a large disparity exists in the number of neighbors for each $v$ retrieved from $Q$ for each thread. Additionally, each thread must independently retrieve new offsets for insertion into $N$ via atomic operations. Kokkos has a `team_scan` operation which allows calculation of offsets for an entire team with only

**Algorithm 4** Baseline BFS algorithm

```
visited(1 ··· n) ← 0
visited(root) ← 1
Insert root into Q
while Q ≠ ∅ do
    for all v ∈ Q in parallel do
        Remove v from Q
        for all ⟨v, u⟩ ∈ E do
            if visited(u) = 0 then
                visited(u) = 1
                Insert u into Q_t
                if |Q_t| = Qt_SIZE then
                    for all w ∈ Q_t do
                        Remove w from Q_t
                        Insert w into N
    Move contents of all Q_t to N
    Swap(Q, N)
```

a single global atomic required. However, this `team_scan` function cannot be simply called within Algorithm 4 due to the fact that it is a blocking operation for the entire team, but not necessarily every thread in the team will reach the inner loop where the function needs to be called. There are workarounds to this issue, such as breaking up the loop over $\langle v, u\rangle \in E$, but this introduces additional overheads.

**3.2. Bottom-Up and Hybrid Parallelization.** The bottom-up BFS approach is demonstrated in Algorithm 5. This approach benefits from not requiring an explicit queue or any global synchronization. While the approach is iterative, it is not explicitly *level-synchronous* since we don't require creations of the BFS tree, i.e. a vertex $w$ that is three hops away from *root* might be visited on the first iteration if all vertices between *root* and $w$ are visited prior to $w$ during the first iteration. This can result in a much smaller number of iterations being required than the distance between *root* and the farthest vertex reachable from it. Additionally, the inner-most loop is limited by the fact that only a single explored vertex $u$ needs have been previously visited, and, since we're considering real-world graphs, it is likely that most large degree vertices $v$ are within a few hops of each other which reduces the number of times the neighbors of $v$ need to be examined before $v$ can be marked as visited.

**Algorithm 5** Bottom-Up BFS algorithm

```
visited(1 ··· n) ← 0
visited(root) ← 1
changes = 1
while changes > 0 do
    changes = 0
    for all v ∈ V in parallel do
        if visited(v) = 0 then
            for all ⟨v, u⟩ ∈ E' do
                if visited(u) = 1 then
                    visited(v) ← 1
                    changes ← changes + 1
                    break
```

This approach suffers due to the fact that all $v \in V$ are examined on each iteration, so for high diameter graphs and on the first few and last few iterations on small

diameter graphs a lot of extra work is required. It is possible to maintain a set of unvisited vertices that gets updated on each iteration, but for a relatively disconnected graph this has minimal benefit.

Due to these drawbacks, it is common to utilize the bottom-up algorithm in conjunction with the baseline top-down algorithm. This is referred to as the hybrid BFS approach of Beamer et al. [2]. With a hybrid BFS approach, the baseline algorithm is performed for the first few levels, until, based on the size of the queue and number of currently unexplored vertices, it is deemed more work-efficient to switch to the bottom-up algorithm. A switch can again occur after some number of iterations back to the top-down algorithm.

One final thing to note is that using a bottom-up approach can reduce total required memory utilization. This is because with both FW-BW and color propagation-reverse color propagation, one can do forward searches top-down and reverse searches bottom-up to only require out-edges. This essentially reduces total memory consumption for graph storage by half. We consider this case in our results for graphs larger than could otherwise be fully stored in the limited GPU memory.

**3.3. Inner Loop Parallelization.** Inner loop parallelization, given by Algorithm 6 and also referred to as *coarse-grained warp-based gathering* [14], is similar to the baseline approach. However, each thread team works in unison to explore the adjacencies of each vertex in the current level queue. For large degree vertices, this approach is preferred due to high warp utilization and the ability for synchronous insertions into the global next level queue. However, for low degree vertices, this approach again suffers from low warp utilization. Therefore a hybrid approach that combines the two approaches is preferred.

**Algorithm 6** Inner Loop BFS algorithm

```
visited(1 ··· n) ← 0
visited(root) ← 1
Insert root into Q
while Q ≠ ∅ do
    for all v ∈ Q do
        Remove v from Q
        for all ⟨v, u⟩ ∈ E in parallel do
            if visited(u) = 0 then
                visited(u) = 1
                Insert u into Q_t
                if Q_t = Qt_SIZE then
                    for all w ∈ Q_t do
                        Remove w from Q_t
                        Insert w into N
    Move contents of all Q_t to N
    Swap(Q, N)
```

**3.4. Delayed Exploration.** Delayed exploration, given by Algorithm 7 and also referred to as *deferring outliers* [9], aims to maximize warp utilization by performing baseline parallelization on small degree vertices delaying large degree vertices to be explored with inner loop parallelization.

To do this, a separate per-thread team queue $Q_D$ is maintained in shared memory. When a thread encounters a high degree vertex, that vertex is placed into the queue for later expansion by the entire team. This enables higher warp utilization by limiting the serial expansion of any given neighbors list by a single thread.

**Algorithm 7** Delayed exploration-based BFS algorithm

```
visited(1 ··· n) ← 0
visited(root) ← 1
Insert root into Q
while Q ≠ ∅ do
    for all v ∈ Q in parallel do
        Remove v from Q
        if outDegree(v) < MAX_e then
            for all ⟨v, u⟩ ∈ E do
                if visited(u) = 0 then
                    visited(u) = 1
                    Insert u into Q_t
        else
            Q_D ← v
        for all w ∈ Q_t do
            Remove w from Q_t
            Insert w into N
    for all v ∈ Q_D do
        Remove v from Q_D
        for all ⟨v, u⟩ ∈ E in parallel do
            if visited(u) = 0 then
                visited(u) = 1
                Insert u into Q_t
                if Q_t = Qt_SIZE then
                    for all w ∈ Q_t do
                        Remove w from Q_t
                        Insert w into N
    Swap(Q, N)
    Q ← ∅
```

**3.5. Manhattan Collapse.** The Manhattan loop collapse [1], given by Algorithm 8 and Algorithm 9 and also referred to as *fine-grained, scan-based cooperative expansion* [14], aims to maximize warp utilization across the entire work set. To do this, prefix sums are computed based on the out degrees of each vertex in the queue. Parallelization is then performed over the total calculated workload, with the current iteration's vertex and adjacency (in Algorithm 8 *Adj* refers to a given vertex's adjacencies array) being computed based on a binary search (Algorithm 9) with the iteration counter over the prefix sums.

To simplify calculation of our prefix sums, we take a work chunk from the queue to be equal to that of the size of our thread team. A single `team_scan` based on out-degrees for a thread-specified vertex in the work chunk will compute the prefix sums, which are stored in a shared memory array. The Manhattan loop collapse offers the highest overall warp utilization out of all of the aforementioned approaches, but at a cost of the HighestLessThan search for each iteration *index*. However, since the prefix sums list is limited in size and is in localized storage, computation of the *index* is relatively inexpensive.

**3.6. Chunking.** A final approach that aims to maximize warp utilization by parallelizing over the total work while not requiring additional overheads of calculating the vertex and adjacency for each work iteration is termed as chunking, and is given by Algorithm 10. In this approach, only some limited number of any given vertex's adjacencies will be expanded by a given thread. If the number of adjacencies exceeds the given limit, the remaining adjacencies are broken up into *chunks* to be explored

**Algorithm 8** Manhattan Loop Collapse BFS algorithm

```
visited(1 ··· n) ← 0
visited(root) ← 1
Insert root into Q
while Q ≠ ∅ do
    P ← PrefixSumsOfVertexOutdegrees(Q)
    s ← P(|Q| + 1)
    for i = 1 ··· s in parallel do
        index =HighestLessThan(P, i)
        v ← Q(index)
        u ← Adj(i − P(index))
        if visited(u) = 0 then
            visited(u) = 1
            Insert u into Q_t
            if Q_t = Qt_SIZE then
                for all w ∈ Q_t do
                    Remove w from Q_t
                    Insert w into N
    Q ← ∅
    Swap(Q, N)
```

**Algorithm 9** Highest-less-than algorithm for Manhattan collapse

```
procedure HighestLessThan(P, val)
    found ← 0
    bound_low ← 0
    bound_high ← |P| − 1
    while found = 0 do
        index = (bound_high + bound_low)/2
        if P(index) ≤ val and P(index + 1) > val then
            found = 1
        else if P(index) < val then
            bound_low = index
        else
            bound_high = index
    return index
```

on subsequent iterations. Because this approach does not preserve level-synchronicity, it has not previously been considered for BFS (the number of total iterations can be dependent on the *log* of the largest out degree in the graph in the base of the number of chunks created during each chunking phase).

To do the chunking, rather than store a single vertex $u$ into the queue, its start and end offsets in the adjacency list of a CSR representation, given by $I(u)$ and $I(u+1)$, are stored into two separate queues for the start and end chunks, $N_s$ and $N_e$. On a subsequent iteration when a thread reads the start and end pointers, it begins to explore adjacencies from the start. If the difference between the start and end pointers is greater than our pre-specified chunk size, the remaining adjacencies are placed back into the queue into two chunks as given by Algorithm 10. The number of chunks can be any arbitrary number, but two is chosen here for simplicity. By limiting the amount of work any given thread does, warp-based imbalance is minimized.

**3.7. Extension to Color Propagation.** Extending all of the aforementioned approaches to color propagation is fairly straightforward, and an overview of the main difference to the inner loop is given by Algorithm 11. Instead of maintaining and

**Algorithm 10** Chunking-based BFS algorithm

```
visited(1 ··· n) ← 0
visited(root) ← 1
Insert root into Q
Insert I(root) into Q_s
Insert I(root + 1) into Q_e
while Q ≠ ∅ do
    for all i ∈ |Q| in parallel do
        v ← Q(i), v_s ← Q_s(i), v_e ← Q_e(i)
        end ←Min(v_s + MAX_e, v_e)
        for j = v_s ··· (end − 1) do
            u ← Adj(j)
            if visited(u) = 0 then
                visited(u) ← 1
                Insert u into N
                Insert I(u) into N_s
                Insert I(u + 1) into N_e
        if end < v_e then
            Insert v into N
            Insert end into N_s
            Insert (v_e + end)/2 into N_e
            Insert v into N
            Insert (v_e + end)/2 into N_s
            Insert v_e into N_s
    Q ← ∅, Q_s ← ∅, Q_e ← ∅
    Swap(Q, N), Swap(Q_s, N_s), Swap(Q_e, N_e)
```

checking a *visited* array, comparisons are performed with a per-vertex current colors array $C$. We additionally maintain $InQ$, which specifies if a vertex has currently been placed into the next level queue. To avoid global synchronization on color updates, when a vertex $v$ propagates a color to another vertex $u$, both $v$ and $u$ are added to the next level queue, as it is possible the color from $v$ overwrote a superior color that was being concurrently written by some other neighbor of $u$.

**Algorithm 11** Demonstration of inner loop of color propagation

```
for all v ∈ V do
    C(v) ← v
...
v = vertex from Q
u = current neighbor
if C(v) > C(u) then
    C(u) ← C(v)
    if InQ(u) = 0 then
        InQ(u) = 1
        Insert u into N
    if InQ(v) = 0 then
        InQ(v) = 1
        Insert v into N
...
Q ← ∅
for all v ∈ V do
    InQ(v) ← 0
Swap(Q, N)
```

Because the backward stage of color propagation for SCC computation can be considered a multi-source BFS, all of the approaches still apply. However, instead of only checking *visited* status, a vertex marks a neighbor only if it is both unvisited and both vertices have the same color.

**4. Experimental Setup.** Experiments were performed on *Shannon*, a dual socket system with 128 GB main memory, two Intel Xeon E5-2670 processors, and NVIDIA Tesla K40M GPUs. The K40M GPUs each have 12 GB DDR5 memory and 2880 CUDA cores running at 745 MHz. The version of Kokkos used for the tests came from release 11.10.1 of Trilinos.

