CSRI SUMMER PROCEEDINGS 2013

The **C**omputer **S**cience **R**esearch **I**nstitute
at Sandia National Laboratories

Editors:

Sivasankaran Rajamanickam
Michael L. Parks
S. Scott Collis
Sandia National Laboratories

July 22, 2014

SAND2014-20409 R

Sandia National Laboratories is a multiprogram laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the DOE's National Nuclear Security Administration under contract DE-AC04-94AL85000.

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

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

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

Available to DOE and DOE contractors from

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

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

Available to the public from

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

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

Preface

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

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

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

For the 2013 CSRI Proceedings, research articles have been organized into the following broad technical focus areas — *Computational Mathematics and Algorithms, Combinatorial Algorithms and Visualization, Advanced Architectures and Systems Software, Computational Applications* — which are well aligned with Sandia's strategic thrusts in computer and information sciences.

We would like to thank all participants who have contributed to the outstanding technical accomplishments of CSRI in 2013 as documented by the high quality articles in this proceedings. The success of CSRI hinged on the hard work of 19 enthusiastic student collaborators and their dedicated Sandia technical staff mentors. It is truly impressive that the research described herein occurred primarily over a three month period of intensive collaboration.

CSRI benefited from the administrative help of Amy Levan, Phyllis Rutka, Denise Laporte, and Bernadette Watts. The success of the CSRI is, in large part, due to their dedication and care, which are much appreciated. We would also like to thank those who reviewed articles for this proceedings — their feedback is an important part of the research training process and has significantly improved the quality of the papers herein.

Sivasankaran Rajamanickam
Michael L. Parks
S. Scott Collis

July 22, 2014

Table of Contents

Computational Mathematics and Algorithms

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

Deweese and Boman discuss preconditioners for solving linear systems arising from graph Lapacians and present comparisons against traditional preconditioners. *Elliott and Hoemmen* quantify the impact of a bit-flip in GMRES. They use a fault model to either bound the fault or to detect it easily. *Osborn and Phipps* investigate the performance of linear solves on hybrid CPU and GPU architectures and show promising results for solving linear systems in multiple GPU systems. *Zhou et al.* present a new algebraic fast H-matrix solver. Typically, these solvers use geometric information. This present work extends to algebraic problems and shows scalability results. *Shank et al.* extend the KKT preconditioning strategy to the indefinite Helmholtz case. They show that this strategy is insensitive to mesh resolution and modestly sensitive to frequency changes. *Bond et al.* introduce a mesh-free Galerkin method for solving non-local diffusion problems by using a quadratic scheme specific to radial basis functions. They show results on 1D and 2D problems. *Yang et al.* detail an implementation of implicit smoothed particle hydrodynamics package in LAMMPS and use Trilinos solvers for linear solves. They test their implementations with a Taylor-Green vortex example and present results.

S. Rajamanickam
M.L. Parks
S.S. Collis

July 22, 2014

A COMPARISON OF PRECONDITIONERS FOR SOLVING LINEAR SYSTEMS ARISING FROM GRAPH LAPLACIANS

KEVIN DEWEESE* AND ERIK G. BOMAN[1]

Abstract. We consider the solution of linear systems corresponding to the combinatorial and normalized graph Laplacians of large unstructured networks. A promising approach to solving these problems is to use a class of support tree preconditioners. We previously implemented such a preconditioner in Trilinos in serial using the Epetra software stack. This work extends that implementation to run in parallel on distributed memory systems and migrates the implementation to the Tpetra software stack to help with future development. This preconditioner is compared against the other preconditioners currently available in Trilinos.

1. Introduction. Networks play an important role in many application areas, for example engineering, social sciences, and biology [8]. We focus on networks that are large and unstructured. Several analysis techniques rely on solving linear systems and eigensystems of graph Laplacians such as random walks [4] and Katz centrality scores [6] using linear solvers and graph partitioning [10] and clustering [11] using eigensolvers. These solvers can be very compute intensive tasks but preconditioners can be used to dramatically reduce the solution time. Although there are many good preconditioners for PDEs, they are generally not suited for graph Laplacians from highly irregular graphs or scale-free networks. Several graph based preconditioners with strong theoretical results have been developed over the past decade [7, 9]. However these are difficult to implement. We instead seek to better understand the support tree preconditioner first proposed by Vaidya [1] and how it compares to other preconditioners on these graph problems.