Several real small-world directed graphs were used for testing, and are given in Table 4.1. These graphs range in scale from 800 K to 600 M edges and were retrieved from the SNAP database [12], the Koblenz Network Collection [11], the University of Florida Sparse Matrix Collection [6], and the Max Planck Institute [5]. The bottom three listed graphs are only considered for the *out-edge only* case, as they are too large for storage of both out and in-edges in the memory of our test GPU.

| **Network** | $n$ | $m$ | *deg* | | (S)CCs | | |
|---|---|---|---|---|---|---|---|
| | | | avg | max | count | nontrivial | max |
| Google | 875 K | 5.1 M | 5.8 | 5 K | 370 K | 12 K | 410 K |
| LiveJournal | 4.8 M | 69 M | 14 | 20 K | 970 K | 23 K | 3.8 M |
| Indochina | 7.4 M | 194 M | 26 | 180 K | 1.6 M | 40 K | 3.8 M |
| HV15R | 2.0 M | 283 M | 140 | 170 K | 24 K | 15 | 120 K |
| uk-2002 | 18 M | 398 M | 16 | 4 K | 3.7 M | 70 K | 12 M |
| WikiLinks | 26 M | 600 M | 23 | 400 K | 6.6 M | 60 K | 19 M |
| it-2004 | 41 M | 1.2 B | 28 | 10 K | 6.8 M | 150 K | 30 M |
| sk-2005 | 50 M | 1.9 B | 39 | 2.9 M | 8.8 M | 46 K | 35 M |
| Twitter | 53 M | 2.0 B | 37 | 780 K | 12 M | 125 K | 40 M |

Table 4.1: Information about test networks. Columns are # vertices, # edges, average and max. degree, # of SCCs, # number of nontrivial SCCs, and size of the largest SCC.

These graphs were selected to represent a wide mix of graph sizes and topologies. The number of total and nontrivial SCCs as well as the max SCC size all play an important role in the general performance of decomposition algorithms. In general, the graph topology also has a strong influence over the performance of the BFS and color propagation subroutines.

**5. Results.** Here, we will demonstrate our results in four parts. For our GPU algorithms, we will compare the performance of all of the BFS, color propagation, and reverse color propagation algorithms. We will then choose the overall best performing algorithm for each Multistep stage and compare the total execution times for a full strongly connected components decomposition to the original CPU approach. For the GPU approaches, we fix thread queue sizes, delayed exploration cutoffs, and chunk sizes at 16.

**5.1. Breadth-First Search.** Figure 5.1 gives the running times of the numerous bread-first search algorithms across our suite of test graphs. These results are for a

single breadth-first search, so doubling the times would give an approximate running time for the first stage of the Multistep algorithm. Results are given for the five top-down approaches, two bottom-up approaches, and four hybrid approaches.

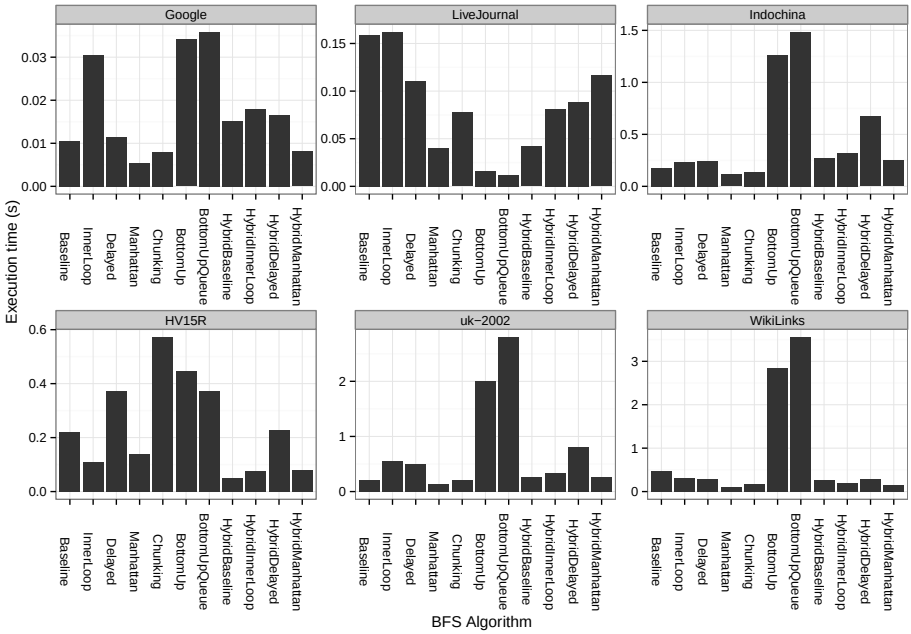



Fig. 5.1: Performance of various breadth-first search algorithms across the test graphs.

Overall, the top-down Manhattan loop collapse demonstrates the most consistent performance across the various graph topologies. The pure bottom-up approaches are consistently the worst with the exception of LiveJournal, which is likely due to favorable vertex ordering in the given graph. It was observed that the bottom-up approaches finished in only six to eight iterations, considerably less than LiveJournal's approximate diameter of twenty. Chunking and the hybrid approaches also show relatively good performance across most of the graphs.

Due to these performance results, we utilize the top-down Manhattan loop collapse in our GPU SCC algorithm. The Multistep CPU algorithm previously utilized an approach most similar to HybridBaseline.

**5.2. Color Propagation.** Color propagation was implemented with the same top-down optimizations except for chunking. As mentioned, due to the possible recoloring of large degree vertices, the size of the queue can grow considerably to exceed available memory on the larger test instances. A smarter dynamic-queue-based approach would alleviate this issue, and is saved for future work. We also include a naive algorithm for comparison on select instances. In the naive approach, we avoid using queues and simply examine all vertices on each iteration. This was the original approach for color propagation parallelization. Note that several of the execution times for the naive method are excluded from the plot, due to poor performance orders of magnitude slower than the other methods.

The color propagation running time results are given by Figure 5.2. The results

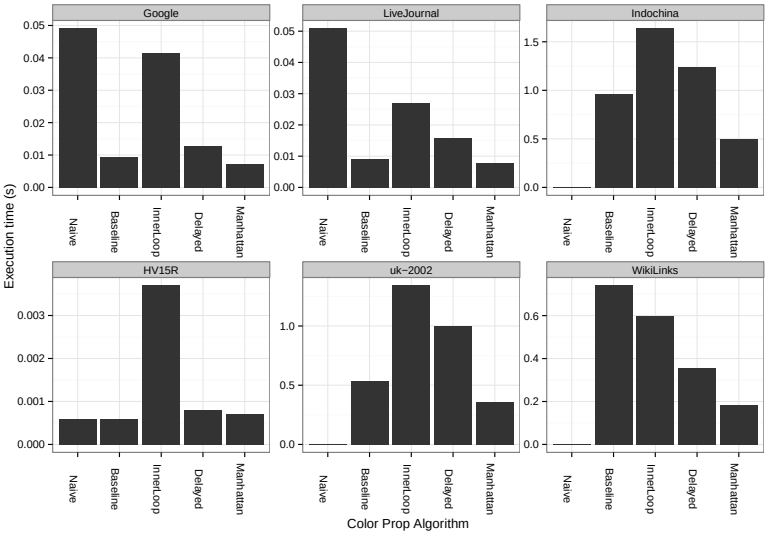



Fig. 5.2: Performance of various color propagation algorithms across the test graphs.

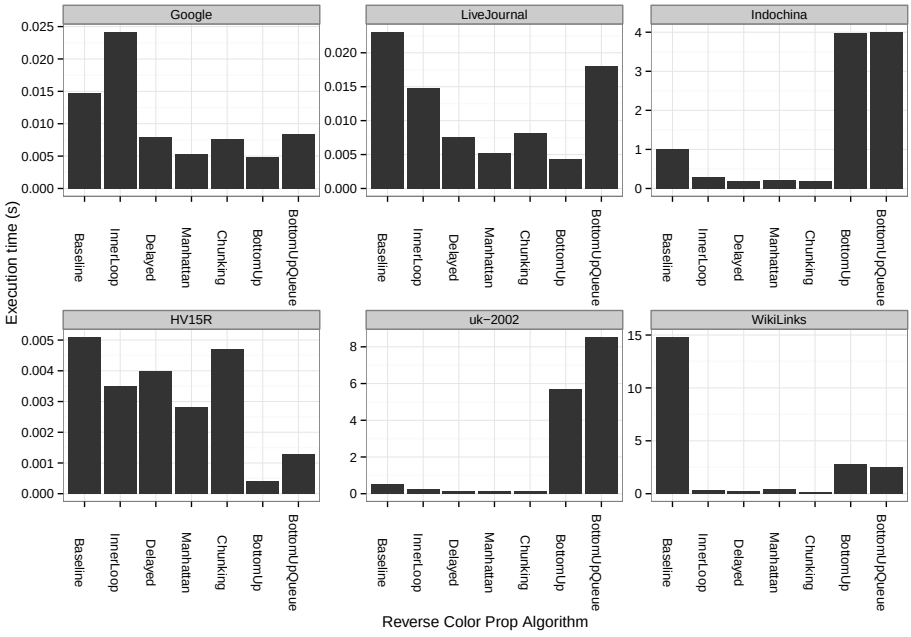



Fig. 5.3: Performance of various reverse color propagation search algorithms across the test graphs.

shown are the total time spent for color propagation over a full run of the Multistep algorithm, that is, the times are taken to perform color propagation with the largest strongly connected component removed. We choose to take these times as apposed

to running color propagation on its own as these results are more relevant to SCC detection in this context. However, select individual runs for propagating across the full graph have resulted in similar trends.

Overall, we again observe the best performance with the Manhattan loop collapse. This is likely due to the skewed degree distributions inherent to all of these networks. Again, based on these results we utilize the Manhattan loop collapse in the GPU SCC algorithm. The original Multistep CPU algorithm is most similar to the Baseline algorithm shown here.

Figure 5.3 gives the running time results for the reverse color propagation. Once again, these are taken as the sum over however many necessary iterations with the largest SCC previously removed. We note again consistent fast performance with the Manhattan collapse and poor performance with the bottom-up algorithms. We however note that chunking outperforms the Manhattan collapse on the larger networks where there is a considerable number of non-trivial SCCs. This is likely due to the lower overall average but still skewed degree distribution post-removal of the largest component. Due to this observation, we utilize chunking for the reverse propagation stage of our GPU algorithm. Again, the Multistep approach is most similar to the Baseline.

**5.3. Putting it All Together.** We now give the total running time of our strongly connected components algorithm. The results for five different configurations are shown in Figure 5.4. The labeling follows **Algorithm-Runtime-Hardware**, where Algorithm is the original Multistep versus the modified GPU algorithm, the runtimes are the original OpenMP and Kokkos, and the hardwares are CPU and GPU.

Most obviously from Figure 5.4 is the poor performance of the two algorithms running on hardware opposite that they were designed for. The original Multistep algorithm compiled with OpenMP and running on CPU shows the most consistent performance, followed by the GPU Kokkos algorithm running on GPU. For the two largest and most complex instances, uk-2002 and WikiLinks, the GPU-Kokkos-GPU configuration runs fastest. This is due to the higher available parallelism available on these instances and the lower relative overhead costs.

Finally we show the results from using our top-down bottom-up out-edge only approaches in Figure 5.5. This algorithm for GPU utilizes the Manhattan loop collapse on the top-down BFS and color propagation stages and the bottom-up algorithm for the backward BFS and reverse color propagation stages. We include the original Multistep-OpenMP-CPU running times for comparison as well as the GPU Algorithm-Kokkos-GPU algorithm for where the graph is able to fully fit in GPU memory.

We can observe that there is a considerable cost for most of the graphs for using the out-edge only approach. This is especially prevalent on the high diameter webcrawls (indochina, it, sk, uk), where the bottom-up algorithm performs especially poorly. Looking at the execution time breakdown from the various stages, it is apparent that the reverse color-propagation stages account for a vast majority of the total running time for most test instances.

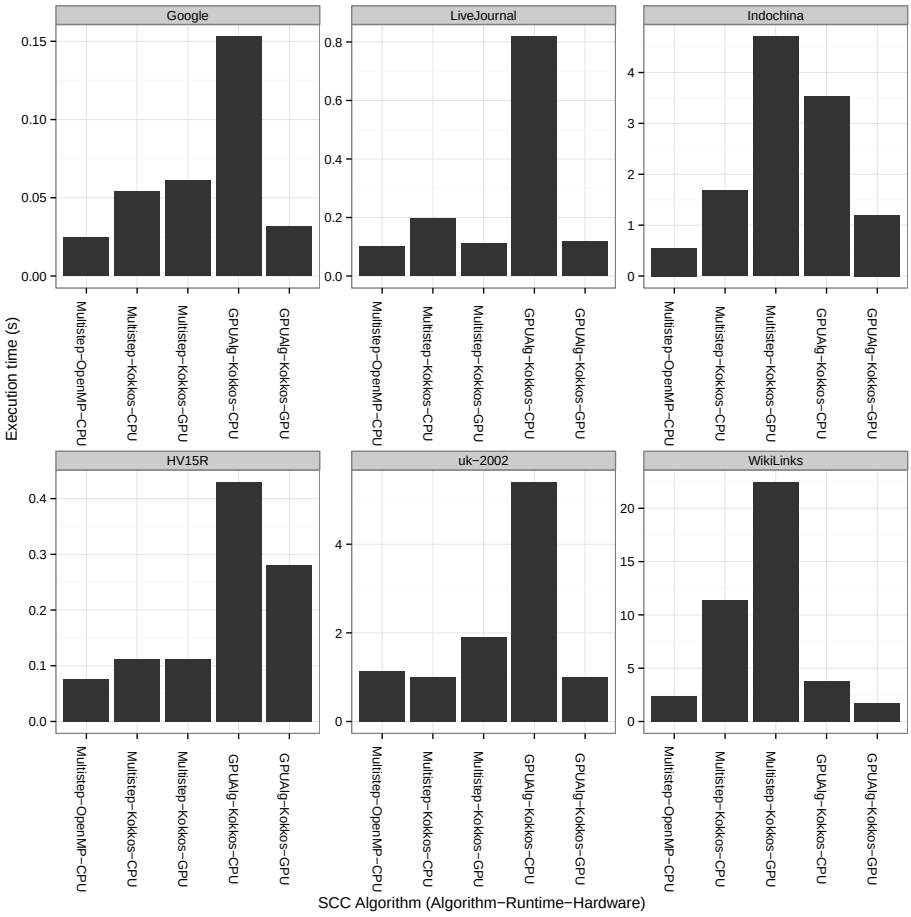



Fig. 5.4: Performance of various reverse strongly connected components algorithms across the test graphs and architectures.