1.1. Background. The combinatorial Laplacian of a graph G is given by

$$L_G = D - A_G$$

where A_G is the adjacency matrix of G and D is the diagonal matrix containing the sum of adjacent edge weights, or in the unweighted case just the vertex degree. A special scaling of this matrix called the normalized Laplacian is given by

$$N_G = D^{-1/2} L_G D^{-1/2}.$$

Both L_G and N_G are positive-semidefinite and diagonally dominant. An interesting note is that solving $L_G x = b$ can be done indirectly by solving $N_G x' = b'$. We can see this by setting $x' = D^{1/2}x$ and $b' = D^{-1/2}b$ yielding the following.

$$N_G x' = b'$$

$$D^{-1/2} L_G D^{-1/2} D^{1/2} x = D^{-1/2} b$$

$$D^{1/2} D^{-1/2} L_G x = b$$

*UC Santa Barbara Dept. of Computer Science, kdeweese@cs.ucsb.edu

[1]Sandia National Laboratories, egboman@sandia.gov. Sandia 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.

$$L_G x = b$$

Thus if solving one system is faster than the other, perhaps the normalized Laplacian as it is typically better conditioned, then it could be solved and the solution converted. However the eigenproblems must be solved separately as each Laplacian has a different spectra.

1.2. Preconditioners. A good preconditioner M of a matrix A should reduce the number of iterations of the preconditioned system $M^{-1}A$. In other words M should be a good inverse approximation of A. In addition solving the system $Mw = y$ should be much easier to solve than $Ax = b$ as it will be solved at every iteration. The first constraint prompts us to bound the condition number of $M^{-1}A$ while the second constraint requires us to bound the fill in the triangular factors of the preconditioner. Assuming a complete Cholesky factorization is used these factors will be of the form $M = CC^T$ and will be used to quickly solve $CC^T w = y$. Perhaps the most simple preconditioner is the Jacobi method which sets $M = D$ where D is the diagonal of A. While not a very good inverse approximation this preconditioner is very cheap to apply at every iteration. A slightly better inverse approximation could be used such as the symmetric Gauss-Seidel preconditioner which decomposes the input matrix into triangular parts $A = L + D + U$ and uses $M = (D+L)D^{-1}(D+U)$. Another popular preconditioning technique is to use an incomplete Cholesky factorization to approximately factor the matrix $A \simeq M = \widetilde{C}\widetilde{C}^T$ by dropping some entries during the factorization of A. The first of a class of support graph preconditioners was proposed by Vaidya which finds a maximum-weight spanning tree of the graph of A and uses this as a preconditioner. This preconditioner has condition number $O(nm)$ which bounds the number of iterations of the preconditioned system to $O(\sqrt{nm})$, where n is the dimension of the matrix (number of vertices) and m is the number of non-zero entries of the matrix (number of edges). However these are worst case bounds and typically the number of iterations required is much less. Since the preconditioner corresponds to a tree it can be factored with no fill.

2. Software. A version of Vaidya's preconditioner was previously implemented inside the Ifpack package of Trilinos [5]. Ifpack uses Epetra linear algebra primitives as opposed to the more recent, templated Tpetra primitives. Tpetra allows arbitrary scalar data types and arbitrary index types. Most future Trilinos development will focus on packages using this new Tpetra stack. For these reasons we decided to migrate all future work concerning support graph preconditioners to this new Tpetra software stack. The packages relevant to our work can be seen in Figure 2.1.

Feature	Old Stack	New Stack
Core	Epetra	Tpetra
Sparse Direct	Amesos	Amesos2
Preconditioners	Ifpack	Ifpack2
Partitioning and Ordering	Zoltan	Zoltan2

FIG. 2.1. *Trilinos software stack*

To accomplish this migration and improve upon the previous support graph implementation a few pieces of software were added to the Trilinos software library. Previously Trilinos did not contain a sparse Cholesky solver. Using the Amesos2 adapter package an interface was added to the CHOLMOD package [2]. Additionally a support graph preconditioner was added to the Ifpack2 preconditioner package which creates the support graph and calls the CHOLMOD interface to perform a complete factorization. Furthermore a bug in Ifpack2's Additive Schwarz domain decomposition class was corrected so that it could be used with the support graph preconditioner.

3. Experimental. Experiments were run to solve preconditioned linear systems with random right hand sides using the conjugate gradient solver in the Belos linear solver package. Experiments were run on 4 different graphs from the University of Florida sparse matrix collection [3] shown in Figure 3.1. The first 3 graphs in this table are network graphs and experiments were run using both the combinatorial Laplacian and normalized Laplacian matrices of these graphs. An important note is that these graphs are all unweighted so the support graph preconditioner of the combinatorial Laplacian is simply a random spanning tree. The last graph is a stiffness matrix problem included to see how a support graph preconditioner fares on a more traditional problem. Experiments with the MSF support graph preconditioner were run with slightly random edge weights so that the on the unweighted combinatorial Laplaican a random tree would be selected every time. The diagonal of the original matrix was kept for the preconditioner and on the F1 graph was scaled so the preconditioner would be positive-definite. The performance of the support graph preconditioner was compared against other Ifpack2 preconditioners. These include Jacobi, Symmetric Gauss-Seidel (SGS), and Incomplete Lower-Upper (ILUT). In parallel, Ifpack2's Additive Schwarz with no overlap was used with the support graph and ILUT preconditioners as sub-domain solvers. Ideally ILUT would be replaced with Incomplete Cholesky as we are dealing with symmetric matrices but this is currently not implemented in Ifpack2. The default ILUT parameters were used including a drop tolerance of 10^{-12} and fill value of 1. Additionally ILUT required a small, relative scaling of the diagonal ($\sim$1.01) to ensure a positive-definite preconditioner. All experiments were run on the 64 core Vesper (vesper@sandia.gov). Zoltan2's interface to the Scotch partitioner was used to distribute matrix rows amongst processors. Experiments were done with partitioning turned on and off using 8 cores to demonstrate the effect of partitioning on the various solvers. Scaling experiments were performed up to 32 cores.

Graph	Rows (Vertices)	NNZ (Edges $\times$ 2)
flickr	820,878	13,250,560
as-Skitter	1,696,415	22,190,596
hollywood-2009	1,139,905	57,515,616
F1	343,791	26,493,322

FIG. 3.1. *Graphs used in experiments*

	Iters.	Solve Time (s)
MSF	42	6.247
Jacobi	76	7.841
SGS	25	7.411
ILUT	30	8.331
None	2689	266.3

(a) Combinatorial Laplacian

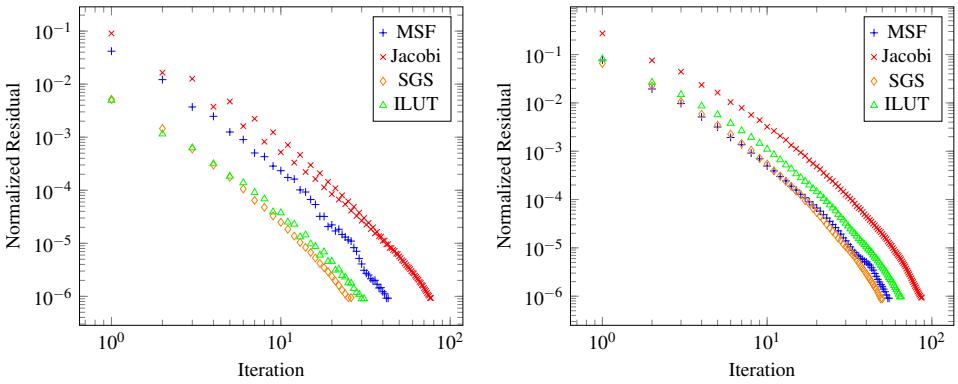
	Iters.	Solve Time (s)
MSF	54	7.713
Jacobi	86	8.942
SGS	49	14.32
ILUT	64	15.85
None	86	8.573

(b) Normalized Laplacian

FIG. 4.1. *Serial results on flickr graph*

4. Results.

4.1. Serial. The results of serial solves on the flickr graph for both the combinatorial and normalized Laplacians can be seen in Figure 4.1. It is clear from using no preconditioning on the combinatorial Laplacian that some preconditioning method is needed. In the combinatorial case MSF yields the best solution time with SGS yielding the fewest number of iterations. The normalized Laplacian is scaled so that solving