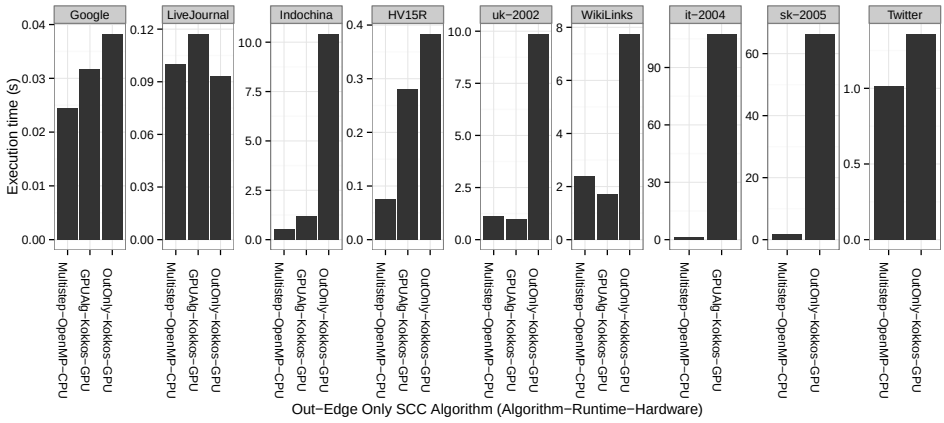



Fig. 5.5: Performance of various reverse out-edge only strongly connected components algorithms across the test graphs and architectures.

**6. Conclusions and Future Work.** This work examined the conversion of a CPU optimized algorithm for performance on GPU for the problem of strongly connected components. While implementing various optimizations to better utilize the wide parallelism available on GPU, we observe that the Manhattan loop collapse and chunking approaches tended to perform best across the various test instances. For the larger test graphs with more available parallelism and lower relative overhead, the GPU approach offered up to 1.5× speedup on our test system for a complete SCC decomposition algorithm. We additionally note that it is likely necessary to store both in and out-edges for performance reasons when computing SCCs on large high-diameter graphs.

Future work might extend the aforementioned optimizations to various other connectivity-based problems, including connected, weakly connected, and biconnected components algorithms. Additionally, an exploration into the parameters controlling the hybrid BFS approaches, delayed exploration, and chunking would lend better insight into the performance of these approaches. Additionally, storing and transferring in and out-edges dynamically for each forward-backward stage of BFS and color propagation during GPU computation might considerably improve execution times compared to the out-edge only algorithm while allowing larger graph instances to be run.

REFERENCES

[1] *Optimizing loop-level parallelism in cray xmt applications*, tech. rep., Cray Inc., 2009.
[2] S. Beamer, K. Asanović, and D. Patterson, *Direction-optimizing breadth-first search*, Scientific Programming, 21 (2013), pp. 137–148.
[3] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener, *Graph structure in the web*, Computer networks, 33 (2000), pp. 309–320.
[4] H. Carter Edwards, C. R. Trott, and D. Sunderland, *Kokkos: Enabling manycore performance portability through polymorphic memory access patterns*, Journal of Parallel and Distributed Computing, (2014).
[5] M. Cha, H. Haddadi, F. Benevenuto, and K. P. Gummadi, *Measuring User Influence in Twitter: The Million Follower Fallacy*, in Proc. 4th Int'l. AAAI Conf. on Weblogs and Social Media (ICWSM), May 2010.
[6] T. A. Davis and Y. Hu, *The university of florida sparse matrix collection*, ACM Transactions on Mathematical Software, 38 (2011), pp. 1–25.
[7] H. C. Edwards and D. Sunderland, *Kokkos array performance-portable manycore programming model*, in Proceedings of the 2012 International Workshop on Programming Models and Applications for Multicores and Manycores, ACM, 2012, pp. 1–10.
[8] H. C. Edwards and C. R. Trott, *Kokkos: Enabling performance portability across manycore architectures*, in Extreme Scaling Workshop (XSW), 2013, IEEE, 2013, pp. 18–24.
[9] S. Hong, S. K. Kim, T. Oguntebi, and K. Olukotun, *Accelerating cuda graph algorithms at maximum warp*, ACM SIGPLAN Notices, 46 (2011), pp. 267–276.
[10] S. Hong, N. C. Rodia, and K. Olukotun, *On fast parallel detection of strongly connected components (scc) in small-world graphs*, in Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis, ACM, 2013, p. 92.
[11] J. Kunegis, *KONECT - the koblenz network collection.* konect.uni-koblenz.de, last accessed 31 July 2013.
[12] J. Leskovec, *SNAP: Stanford network analysis project.* http://snap.stanford.edu/index.html, last accessed 3 July 2013.
[13] W. Mclendon Iii, B. Hendrickson, S. J. Plimpton, and L. Rauchwerger, *Finding strongly connected components in distributed graphs*, Journal of Parallel and Distributed Computing, 65 (2005), pp. 901–910.
[14] D. Merrill, M. Garland, and A. Grimshaw, *Scalable gpu graph traversal*, in ACM SIGPLAN Notices, vol. 47, ACM, 2012, pp. 117–128.
[15] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, *Measurement and analysis of online social networks*, in Proceedings of the 7th ACM SIGCOMM

conference on Internet measurement, ACM, 2007, pp. 29–42.
[16] S. Orzan, *On Distributed Verication and Veried Distribution*, PhD thesis, Free University of Amsterdam, 2004.
[17] A. Pothen and C.-J. Fan, *Computing the block triangular form of a sparse matrix*, ACM Transactions on Mathematical Software (TOMS), 16 (1990), pp. 303–324.
[18] G. M. Slota, S. Rajamanickam, and K. Madduri, *Bfs and coloring-based parallel algorithms for strongly connected components and related problems*, in International Parallel & Distributed Processing Symposium (IPDPS), 2014.
[19] W. McLendon III, B. Hendrickson, and S. J. Plimpton, *Finding strongly connected components in distributed graphs*, Lecture Notes in Computer Science, 1800 (2000), pp. 505–512.

# A FUNCTION SHIPPING LAYER FOR THE KITTEN LIGHTWEIGHT KERNEL

JORGE CABRERA* AND KEVIN PEDRETTI†

**Abstract.** In this work, we implement a function shipping layer for the Kitten lightweight kernel. This layer enables the Kitten kernel to proxy system call requests that would normally be performed locally to a remote system for external execution. As a test case for our framework, we implement an I/O forwarding layer built using our function shipping layer. This I/O forwarding layer allows I/O requests to be intercepted at Kitten's system call level and then forwarded to a remote Linux front-end node for servicing. Our current system allows for applications to keep their existing interfaces, without the need to modify user code or rebuild. This paper presents the high-level design and implementation details of the Kitten function shipping layer.

**1. Introduction.** Lightweight kernel (LWK) operating systems (OS), such as Puma/Cougar [9], Catamount [6], and CNK [5], typically function ship their I/O requests to remote I/O nodes for servicing, rather than handling them locally. This offloads the complexity of the I/O stack from the compute nodes to a dedicated set of specialized I/O nodes. In this work, we design and implement a new function shipping layer for the Kitten [7] LWK OS that, unlike prior work, allows unmodified Linux binaries (no recompile needed) to be executed on Kitten and have their I/O functions shipped off-node. The design of our layer is modular, keeps the majority of the function shipping functionality at user-level, and leverages the RDMA capabilities of HPC networks to move data efficiently.

The remainder of this paper is organized as follows: Sections 2 and 3 describe the basic approach and design of our function shipping layer. The details of our specific implementation are then described in Section 4. In Section 5 we describe how the function shipping layer was used as the foundation to develop an I/O forwarding layer for the Kitten LWK, as well as some of the key optimizations that were made. Related work is discussed in Section 6 and we conclude in Section 7.

**2. Approach.** Kitten's existing I/O layer only supported on-node, in-memory I/O handling. Its I/O stack is composed of a Virtual File System (VFS) layer that allows for Linux-based virtual file systems (e.g. /proc, /dev, /sys) to be created. It also supports in-memory file I/O, which leverages the system's memory to store file contents for the duration of a program's execution. Because of these limitations, Kitten could only support applications that had minimal I/O requirements.

The Kitten Function Shipping Layer (KFSL) is designed to expand the limited I/O services provided by Kitten's current VFS layer. Kitten's filesystem capabilities can be expanded through the use of a function shipping mechanism. By supporting an I/O forwarding layer, Kitten can transfer its filesystem I/O services to a remote system capable of handling more complex and large scale I/O functionality. Such an I/O forwarding layer can be built using a function shipping layer that intercepts system calls and forwards them to a handler on a remote server.

Our approach is to build a Kitten I/O forwarding layer that intercepts system calls for non-system I/O services. Kitten's VFS layer would still handle I/O for its

*Florida International University, jcabr020@fiu.edu

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

pseudo-file systems (/proc, /dev, /sys), however, it would function ship system call invocations for I/O services such as sys_open, sys_read, sys_write, sys_mkdir, etc.

**3. Design.** There are several techniques that can be used implement a function shipping layer, all of which result in a variety of advantages and disadvantages when it comes to issues such as performance, compatibility, and functionality. Thain and Livny discuss seven classes of interposing agents; frameworks that "transform standard interfaces into remote I/O protocols not normally found in an operating system" [8]. These classes are separated by their level of intrusion into the application being executed. The first type of these is internal techniques. These are the class of interposing techniques that modify the memory space of the original application. An example of this approach is modifying the application to use a new implementation of an existing interface. However, this requires the application to be rebuilt, an option that is not possible when the source code is not available. The other type of interposing techniques are external. External techniques are those where an application's operation is intercepted at a level that is outside of the application's address space. An example of this approach is the interception of system calls. This class of interposing technique would use a different implementation of the system call (e.g., a remote node's system call) instead of that of the local OS.

The KFSL is an example of the external interposing technique because it is built at the system call level of the Kitten OS. The KFSL captures the arguments of a system call and sends them off to a user-space daemon that is in charge of forwarding the requests to a remote request handler. The KFSL can be structurally separated into three main components, the Kernel Interposing Agent, the Request Forwarding Daemon, and the Remote Request Handler. These components are described in the following subsections.

**3.1. Kernel Interposing Agent.** The Kernel Interposing Agent (KIA) is a mechanism that intercepts the invocation of a system call and packages its arguments to send them to the user space Request Forwarding Daemon (RFD). In essence, the KIA can nullify the Kernel's local system call, and replace its result with the response returned by the RFD.

The following is an example of how a call to the `write()` POSIX I/O function is captured:

1. Application: When the application makes a call to the `write()` function, its arguments are passed on from user space to the kernel via the normal system call mechanism. In the kernel, the call is handled by the `sys_write()` handler.
2. System Call: The corresponding call is intercepted by way of a function hook. In essence, inside the system call the kernel will first check if the file belongs to one of the pseudo file systems (/proc, /dev, /sys). If not then the arguments of the system call are shipped.
3. Argument Packaging Hook: Each system call has a corresponding hook, which will wrap a system call's arguments into a special request structure and then insert it into a pending request queue. A user-space daemon will fetch each request one by one from the pending queue, and then forward it to a remote server.

**3.2. Request Forwarding Daemon.** The Request Forwarding Daemon (RFD) is a user space daemon whose job is to fetch system call requests from the KIA, and then send the corresponding packaged requests to a remote server. Once the RFD

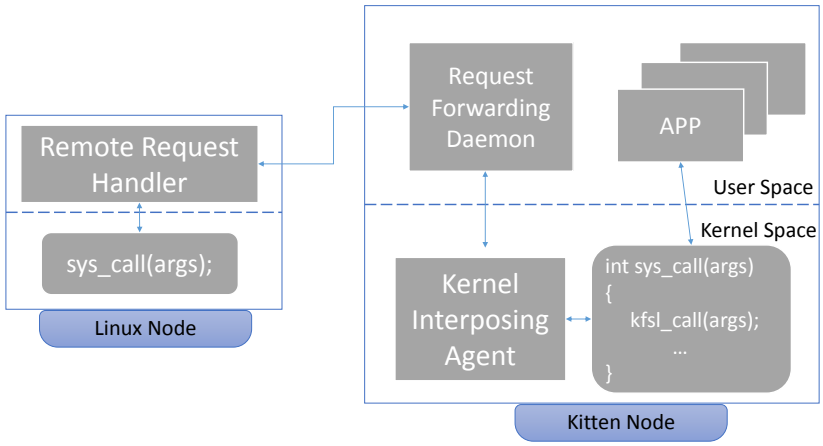



Fig. 3.1: Overview of the Kitten Function Shipping Layer.

receives the response from the remove server, it will send it back to the KIA, which will in turn send it back to the application. Currently the RFD sends one request at a time to the remote server.

**3.3. Remote Request Handler.** The Remote Request Handler (RRH) is a daemon running on an remote I/O server that is responsible for receiving system call requests from a RFD and executing the requests locally. The requests that the RRH receives from the RFD contain information such as the system call type, its arguments, network contact information from the initiator node, and a unique request identifier.

**4. Implementation.** This section describes the implementation details of the Kitten function shipping layer's KIA, RFD, and RRH components. The KIA is privileged OS-level code implemented inside of the Kitten kernel, while the RFD and RRH are implemented at user-level.

**4.1. KIA.** There are two core functions that are performed by the KIA component of the KFSL: function argument passing from the user application to the KIA, and the communication between kernel and user space.

The KIA has a series of function hooks, one for each system call that is intercepted. Each function hook has an identical prototype to that of the system call. The hook's job is to capture the system call's arguments and package them into a special data structure. These special data structures are then embedded into a request which is inserted into a queue of pending requests. The caller application will be put to sleep until the request has been completed and returned with a response.

One of the data structures contained in a request used by the KFSL is the kfs_in_header:

```
struct kfsl_in_header {
        __u32   len;
        __u32   opcode;
        __u64   unique;
};
```

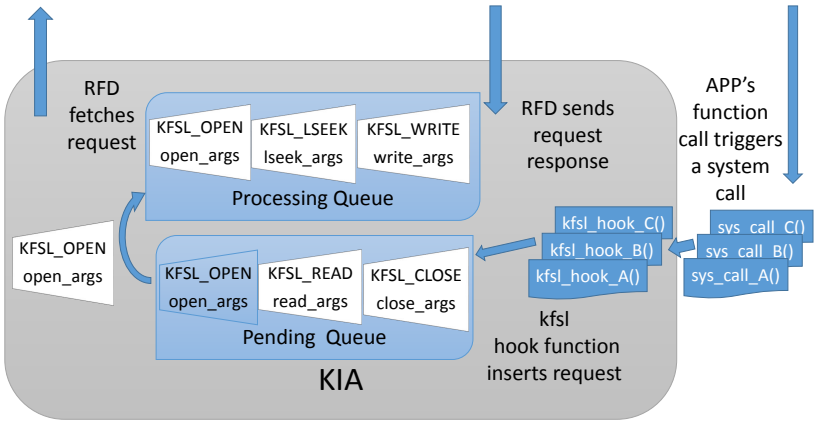



Fig. 4.1: Structure of KIA.

A kfsl_in_header struct contains general information about each request: the length of the header plus the arguments, the opcode or type of system call, and a unique id. The request is followed by the struct containing the arguments of the system call. Since each system call has different arguments, there is one struct for each of these. Here is an example of the arguments struct for the read system call:

```
struct kfsl_read_in {
        __u64    priv_data;
        __u64    buffer;
        __u32    size;
        __u32    offset;
        __u32    pos;
};
```

These requests are fetched by the RFD through the use of a special kernel file located in the /dev directory, in a similar way to how FUSE works [2]. This special file is used to allow communication from the user space RFD and the kernel space KIA.

**4.2. RFD.** The RFD fetches a single request at a time from the KIA and then forwards it to the RRH. In the future the implementation will be improved to process multiple requests at a time, allowing multiple requests to be pipelined to the RRH.

As previously mentioned, the RFD uses the special /dev file to fetch requests from the KIA. Basically, the RFD will run a loop that constantly tries to do a blocking read from the /dev file. A read to this file launches a call to special handler in the KIA which will check to see if there are any requests in the pending queue. If there are it will dequeue a request and copy it to a user space buffer provided by the RFD. The request is moved from the pending queue to a processing queue. The processing queue is used to search for the request when it is completed. If there are no requests the RFD will be put to sleep using Kitten's waitqueue mechanism. The RFD is woken up when a new request is inserted in the pending queue.

Communication between the RFD on the Kitten node and the RRH on a remote

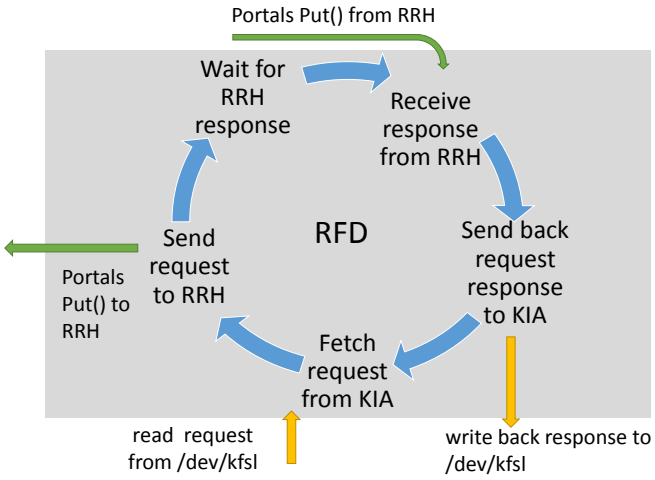



Fig. 4.2: Structure of RFD.

node is done using the Portals network programming API [1]. Portals is a special networking API designed for HPC systems. One of its main characteristics is that it is capable of moving data without the need for intermediary memory-to-memory copies, performing network transfers directly to and from user application buffers. Instead of multiple memory copies, which can be detrimental to performance for HPC systems, Portals supports the use of matching semantics to specify the memory buffer into which to copy the data directly. Specifically, there are two Portals operations which we use in our implementation of the RFD: Portals `PtlPut()` and `PtlGet()`. The `PtlPut()` operation allows us to send data from the RFD to the RRH, and the `PtlGet()` operation allows us to request data from the RRH. Using these two operations we implemented the RFD to perform a `PtlPut()` operation to send a request to the RRH server. The RFD then waits for the RRH server to perform a `PtlPut()` operation to the RFD, which writes the result of the operation into a special buffer allocated by the RFD. This special buffer is just a kfsl_out_header struct, which contains information about the return values of a system call:

```
struct kfsl_out_header {
        __u64    data;
        __u64    error;
        __u64    unique;
};
```

The data field contains a special value returned by the system call (if any) such as an error code, or file handle value. The error field contains special codes that represent internal errors that must be handled by KFSL. Lastly, the unique field contains the ID of the request that was just completed. This is the key used to search for the request in the processing queue.

Each I/O request from the application results in one request being received by the RFD. The RFD may internally chose to split up a large I/O request into a number smaller requests made to the RRH, but the current implementation does not do this.

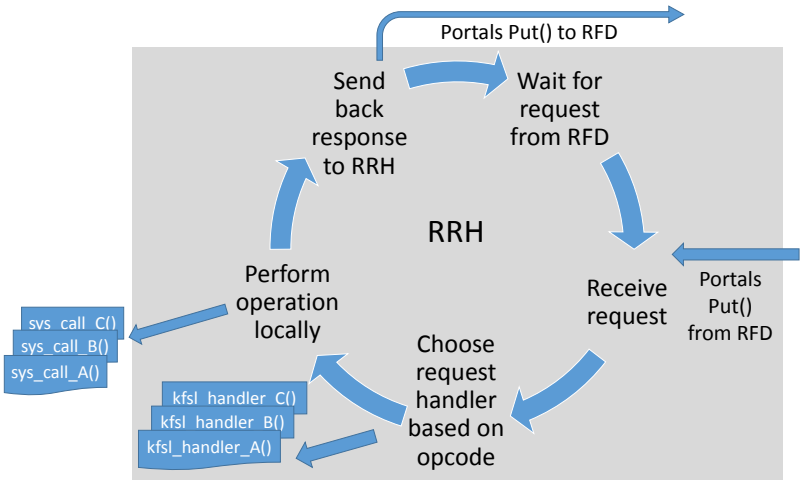



Fig. 4.3: Structure of RRH.

Each request results in a single `PtlPut()` or `PtlGet()` being used to move the request's data to or from the RRH.

**4.3. RRH.** The RRH is a daemon that runs on the remote server that is in charge of receiving system calls requests and performing them locally.

The RRH waits for a request to arrive using the Portals API's `PtlEQPoll()` interface. Once a request arrives, it will determine the type of request by checking the opcode, and will then unpack the request to the corresponding data structure which contains the system call arguments. The RRH will pass the request and its arguments to a specific handler, one of which exists one for each system call; this handler will perform the operation. Since requests do not include data, the handler is responsible for pushing or pulling payload data from the application running on the compute node. This is done using `PtlPut()` for read requests (data moves from the I/O node to the compute node) and `PtlGet()` for write requests (data moves from the compute node to the I/O node). The handler returns the result of the request to the RRH, which then packages it into a kfsl_out_header data structure and sends this back to the RFD client using a Portals `PtlPut()`.

**5. I/O Forwarding Layer.** An I/O Forwarding Layer (IOFL) has been implemented using Kitten's function shipping framework. In order to implement the IOFL, there are few components which must added to the KFSL framework. First of all, a hook must be added to each system call that needs to be function-shipped. Currently the Kitten IOFL supports the forwarding of the following system calls: open, write, lseek, read, close, unlink, mkdir, and rmdir, so each contains a hook to intercept the arguments. Secondly, a new opcode together with its arguments data structure must be added to the framework to hold the values of the system call arguments. This is the data structure that will be sent as part of the request. Lastly, a handler must be implemented on the RRH to perform the operation for the new opcode. These are the basic modifications which must be done to the KFSL framework.

In addition to the functionality added to the KFSL, there are some special opti-

mizations which are done to reduce the overhead of the IOFL. The main optimization performed is the use of SMARTMAP as the memory management protocol used by the framework. SMARTMAP allows a portion of one process's address space to be mapped into another process's address space directly [3]. This mechanism allows us to bypass the need to perform multiple memory copies. For example, our framework uses SMARTMAP to copy data directly to and from the application's user space buffer, without the need to copy the data to a kernel buffer or intermediary RFD buffer.

**6. Related Work.** Interposing techniques, I/O forwarding layers, and function shipping frameworks systems are active research areas. Ali et al. developed the I/O Forwarding Scalability Layer (IOFSL), a scalable I/O forwarding framework for HPC systems [4]. In their work, they present a framework which allows the I/O functionality of an HPC system to be forwarded to dedicated I/O nodes. Their work is similar to the KFSL, but differs in the interposing techniques that they use for implementation. Specifically, they implement an API called ZOIDFS, which is used by the application to perform I/O that is forwarded by IOFSL. In addition, they implement a FUSE filesystem that can be used to support applications that cannot be recompiled. Our approach of intercepting at the system call layer eliminates the need to recompile an application to use our IOFL.

Thain et al. present an excellent overview of the different interposing techniques that can be implemented, as well as the pros and cons of using each [8].

**7. Conclusion.** We have presented the design and implementation of a function shipping layer for the Kitten Lightweight Kernel. This framework was utilized to create an I/O Forwarding Layer for Kitten, enabling I/O requests to be offloaded from Kitten compute nodes onto dedicated remote I/O nodes. Kitten's support for the SMARTMAP memory mapping protocol and Portals API were used to optimize our IOFL implementation, eliminating the need for intermediate data copies. In the future we plan to further optimize the communication path between the Kitten RFD client and Linux RRH daemon, as well as explore transforming our protocol from its current stateful connection design into a stateless protocol.