(a) Combinatorial Laplacian (b) Normalized Laplacian

FIG. 4.2. *Iterations to convergence*

without preconditioning is equivalent to using Jacobi. Interestingly in the normalized case the preconditioned solves are all slightly more expensive. One idea being considered to solve the combinatorial problem involves solving the normalized problem and converting the solution vector since the normalized problem should be better conditioned. However these results seem to suggest such a method would not be fruitful. In Figure 4.2 the convergence rates are shown by the normalized residual at each iteration. Interestingly the preconditioners seem to have very similar convergence rates that just appear to be off by some constant which seems to be smaller in the normalized case.

	Iters.	Solve Time (s)
None	18	9.516
Scotch	17	8.695

(a) MSF

	Iters.	Solve Time (s)
None	122	10.71
Scotch	128	4.005

(b) Jacobi

	Iters.	Solve Time (s)
None	116	16.39
Scotch	128	6.166

(c) SGS

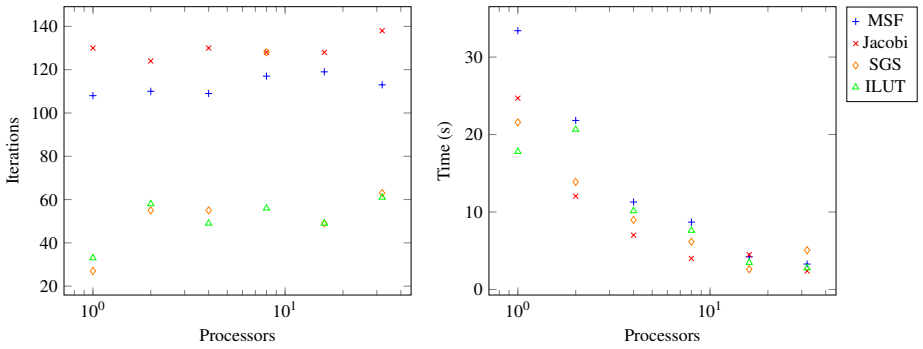
	Iters.	Solve Time (s)
None	120	16.45
Scotch	56	7.613

(d) ILUT

FIG. 4.3. *Partitioning results using 8 cores on as-Skitter combinatorial Laplacian*

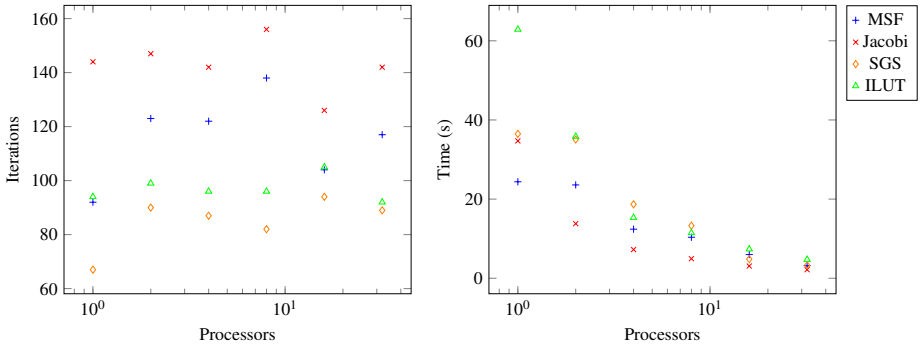
4.2. Partitioning. The Scotch graph partitioning algorithm was used in parallel solves to try and increase the quality of the preconditioner and improve the load balance across processors. The results with partitioning turned on and off for 8 processors on the as-Skitter combinatorial Laplacian are shown in Figure 4.3. Improvement in the quality of the preconditioner can be inferred by the change in iteration count while the change in run time is some mix of load balance improvement and change in preconditioner quality. Jacobi behaves as expected; since it just uses diagonal scaling the quality of the preconditioner does not change but load balancing greatly improves performance. Only with ILUT does using partitioning seem to greatly improve the quality of the preconditioner. Each sub-domain incomplete factorization needs to have as many edges as possible to improve quality. However, the number of edges in the MSF sub-domain is constant so having an extra edge in the sub-domain won't change the quality very much. The graph algorithm might

have selected this edge instead but some other edge would not have been selected. This leads us to believe that the MSF preconditioner should scale reasonably well since losing the edges off processor won't hurt the preconditioner quality very much. Since turning partitioning on always leads to at least slightly better performance due to better load balancing, it is always turned on for the scaling experiments in the next section.