REFERENCES

[1] *The Portals 4 specification.* `http://www.cs.sandia.gov/Portals/portals4.html`.
[2] *Fuse: Filesystem in userpace.* `http://fuse.sourceforge.net`, 2013.
[3] R. Brightwell, K. Pedretti, and T. Hudson, *Smartmap: Operating system support for efficient data sharing among processes on a multi-core processor*, ACM/IEEE Conference on Supercomputing, (2008).
[4] N. A. et al., *Scalable i/o forwarding framework for high-performance computing systems*, Cluster Computing and Workshops, (2009), pp. 1–10.
[5] M. Giampapa, T. Gooding, T. Inglett, and R. Wisniewski, *Experiences with a Lightweight Supercomputer Kernel: Lessons Learned from Blue Gene's CNK*, in International Conference for High Performance Computing, Networking, Storage and Analysis (SC), Nov 2010, pp. 1–10.
[6] S. Kelly and R. Brightwell, *Software Architecture of the Lightweight Kernel, Catamount*, in 2005 Cray Users' Group Annual Technical Conference, Cray Users' Group, May 2005.
[7] K. Pedretti, *Kitten: A lightweight operating system for ultrascale supercomputers.* `https://software.sandia.gov/trac/kitten/wiki/KittenPresentations`, 2011.
[8] D. Thain and M. Livny, *Parrot: An application environment for data-intensive computing*, Scalable Computing: Practice and Experience, 6(3) (2005), pp. 9–18.
[9] S. R. Wheat, A. B. Maccabe, R. Riesen, D. W. van Dresser, and T. M. Stallcup, *Puma : An operating system for massively parallel systems*, Scientific Programming, 3 (1994), pp. 275–288.

# BALANCING POWER AND TIME OF MPI OPERATIONS

TAYLOR GROVES * AND KURT B. FERREIRA†

**Abstract.** The hypothesis of this paper is that: By adjusting message sizes and altering the underlying algorithms of collectives, a reduction in peak power of a system's communication operations is achievable, with reasonable costs to the throughput and latency of the network. To evaluate the hypothesis we tested the power and performance of point to point communication 0 in the MPI framework as well four algorithms for two collective 0: AllReduce and AllGather. For our testbeds, we found that power did differ for varying message sizes and mechanisms in point to point communications, but at extreme costs to bandwidth and latency making such a tradeoff impractical on current systems. Contrasting the results of the point to point communications, collective operations saw relatively stable power draw across differing algorithms despite the difference in message sizes.

**1. Introduction.** Power consumption has been identified as a massive obstacle to building an exascale computer. A power budget of 20MW [1] set by the United States Department of Energy (DOE) means that existing approaches to system design must be reevaluated, with the goal of maximizing flops per watt. As of this writing, the top rated computer from The Top500 list, Tianhe-2, consumes near 18 MW of this 20MW power budget while providing only 34 petaflops of computation [13]. Despite it's tremendous size, Tianhe-2 is one of the world's most energy efficient computers with a Green500 ranking of 49 [4], consuming 1,902 MFLOPS/watt. In order to match the DOE's goals for exascale computation, flops per watt must reach 50 GFLOPS/watt – a 25X increase in power efficiency. Improving the flops to watt ratio by this amount requires prudent use of power by system components and application design. In some cases, hard caps on power will necessitate strategies that reduce power at the expense of longer run-times and energy costs.

While there are many techniques for reducing the power of a system, some approaches are understood better than others. CPU frequency scaling is a well understood mechanism for reducing the power of a system at the expense of execution time. However, as future systems move towards an architecture of many cores, operating at low frequency, this technique may be limited in its effectiveness. Additionally, there is the case, where an application is computationally-bound or memory-bound, such that a reduction in power at the expense of network throughput has little impact on the overall runtime of the application.

In this work, we explore alternative strategies for reducing the power of a subset of MPI point to point and collective operations, as well as the impact of these strategies on the run time of the system. In Section 3 we explore the trade-off between power consumption and run time, for sending data using MPI buffers of varied size. We test this by performing ping-pong and streaming tests for both onload and offload network cards on systems with specialized power 0 capabilities. Later, in Section 4 we look at the effects that different algorithms for MPI_AllGather have on the runtime and power costs for different node counts. To clarify, the specific contributions of this paper are:

- An exploration of the effects of message sizes on the power/time of MPI point to point communications.

*University of New Mexico Department of Computer Science, tgroves@cs.unm.edu
†Sandia National Laboratories, kbferre@sandia.gov

- An evaluation of the power/time trade-off for different algorithms/implementations of MPI_AllGather.

In the past, algorithms and systems have treated power as a ancillary concern, with the primary focus being execution time. With increased interest in power-efficient design, this work provides systems and application developers insight, so that they may leverage approaches that provide the best trade-off between power and time of communication operations. Additionally, this work informs simulation design – revealing what features of communication are most significant and how they should be modeled, in order to provide an accurate representation of the power and run times of real systems.

**2. Background.** For this work, we limit our analysis to the Teller system at Sandia National Laboratories. Teller is comprised of 104 nodes, each with an AMD A10-5800K, 3.8GHz, quad-core processor. Nodes are connected by a onloaded, QLogic, quad data rate, Infiniband network. A subset of the Teller nodes are equipped with offloaded, Mellanox HCA's. All of our experiments utilize version 1.6.4 of OpenMPI. To measure power, our experiments utilize the PowerInsight framework. As described in [6], PowerInsight facilitates component level power and energy instrumentation in commodity hardware. PowerInsight avoids the observer effect by running on electrically separated hardware from the system being tested – allowing for a high sampling rate without perturbing the system being measured. For the purposes of this work, we sample instantaneous power at a rate of 75 Hz per power rail, across a total of 7 rails. These rails represent the power devoted to CPU, memory, local storage, network interface cards, and motherboard.

**3. The Many Small vs. the Few Large.** How to best break up information to send it across the network, is a question that has traditionally been directed towards finding the optimal network throughput. Coming into this work, we decided to explore the topic with a focus on power. We had two initial questions:

- What is the effect on power and throughput, for sending a fixed amount of data at varying message sizes?

- With respect to the question above, how do offloaded and onloaded HCA's differ?

In this section, we answer these questions with respect to the system described in Section 2. Additionally, we offer some insight as to the underlying factors responsible for the results observed. The goal being that these insights may be generalized, to provide an understanding of the the power and throughput trade-offs for a broader set of systems.

**3.1. Experiment Design.** In order to answer the questions above we ran a set of experiments using the NetPipe [10] benchmark. NetPipe is a protocol independent performance tool, which provides a measurements for latency and throughput of a network through a series of ping pong or streaming tests over increasing message sizes. NetPipe is extendable through modules allowing the evaluation of a variety of environments. For the purposes of this work we utilize the MPI module.

We evaluate the power and performance of MPI point to point communications using three sets of experiments, namely a ping-pong and a streaming evaluation and an evaluation of the power/throughput tradeoff for different MPI eager/rendezvous

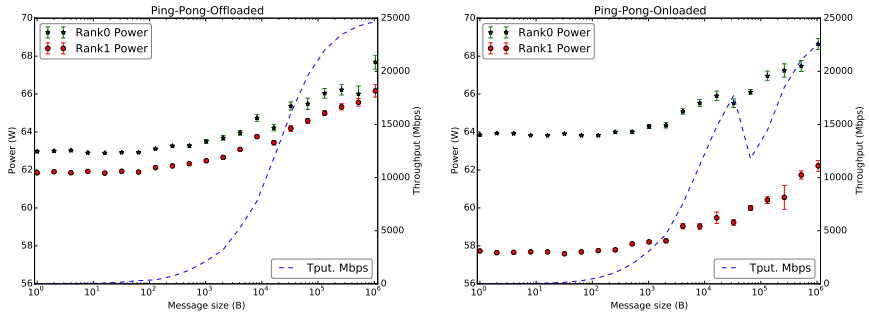



Fig. 3.1: Power and throughput measurements for ping-pong, power-of-two message sizes (1 B to 1MiB) on onloaded and offloaded HCA's.

transition points. For each experiment, we evaluate the power and performance across message sizes at power-of-two intervals, ranging from 1 byte to 1 megabyte in size. Each experiment is performed on both onloaded and offloaded HCA's.

For all tests, we divide the experiments such that each plot represents a fixed amount of data being sent over the network. For example, if a total of 5,000 messages are sent across the network at message size 1MiB, then a message size of 1KiB results in 5,120,000 messages. [1] Power measurements (described in Section 2) are collected during MPI_Send and MPI_Recv operations and averaged, to determine the power costs of point to point communication for each message size.

While NetPipe offers many interesting configurable options such as the ability to disable cache effects, the scope of the current paper did not allow an in depth exploration of all applicable options. However, exploring additional configurations will be considered in future work.

**3.2. Ping-Pong Results.** Figure 3.1 contains the results of sending 5 GB of data between two hosts in ping-pong style. Displayed in each plot, is the average power recorded at both Rank 0 and Rank 1, as well as the throughput recorded by NetPipe.

First, examining the results for the offloaded cards, it is clear that at lower message sizes, both the power and throughput are low with little growth until a message size of 128 Bytes. As the message sizes reach the Kilobyte range, we see a increased rate of growth in throughput and the corresponding power draw. As message sizes near 1 MiB in size, the growth in throughput begins to taper off. There are two additional points of interesting behavior in this plot. The first is at 16 KiB, where the power draw of the system decreases slightly as the throughput continues to increase. The cause of this shift in power is the eager to rendezvous transition that takes place in MPI, which for the nodes with offloaded cards takes place at 12K.

The results of the ping-pong test using onloaded cards is different from the onloaded results in several ways. At first glance one of the major differences appears to be the absolute power values of the Rank 1 process. However, this is not as significant as it seems. In the shift to running the experiments on onloaded cards, Rank 0 and Rank 1 are executed on different nodes entirely, such that the absolute

[1] The total amount of data transfer ed for each plot point can be calculated as 5000 × the largest message size of the plot.

power values across experiments should not be compared. As discussed in [6], this difference in power is within the range expected for manufacturing and temperature differences across different nodes. While a comparison between the absolute power values recorded across the two plots is inappropriate, the trends in power growth are comparable.

Another noticeable difference can be seen at the eager to rendezvous transition point – which is for the onloaded cards takes place at 64 KiB. At the transition point there is a sharp decrease to throughput on the onloaded cards, after which, power continues a steady growth. We discuss the effects of the eager to rendezvous transition point further in Section 3.4. In this work we did examine what subcomponents are responsible for the growth in power. For the onloaded nodes, approximately 60 percent of the increased power costs can be attributed to the CPU, with roughly 30 percent of the increased power costs stemming from memory. The remaining 10 percent is attributed to the NIC and rails that represent miscellaneous power costs of the motherboard.

Overall, for the ping-pong experiments, we see a slow increase to power as we increase the message size and enable higher levels of throughput. The only significant exception to this observation is for the onloaded cards at 64MiB. However these results do not tell the entire story. The nature of a ping-pong experiment means that much of the time spent on either end is in polling, as sender and receiver swap roles each iteration. In order to contrast this with a more communication intensive workload, in the next section we explore the power/throughput tradeoff in the scope of a streaming communication.

**3.3. Streaming Results.** Streaming tests provide insight about the power costs of a workload which rapidly sends data in a single direction. In our tests, the source node sends a continuous stream of data to the destination who provides a non-blocking receive.

With default settings, examination of the results for the onloaded card show a significant trade-off in throughput for small and large messages. At the low end throughput begins at 19 Mbps at a cost of 54 Watts. The increases to power are modest until we pass 128 bytes in size. One notable exception is in the range of 32 to 64 byte messages. At this point we see a decrease in power and an increase to throughput. Specifically, power goes to it's lowest point of our tests 52, and 48 Watts, respectively. The gains in throughput continue to grow doubling with the message size, from 390 to 730 Mbps. An application with modest bandwidth needs could optimize its message sizes accordingly.

For example if the total amount of data requiring transmission was 1024 bytes, this could be split into $16 \times 64$ byte sub-messages. By doing so the application could reduce it's power costs by 14 Watts or 23% of total system power. While, there is a substantial cap placed on the throughput of the node, The other cost to this power savings comes from latency of the network. While smaller messages have reduced overheads and lower latency, this strategy requires that we send more of them. In the case of this example, NetPipe reported a latency of 1.7 $\mu$s for the 64 byte message. And 2.5 $\mu$s for the 1024 byte message. An upper bound on the total latency for splitting the messages would be $16 \times 1.7 = 27.2\mu$s. However, this upper bound greatly overestimates the costs since there is significant overlap when sending a series of messages. Analyzing the results of our tests show that a closer estimate would be 2.5 $\mu$s for the 1024 byte message and 4.7 $\mu$s for the series of 16 smaller messages.