(a) Iteration Performance (b) Timing Performance

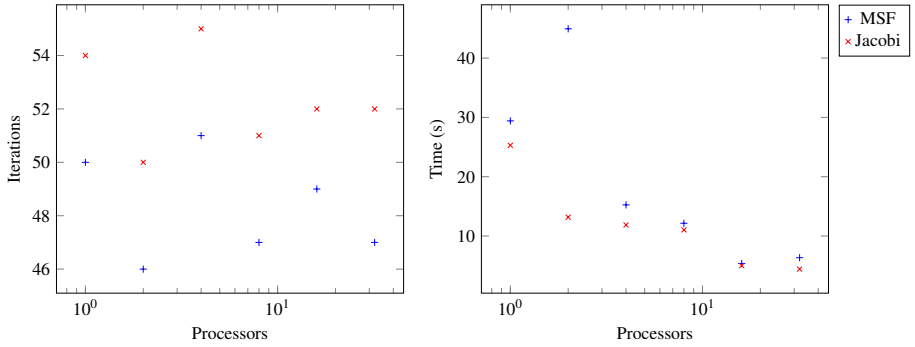
FIG. 4.4. *Scaling on as-Skitter graph combinatorial Laplacian*



(a) Iteration Performance (b) Timing Performance

FIG. 4.5. *Scaling on as-Skitter graph normalized Laplacian*

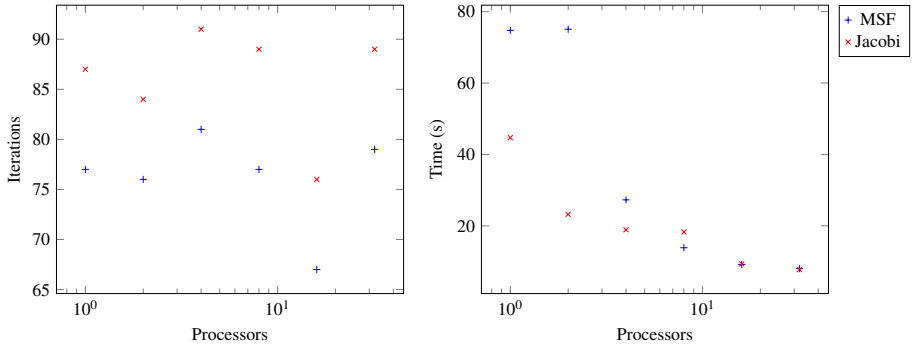
4.3. Scaling. Up to 32 cores of Vesper were used to perform scaling experiments on as-Skitter and hollywood-2009. Results for as-Skitter's combinatorial Laplacian are shown in Figure 4.4. A trial with no preconditioning was run but the performance was so poor (10 to 100 $\times$ slower) that the results are excluded. Results for as-Skitter's normalized Laplacian are shown in Figure 4.5. The number of iterations required for each preconditioner doesn't fluctuate much as the number of processors increases and their order stays relative the same. Some fluctuation in iteration count is expected due to using a random right hand side during every solve. Jacobi seems to have the best solve time performance though as the number of processors increase the gap between the preconditioners decreases. Since Jacobi and MSF seem the most competitive the experiments on the larger hollywood-2009 were run only with them. The combinatorial results are shown in Figure 4.6 and the normalized results are shown in Figure 4.7. The number of iterations



(a) Iteration Performance (b) Timing Performance

FIG. 4.6. *Scaling on hollywood-2009 graph combinatorial Laplacian*

for both of these methods seems to fluctuate a bit more on this larger graph though the gap between them stays the same. With the exception of MSF on 2 processors both methods seem to scale well regarding solve time. We suspect that there is some initial MPI overhead causing the spike at 2 processors which is quickly overcome.



(a) Iteration Performance (b) Timing Performance

FIG. 4.7. *Scaling on hollywood-2009 graph normalized Laplacian*

	Iters.	Solve Time (s)
MSF	408	3.106
Jacobi	580	4.254
SGS	297	7.168
ILUT	199	3.654

FIG. 4.8. *F1 stiffness matrix solved using 32 cores*

4.4. Stiffness Matrix. Out of curiosity we ran one experiment on a more traditional stiffness matrix F1. This matrix is symmetric positive-definite so no modifications were needed to use Belos' CG solver.

However, this matrix has both positive and negative off-diagonal entries so the diagonal of the MSF and ILUT preconditioners had to be modified to ensure the preconditioners were positive-definite. Solves were done using 32 cores and the results are shown in 4.8. ILUT and MSF perform relatively better against Jacobi and SGS than they did in with the network graphs. The structure of this graph is much simpler so we suspect that the main reason is the introduction of edge weights.

5. Conclusions. It is clear that some method of preconditioning is required to solve linear systems coming from graphs. However, it is not clear if a support graph preconditioner is useful or if something as simple as Jacobi should be used instead. We observed that Jacobi preconditioning works quite well: even if the iteration count is high, each iteration is very fast. We remark that MSF is a very simple support graph preconditioner and there is potentially room for improvement by choosing better subgraphs. Partitioning has been shown to be useful for all the preconditioners tested but most important for SGS and ILUT. All of the preconditioners seem to scale well on the graphs used as the number of processors increase. This suggests that using additive Schwarz with local (serial) preconditioning is a viable approach for network problems. We observed that it is typically more difficult (longer run times) to solve for the normalized Laplacian so using the normalized Laplacian to solve a system arising from the combinatorial Laplacian does not seem viable.

6. Future Work. Most of the software for these experiments was just written so while the results of these experiments seem reasonable there could still be bugs to clean up. Tests and documentation are needed before this software is brought out of experimental. There are a few followup experiments that should be run. A set of weak scaling experiments would help understand how these methods perform as graph sizes increase. This will require choosing a reasonable graph generator. These experiments used the matrix ordering in the original files and it is possible that this ordering might be helping some of the methods so a trial of experiments with randomly reordered matrices should be performed.

Acknowledgment.
The authors would like to thank Karen Devine, Rich Lehoucq, and Siva Rajamanickam.

REFERENCES

[1] M. BERN, J. R. GILBERT, B. HENDRICKSON, N. NGUYEN, AND S. TOLEDO, *Support-graph preconditioners*, SIAM J. on Matrix Anal. and Appl., 27 (2006), pp. 930–951.
[2] Y. CHEN, T. A. DAVIS, W. W. HAGER, AND S. RAJAMANICKAM, *Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate*, ACM Trans. Math. Softw., 35 (2008), pp. 22:1–22:14.
[3] T. A. DAVIS AND Y. HU, *The University of Florida Sparse Matrix Collection*, ACM Transactions on Mathematical Software, 38 (2011), pp. 1:1–1:25.
[4] L. GRADY, *Random walks for image segmentation*, IEEE Trans. Pattern Anal. Mach. Intell., 28 (2006), pp. 1768–1783.
[5] M. A. HEROUX, R. A. BARTLETT, V. E. HOWLE, R. J. HOEKSTRA, J. J. HU, T. G. KOLDA, R. . B. LEHOUCQ, K. R. LONG, R. P. PAWLOWSKI, E. T. PHIPPS, A. G. SALINGER, H. K. THORNQUIST, R. S. TUMINARO, J. M. . WILLENBRING, A. WILLIAMS, AND K. S. STANLEY, *An overview of the trilinos project*, ACM Trans. Math. Softw., 31 (2005), pp. 397–423.
[6] L. KATZ, *A new status index derived from sociometric analysis*, Psychometrika, 18 (1953), pp. 39–43.
[7] I. KOUTIS, G. MILLER, AND R. PENG, *Approaching optimality for solving SDD linear systems*, in Proc. of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science, IEEE Computer Society, 2010, pp. 235–244.
[8] M. NEWMAN, *Networks: An Introduction*, Oxford University Press, Inc., New York, NY, USA, 2010.
[9] D. A. SPIELMAN AND S.-H. TENG, *Nearly-linear time algorithms for graph partitioning, graph sparsification, and solvi ng linear systems*, in Proceedings of the thirty-sixth annual ACM symposium on Theory of computing, STOC '04, New York, NY, USA, 2004, ACM, pp. 81–90.
[10] D. A. SPIELMAN AND S.-H. TENG, *Spectral partitioning works: Planar graphs and finite element meshes*, Linear Algebra and its Applications, 421 (2007), pp. 284 – 305.
[11] U. VON LUXBURG, *A tutorial on spectral clustering*, CoRR, abs/0711.0189 (2007).