From 128 bytes to 4096 bytes the throughput rises rapidly while power increases

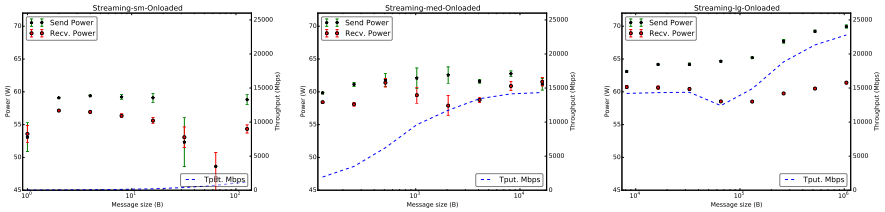



Fig. 3.2: Power and throughput measurements for streaming, power-of-two message sizes (1 B to 1MiB) on onloaded HCA's.

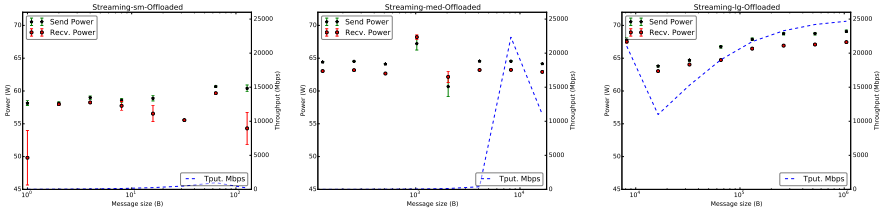



Fig. 3.3: Power and throughput measurements for streaming, power-of-two message sizes (1 B to 1MiB) on offloaded HCA's.

at a slower rate. Afterwards power and throughput remain relatively flat. The eager/rendezvous transition point is reached at 65536 bytes and a shift in power and throughput is apparent. At the transition point, throughput is briefly reduced, with a reduction in power on the receiving end. Immediately following this point, throughput rises rapidly. While both sender's and receiver's power increase, the rate of increase is greater at the sender.

In our streaming tests, the offloaded cards present less opportunity for worthwhile tradeoffs among power and throughput, than their onloaded counterparts. At lower message sizes throughput slowly increases and then drops for messages sizes exceeding 128 bytes. Past 128 bytes, there is a power increase in the 5 Watt range and then temporary 5 Watt increase in power, observed for message sizes of 1024 bytes.

When comparing onloaded and offloaded cards, there are several distinguishing features of the offloaded cards. Firstly, the offloaded cards see a large shift in throughput and power for message sizes of 8KiB. Additionally, the offloaded cards see a decrease in power on both the sending and receiving node at the eager/rendezvous transition point.

### 3.4. Eager to Rendezvous Transition Point.

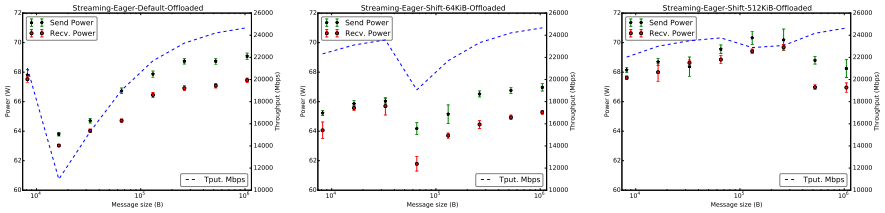



Fig. 3.4: The effect of changing the eager/rendezvous transition point default(12K), 64KiB and 512KiB, for offloaded HCA's.

**4. A power centric evaluation of AllReduce and AllGather.** In Section 3 we examined the effect different message sizes had on power of a system utilizing point to point communication. Using this knowledge as a foundation, in this section we explore the power and performance tradeoffs of an MPI collective. For this work we focus on the collectives AllReduce and AllGather.

We focus on how power changes while adjusting these collectives in two ways. Firstly, like Section 3, we explore the collectives with respect to varying message size. Secondly, we look at how different underlying algorithms for the collectives could effect the power and performance. Specifically the algorithms we examine are *linear* and *logarithmic* algorithm for AllReduce and a textitlinear and *recursive doubling* algorithm.

Here, we give a brief description of how the algorithms differ for the AllGather collective and refer the reader to [12] for additional details. The linear algorithm is a simple algorithm with poor scaling characteristics. The approach of this algorithm is for each node to send their original send buffer directly to the rank 0 node. The rank 0 node then concatenates the data from each node, before sending the final result back to all $n-1$ nodes. This results in a wave of $n-1$ small messages being sent to rank 0, followed by final wave of $n-1$, sizeof(send_buffer) $\times n$ being sent from rank 0. For a large number $n$, the bottleneck at rank 0 becomes problematic.

In contrast the recursive doubling algorithm, as described in [], each node begins sending it's original send buffer to it's nearest ranking node. For the additional, $lg(n)-1$ iterations, each node swaps data with another node from the remaining set of uncontacted nodes. [2] In each of the iterations, the amount of data swapped doubles, reaching a maximum of sizeof(send_buffer)$\times\frac{n}{2}$ in the final iteration.

While both algorithms provide the same end-result, the linear algorithm's maximum message size is twice that of the recursive doubling algorithm. Additionally the linear algorithm results in a larger number of messages overall. For a given $N$ nodes and an initial send buffer of $k$ data, the amount of data sent over the network for each algorithm is:

**Recursive Doubling** $kn(2^{\lg n}-1)$
**Linear** $kn+kn^2$

**4.1. Experiment Design.** In order to test the power and performance of the collectives, we utilize an allocation of 64 nodes on the Teller cluster. On these 64 nodes we performed 5,000 iterations of each collective. For AllReduce, our initial

[2] A node is considered uncontacted, if it, nor any of its previously communicated nodes have been contacted.

data size scales up to 1KiB in size For AllGather, the initial send buffer on each node ranged from 4 bytes to 1MiB of unique data. During each run, the power is measured on as described in Section 2. Afterwards we take the per-node averages of power for every node. For all of the plots representing our collective experiments, we plot the minimum, the maximum, and the average of the per-node averages. Additionally, we plot the average time taken for the completion of a single collective operation. The results of these experiments show the power and time trade-off for very different algorithms.

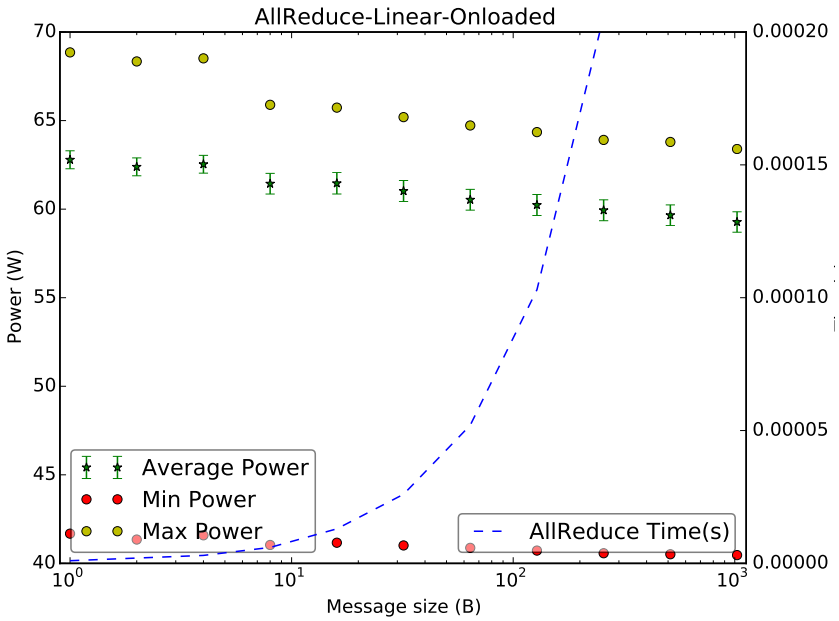



Fig. 4.1: Power and completion time for linear AllReduce for varying message size.

**4.2. AllReduce Results.** Figure 4.1 and Figure 4.2 show very little difference in power costs of AllReduce tests, relative to the algorithm selected. In fact, we see little to no shift in power, even as the time to perform an AllReduce operation ramps up. As expected logarithmic AllReduce scales significantly better than the linear AllReduce. These tests only perform reductions on relatively small messages – going up to 1024 bytes. This covers the range of message buffers expected in a typical HPC application.

**4.3. AllGather Results.**

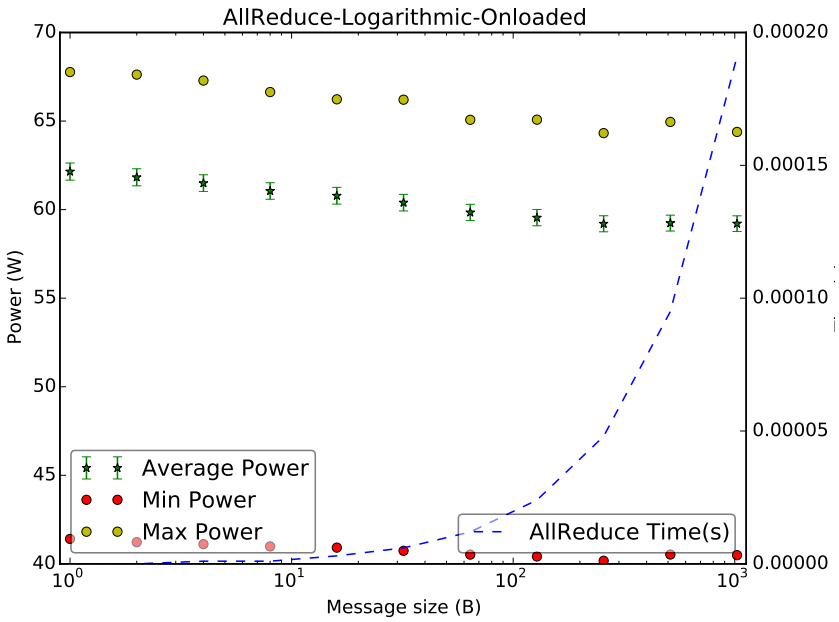



Fig. 4.2: Power and completion time for logarithmic AllReduce for varying message size.

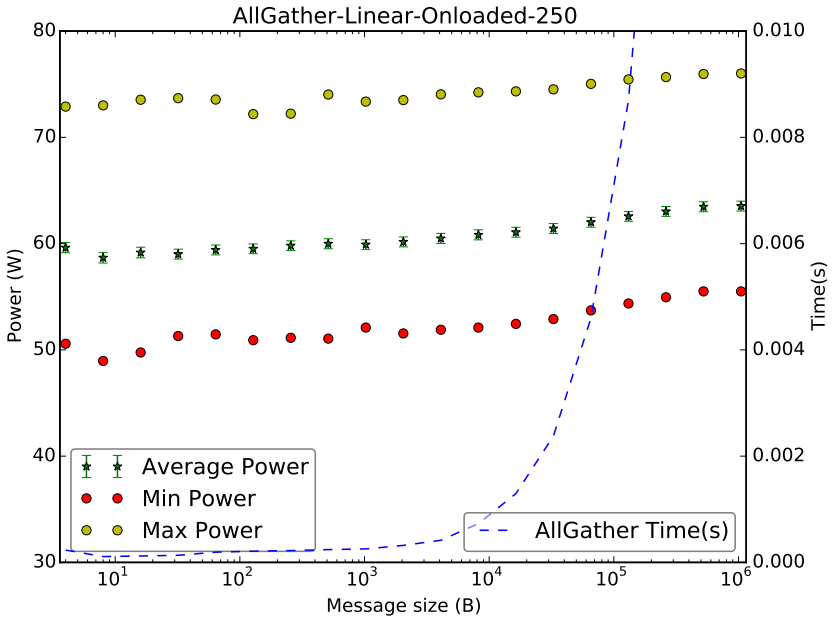



Fig. 4.3: Power and completion time for linear AllGather for varying message size.

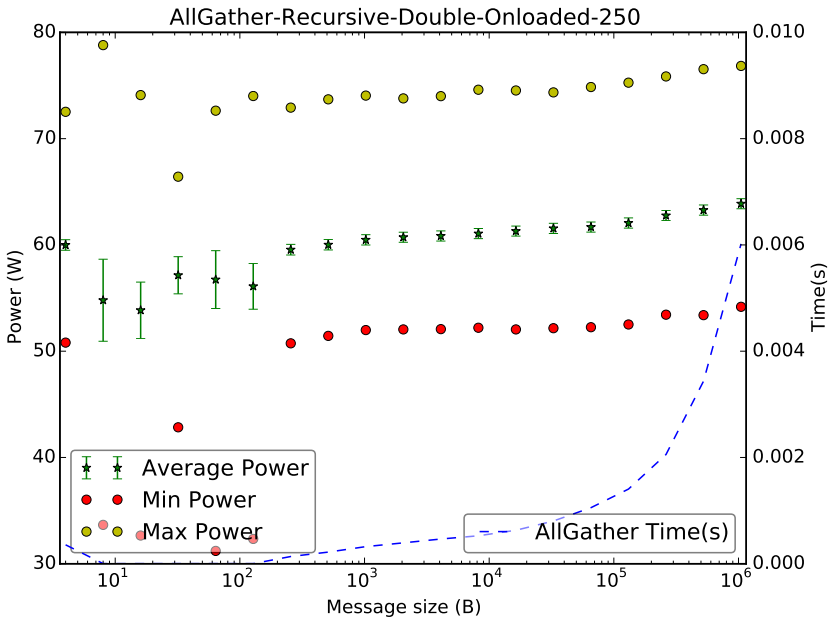



Fig. 4.4: Power and completion time for recursive-doubling AllGather for varying message size.

In an effort to determine if negative results from the AllReduce Experiment were indicative of a broader range of collective operations, our second experiments examined the power costs of different AllGather algorithms. AllGather collects a fixed size buffer from each of the collective participants, merging them together until each participant has gathered the data from all other participants. AllGather makes a good candidate for exploring power tradeoffs since it works with a variety of message sizes. In each iteration the block of data sent from the jth process is received by every process and placed in the jth block of the buffer recvbuf, per the MPI standard. However, Figure 4.3 and Figure 4.4 show very little difference in power costs of AllGather tests, relative to the algorithm selected. In fact, we see little to no shift in power, even as the time to perform an AllGather operation ramps up. As expected recursive doubling AllGather scales significantly better than the linear AllGather.

The larger question is why is the power cost of these collective operations so stable when performance is so different across algorithms? Our hypothesis is that this is the result of polling within the MPI collective. In each phase of the collective nodes spend a large portion of time polling or idle waiting for the slowest participant of the collective to complete. The amount of time spent waiting on messages or other nodes results in relatively flat power costs when compared to the set of streaming point to point messages seen in Section 3. Verifying this hypothesis is designated as future work.

**5. Related Work.** Authors have explored the performance of MPI operations in a variety of publications [8, 11]. This earlier work did not reflect the power and energy constraints expected on future systems.

There has been work exploring power and energy savings on systems running high performance, MPI-based applications [3, 5, 2, 7]. However, the majority of these focus on finding power savings by adjusting the frequency/voltage of the system directly rather than examining differences relative to message sizes or collective algorithms. Other approaches have looked at scheduling policy to manage power consumption of a system [9].

**6. Conclusions.** In conclusion we found that there are some power savings to be found in point to point communications. The most noticeable changes in power occurring in the transition from eager to rendezvous messages. However, these power savings are offset by a tremendous cost to potential bandwidth and a significant cost to latency. Furthermore we did not see a substantial difference in power across the AllReduce and AllGather algorithms that we tested on our system. As future systems shift towards many-core architectures, operating at reduced power, this topic may merit further examination.

REFERENCES

[1] S. Ashby, P. Beckman, J. Chen, P. Colella, B. Collins, D. Crawford, J. Dongarra, D. Kothe, R. Lusk, P. Messina, et al., *The opportunities and challenges of exascale computing*, Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee (November 2010), (2010).

[2] Y. Dong, J. Chen, X. Yang, C. Yang, and L. Peng, *Low power optimization for mpi collective operations*, in Young Computer Scientists, 2008. ICYCS 2008. The 9th International Conference for, Nov 2008, pp. 1047–1052.

[3] V. W. Freeh and D. K. Lowenthal, *Using multiple energy gears in mpi programs on a power-scalable cluster*, in Proceedings of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '05, New York, NY, USA, 2005, ACM, pp. 164–173.

[4] *The Green 500*. `http://www.green500.org/` (visited August 2014).

[5] K. Kandalla, E. Mancini, S. Sur, and D. Panda, *Designing power-aware collective communication algorithms for infiniband clusters*, in Parallel Processing (ICPP), 2010 39th International Conference on, Sept 2010, pp. 218–227.

[6] J. H. Laros, D. DeBonis, and P. Pokorny, *PowerInsight - A Commodity Power Measurement Capability.*, Apr 2013.

[7] D. Li, B. R. de Supinski, M. Schulz, K. Cameron, and D. S. Nikolopoulos, *Hybrid mpi/openmp power-aware computing*, in Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on, IEEE, 2010, pp. 1–12.

[8] J. Pješivac-Grbović, T. Angskun, G. Bosilca, G. E. Fagg, E. Gabriel, and J. J. Dongarra, *Performance analysis of mpi collective operations*, Cluster Computing, 10 (2007), pp. 127–143.

[9] B. Rountree, D. Lowenthal, S. Funk, V. W. Freeh, B. de Supinski, and M. Schulz, *Bounding energy consumption in large-scale mpi programs*, in Supercomputing, 2007. SC '07. Proceedings of the 2007 ACM/IEEE Conference on, Nov 2007, pp. 1–9.

[10] Q. O. Snell, A. R. Mikler, and J. L. Gustafson, *Netpipe: A network protocol independent performance evaluator*, in IASTED International Conference on Intelligent Information Management and Systems, vol. 6, Washington, DC, USA), 1996.

[11] R. Thakur and W. D. Gropp, *Improving the performance of collective operations in mpich*, in Recent Advances in Parallel Virtual Machine and Message Passing Interface, Springer, 2003, pp. 257–267.

[12] R. Thakur, R. Rabenseifner, and W. Gropp, *Optimization of collective communication operations in mpich*, International Journal of High Performance Computing Applications, 19 (2005), pp. 49–66.

[13] *Top 500 Supercomputer Sites*. `http://www.top500.org/` (visited August 2014).

# DELTA: DATA REDUCTION FOR INTEGRATED APPLICATION WORKFLOWS

GREGORY JEAN-BAPTISTE[‡] AND GERALD LOFSTEAD[§]

**Abstract.** Integrated Application Workflows (IAWs) require high frequency and high volume data transfers between compute nodes and staging area machines during the lifetime of a large, distributed computation. The available network bandwidth between the two areas may not be enough to efficiently support the data movement. As the processing power available to compute resources increases, the requirements for this data transfer will become more difficult to satisfy and perhaps will not be satisfiable at all, since network capablilites are not expanding at a comparable rate. Furthermore, energy consumption in HPC environments are expected to grow by an order of magnitude as exscale systems become a reality. The energy cost of moving large amounts of data frequently will contribute to this issue. It is necessary to reduce the volume of data without reducing the quality of data when it is being processed and analyzed. Delta attempts to resolve the issue by removing multiple copies of the same data during transfers and restoring those copies once the data has reached the staging area. Delta is able to identify duplicated information and determine the most space efficient way to represent it.

**1. Introduction.** Scientific computing at a large scale has driven trends in supercomputing including data parallelism, distributed computing and fault tolerance. That is because scientific applications are becoming increasingly complex as developers and researchers attempt to solve more difficult and varying problems. Through simulation and data analysis, researchers are able to model their use cases more accurately and gather more meaningful information as a result. Not only are these models complex, but they are also increasing in size. The underlying hardware and software had to evolve to accomidate applications that require such capabilites and they are continuing to improve. In the past, users of such systems would have wait until the applications were finished running before the data could be analyzed or presented in a more intuitive format. At the time, the output from the application would be moved into persistent storage over the course of a run. Integrated Application Workflows (IAWs) refers to the practice of moving the data to an intermediate staging area before being placed in persistent storage. In the staging area, users can process, analyze and visualize the data as the application is still running, leading to greater insights and more detailed understanding. IAWs also avoid writing intermediate, transient data to slower persistent storage during the computation. Instead, various methods are employed to move the required data to where it is needed or making it quickly accessable remotely.

IAWs are becoming more popular, as it is appealing to have realtime insights during the course of a simulation. However, the amount of relevant data produced during the run of such an application is quite large and is getting larger as the computational capabilites of supercomputers approaches exascale. Transferring such a high volume of data repeatedly during runtime can have a severely negative impact on the network, where the improvement in bandwidth cannot match the growing data size. The resulting backlogging could affect performance on both the compute area and the staging area. Compounding the issue is the amount of energy consumed over the course of such an application, including the cost of transferring data. Currently, computation cost dominates energy usage but if the data volume grows at the same rate, the cost of moving data can easily become the limiting factor when it comes to

[‡]Florida International University, gjean011@fiu.edu
[§]Sandia National Laboratories, gflofst@sandia.gov

operating these systems. One way to deal with this issue is to understand how most scientific applications work. After an initial setup, these applications run in loops that represent individual timesteps in the simulation. Usually, the same type of data is produced every time (for example, temperature, velocity, position, etc.), but the value may or may not change from timestep to timestep. An application developer could adjust their program such that if a particular variable does not change between timesteps, it is not transfered to the staging area. When the analysis software does not receive the variable, it can assume that it has not changed from the previous timestep and use the value that was obtained previously. An even better (and easier for application developers) solution would be an underlying system that could do this for any application running on top of it.

ADIOS is the implementation of such a system. It removes the responsibility of transporting data from the application developer and handles all the details underneath. The developer only needs to define which variables are to be transported and which transport method to use. Delta is a prototype transport method that does what was previously described: using the variable definitions described by the user, Delta only transports data that has changed between timesteps. Delta can also read data in at the staging area and rebuild what is missing based on data from previous timesteps. The user can transport data and access it on the other side, leaving Delta to handle the packing and unpacking that occurs in between. Delta works even when there are multiple nodes (as expected) participating in the computation and producing output. Figure 1.1 illustrates the architecture of IAWs using ADIOS and Delta to handle the data transportation.

In order to motivate this work, two separate scientific applications were modified to measure the change of output between timesteps. The first, LAMMPS can simulate a number of interactions and events at a fine level. The user can define the interaction in an application specific manner and let it run over a number of timesteps. Here, the modified code was run with and example called “crack”, which simulates a crack propagating through some solid. Roughly 40% of the variables changed between timesteps, getting as high as 60% at times. Figure 1.2 displays the changes aggregated over time. Another example, which simulated a melting solid had 60% of its variables changing between timesteps, getting as high as 75%. Similar results were found when running an exmaple built on DEALII, another scientific simulator. It is easy to see that this strategy will work better with some simulations that others. Simulations that have some form of propagation (such as a crack or melting) would have many of its parts remain static for some period of time. Other simulations even have parts that don’t change at all (such as the ground over which a liquid is flowing). These kinds of simulations would benefit the most from applying Delta or an application specific version. Since ADIOS can be used to support such applications, Delta aims to be an application independent version.

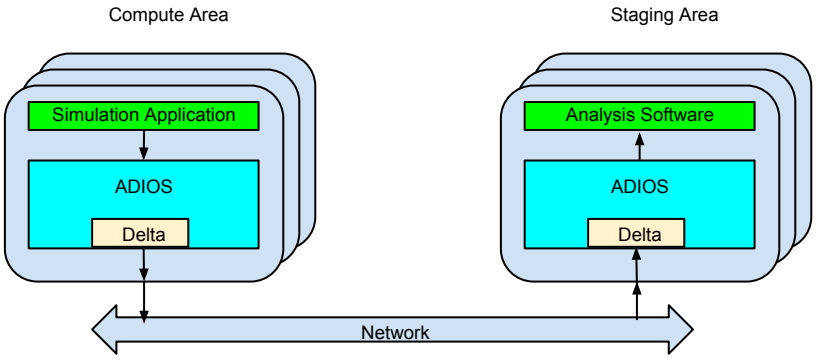



Fig. 1.1: A diagram of the IAW architecture, including ADIOS and Delta

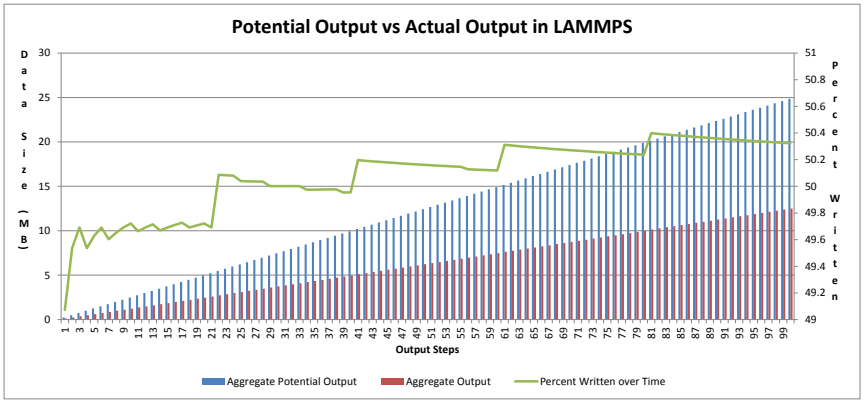



Fig. 1.2: A graph showing the pattern of reduced writes in LAMMPS for the "Crack" example

**2. Related Work.** Improving workflows and workflow managment for scientific computation has been the focus of several projects. Many scientific works involve multiple components in order to get the desired results. Managing these pieces can be difficult and confusing because of the variety of settings, platforms, data formats and tools. A knowledgable user can create their own scripts using Python for example, to manage the ordering of the necessary components and the intermediate procedures such as moving data. On the other hand, there are workload management systems specifically designed for scientific applications. Pegasus simplifies the operation of such applications by mapping high level descriptions of a workflow to the available computational resources. The user does not need to have a deep understand the underlying infrastructure, as Pegasus will automatically locate and allocate them according to the input provided by the user. Pegasus has an array of features, including data replication and transfer. Similar to Pegasus is Kepler, which also provides the user with a graphical interface, allowing them to define a workflow visually while Kepler takes care of the details of execution. While useful, these managers, like many others are offline, which in this case means that slower, centralized storage is used to hold intermediate data and provide access to various components. There are online systems that instead use faster storage mechanisms (direct transfer, staging area, etc.) for data storage and movement operations. Zookeeper for example keeps copies of the relevant data distributed across the main memory of several servers which are eventually consistent. The focus of this paper is a workflow that employs staging areas for intermediate data, but the principles described are not limited to that case.

**3. Actual Content.** In order to reduce data transferred over the network, Delta must determine how much data is changed between every round of the running computation. Each node in the system keeps track of the full set of output from the previous round. When the current round completes, the new output is compared to the previous output. During the first round, there is no comparison because there is no old output to compare with. Every subsequent round compares each variable against its matching predecessor. This includes both scalars and vectors. Delta calculates the difference between the rounds. Anything that has not changed is not included in the payload, and is instead replaced by metadata to describe what is left out. Once the payload arrives at its destination, it can be determined through the metadata what is missing. Whatever it is that is missing can be found in a previous payload. If the change is too small, then the cost of the metadata summed with the changed data will be greater than the cost of just sending the data plainly. Therefore, if the amount of change is below a certain threshold, all the data is sent as is.

Packing the data for transport requires several steps. At top of each payload, certain pieces of information are required.

Group ID: The identifier for the group that was designated in the adios_open() call. In ADIOS, a group is a set of variables that are transported with the same transport method. A group can consist of both scalar and vector variables, each with different data types and sizes. A single application can consist of many groups, each possibly using a different transport method.

Epoch: The epoch is the identifier of the current computation round. This value is written once per payload, regardless of the number of variables.

Current Rank: The rank of the process preparing to write the current payload. This is written once per payload.

Group Name Length: The reader will need to know the length of the name of the current group to read it properly.

Group Name: The name of the group involved in the current payload is required. This value is only written once throughout the entire computation as well as its length. After this, the group id will be sufficient to identify the group.

Variable Count: The number of variables belonging to the group that was designated during the adios_open() call. This is written only once during the length the the operation.

The previously mentioned parameters are written once during each adios_open-adios_close cycle except for the variable count of a previously written group. Variable counts will not change during the course of a full operation. After this section, the individual variables for the designated group are added. Each variable has two required components:

Status: The status has a value of NONE, SOME or NEW. If the status is SOME, there was a change in the variable between the rounds. If the status is NONE, there was no change and no data should be sent besides the required metadata. If the status is NEW, that means that the variable in question is being written for the first time and all the necessary metadata will be sent along with it.

Id: The variable id number used to identify the variable once it is delivered by the receiver.

If the status is NONE, then only these two fields are sent as metadata for a particular variable. The actual data is exactly the same as the previous round. If the status is SOME, then more data is required, but since this variable was previously written, some pieces of information can be left out. Otherwise, the variable is new and will be padded accordingly.

Name Length: The length of the variable name, needed to read in the name when the variable is being unpacked. This is only included for new variables being written for the first time.

Name: The name of the variable, used to identify the variable in the user application. This is also only required for first time variables. During subsequent rounds, the Id will be a sufficient identifier for the variable.

Dim Count: The number of dimensions for the variable. If the variable is a scalar, this value is 0.

Global: A Boolean value that signals whether or not the variable is global. A global array has its dimensions and values split between the processes involved in the computation. The way the array is divided in user defined, and it is up to the reader method to corretly identify an individual piece should it be requested later on.

Dim Sizes: If the variable is a vector, then the size of each dimension must be included in the metadata. If the vector is global, then for each dimension, three pieces of information are necessary. First, the global value represents the total size of the dimension across every involved process. Second, the local value represents the starting point in the current dimension for the current process. Finally, the offset represents the amount of entries in the dimension that belong to the current process, starting from the local value.

Type: This is an integer that represents the data type. The type only needs to be written the first time the variable is encountered.

Type Size: This is the size of the specified data type. This information is also only required the first time.

Next, it must be determined whether all of the data should be sent, or if a portion should be left out with some metadata that describes it. If there are no copies of the variable, then all of the information is sent. If the variable is a scalar, this is not an

issue. If a scalar changed, it must be sent. If it did not change, it is not sent. Vectors present a challenge. Delta represents a vector variable using a bit vector. The bit vector is set to the size of the full output from the round. For example, if the variable is question was an array with 100 elements, the bit vector would consist of 100 bits, etc. Each element in the output is compared to an element in the same position in the previous round. If it is the same as before, the value of the matching bit is set to 0 and the element is excluded from the payload. Otherwise, the bit is set to 1 and the data remains in the payload. If the new output vector is larger than the last, then the overflow is all represented by 1s. All of the extra new data is included in the payload. The output from the last round is then replaced by the current output. Next, the size of the current rounds output is compared to the size of the reduced payload plus the bit vector. If the reduction summed with the bit vector is larger than the original output size, then the bit vector is discarded and all of the data is sent. Otherwise, the reduction and the bit vector are prepared for transport. Here are the remaining metadata fields:

Total Size: The total size of the payload data (number of elements multiplied by the type size).

Data: The actual payload, whether it is the full output or the reduced output.

Bit Count: The number of bits in the bit vector. If this is 0, then there is no bit vector, meaning that it was not worth it to reduce the output or all of the data was changed from the last round (or, as a special case, it is the first round). The bit count also tells the size of the original output, for unpacking purposes.

Bit Vector: The actual bit vector. If the bit count was 0, then the bit vector is not included in the metadata.

This is the format for every variable in every group in the package. When complete, the package is written to whatever the destination is. Figure 3.1 illustrates a full data set with all of the associated metadata included. Figure 3.2 shows the same data set in the following epoch. Some of its data values have changed, so a bit vector has been added to describe the change. Much of the metadata is no longer necessary at that point since it is unchanging and has already been written previously.

In order to unpack the payload, there are a set of data structures to manage the parts of the payload. At the top is the Delta_Data_Struct, which keeps track of two different types of information: The information that can change from epoch to epoch, and the information that stays the same.

For the changing information, The Delta_Data_Struct (or DDS) keeps a linked list of Group_Struct structures. Each Group_Struct represents a different group and contains a linked list of Epoch structures. An Epoch structure represents a single timestep in the associated groups lifespan. It also keeps track of the process (rank) that submitted the group data during that timestep. Finally, each Epoch has a vector of Var_Struct structures that hold the state of each variable during that particular epoch such as its value and size (if it is a vector, for example).

For the unchanging information, the DDS has a linked list of Group_Record structures that hold the group name and the number of variables for that group since those don't change. Also, each contains a vector of Variable_Record structures that hold information such as the variable name, data type, and whether or not it is global.

Unpacking begins at the top of the payload. First, the group id is read in, followed by the epoch. The DDS keeps track of the current group and epoch. If the current epoch is 1, that means that the current group is being seen for the first time. The Group_Record and Group_Struct are initialized at that point. The name length,

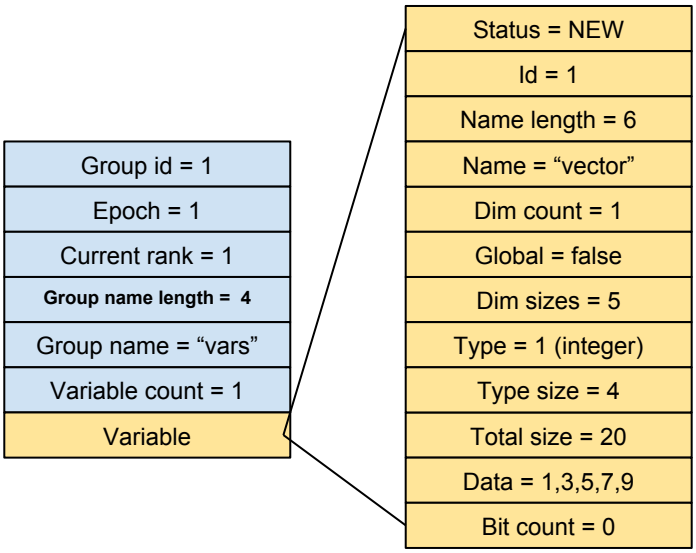



Fig. 3.1: A diagram depicting the structure of a full payload of data

group name and variable count are read in and saved in the Group_Record. These will never be changed.

Next in line is the rank, which is then used to initialize the current Epoch for the current group. The DDS also keeps track of the current rank. Once all of this is done, the current group, rank, epoch and the number of possible incoming variables is known. The reader can now loop through the remainder of file, extracting the information for each variable and filling in its data structures.

First, the variable id and its status are read from the file. If the status is NONE, that means that the variable was same as from the previous epoch. The reader refers to the previous epoch for the variable in the new epoch. If the status is NEW, the name length and the name come next, since this is a variable that has never been seen before. Otherwise, the number of dimensions is read. If the number of dimensions is 0 (scalar variable), the reader skips reading anything dimension related. Otherwise, there is some extra work. If the status was NEW, the reader does not know if the variable is global or not. It reads that and stores the result once, since that will not change throughout the computation. Based on the dimension count, the reader knows how much of the file to scan to get the size of each dimension, whether it is global or not. Next, if the status was NEW, the reader scans the type and type size of the variable, information that also does not change. Otherwise, the total size, the actual data and the bit count are read. If the bit count is 0, that means all of the data was in the payload and no extra work is needed. If the bit count is greater, then some addition unpacking is necessary.

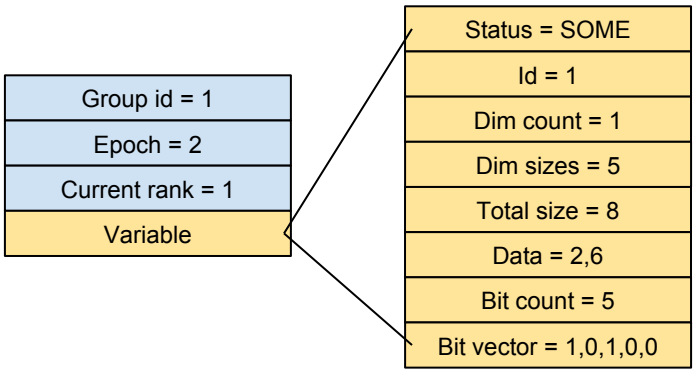



Fig. 3.2: A reduced version of the first depiction, in a subsequent epoch

In the case of having a bit count larger than 0, the reader will scan in the bit vector from the file. An element pointer will be set to the top of the partial data read from the payload. Next, a vector of the variable type equivalent to the size of the bit vector will be created. This will be the full, unpacked data vector. Then, the reader will step through each element is the bit vector. If the value is 1, then the data is in the partial vector at the spot pointed to by the element pointer set earlier. The data is copied into the new vector and the element pointer is incremented. If the value is 0, the data is in the previous epoch, at the position equal to the readers current position in the bit vector. That data is copied into the new vector. When the bit vector ends, the full new data vector has all of the correct data and is saved.

All of these steps are repeated for each variable in the file. Once complete, the payload is saved in the appropriate data structures for retrieval by the application running on top of ADIOS.

**4. Conclusions.** As supercomputers continue gaining ground on the road to exascale, many of the currently acceptable practices when it comes to managing data will become obsolete because of the resulting bottlenecks and the associated energy consumption. One of these prectices involves taking up bandwidth to transfer information that was already previously transmitted and already exists at the destination. Delta makes an attempt to prevent that by keeping track of changes between timesteps in a computation in order to detect stagnant data. By blocking the transfer of such data, the extra bandwidth that would have been consumed is now available to the rest of the application or not used at all. In that process, no data is lost to the staging area

for IAWs. The reader in Delta can rebuild the reduced data using previously obtained information, giving a complete picture of the output produced during computation.