QUANTIFYING THE IMPACT OF SINGLE BIT FLIPS IN GMRES

JAMES J. ELLIOTT* AND MARK HOEMMEN[1]

Abstract. Increasing parallelism and transistor density, along with increasingly tighter energy and peak power constraints, may force computer hardware manufacturers to expose incorrect computation or storage to application codes. In this work, we quantify what a bit flip in floating point data entails, and develop a novel operation-centric fault model that allows us isolate and bound the error introduce by a silent, transient bit flip in data. We consider what happens to an *unmodified* iterative linear solver, GMRES, if we allow transient memory corruption in computational kernels' input data. Coupled with our fault model, we develop a detection strategy for faults in GMRES' major kernels: the matrix-vector product, applying the preconditioner, and orthogonalizing the new basis vector against previous basis vectors. We model the effects of memory corruption on the binary representation of floating-point numbers, and use this to show how scaling the linear system can force faults either to be bounded, or to be easily detectable without additional computation. In future work, we will apply these techniques to improve the convergence rate of Fault-Tolerant GMRES (see [5]) if faults do occur.

1. Introduction. The implications of having a single bit toggle while performing a numerical algorithm can seem daunting. Simple experiments can show that having a single bit flip can cause certain algorithms to "crash" (terminate abnormally, due to invalid states or actions detected by the application or operating system), "stagnate" (keep running but fail to make progress), or even produce the wrong solution, silently. This work focuses on understanding how a transient bit flip in the data used by the algorithm, translates to numerical error. With the ability to translate a bit flip into a numerical error, we are able to use mathematical analysis to reason about hardening algorithms should a bit flip occur. We present two main contributions:

- How to use mathematical analysis and IEEE-754 double precision data representation to bound the error introduced by a bit flip in the data used by the algorithm.
- How to exploit the IEEE-754 specification to ensure that should a bit flip occur, the error introduced will be small (less than one), or much larger than the bounds imposed by mathematical analysis.

To demonstrate our technique we begin with the GMRES iterative linear solver. In future work, we will extend this to the Fault-Tolerant GMRES solver. As a result of extending our work to FT-GMRES, we identify future research challenges, which we are able to pose as mathematical problems, that is, no knowledge of bit flips is required.

1.1. Faults, Failures, and Persistence. In this work we address a very specific type of fault. Our goal is to ensure that this type of fault does not produce a failure. We consider two perspectives: the user and the system. A fault occurs at the system level, e.g., a bit flips or a node crashes. *A fault becomes a failure if it impacts the user.* Figure 1.1 depicts a visual taxonomy of how we consider faults and the scope of our work. We further classify faults into those that interrupt the user's program (hard faults), and those that do not immediately or ever interrupt the users code (soft faults). Clearly, a hard fault results in a failure if the user is running an application, but the very nature of soft faults implies that they may emit no indication that something has gone wrong. In the event that soft faults allow the program to continue execution with tainted data, called Silent Data Corruption (SDC), then we must understand how algorithms behave in the presence of faulty data. Furthermore, if the algorithm uses tainted data and still obtains the correct solution, then the fault does not constitute a failure. If the soft fault leads to the incorrect solution, then the fault leads to a failure.

We further classify soft faults by how long the underlying hardware remains faulty. Persistent faults arise from hardware that is permanently faulty, e.g., a stuck bit in memory, or the Intel Pentium FDIV bug [11]. Sticky faults indicate hardware that is faulty for some duration but returns to normal operation.

*North Carolina State University, Department of Computer Science, jjellio3@ncsu.edu
[1]Sandia National Laboratories, mhoemme@sandia.gov

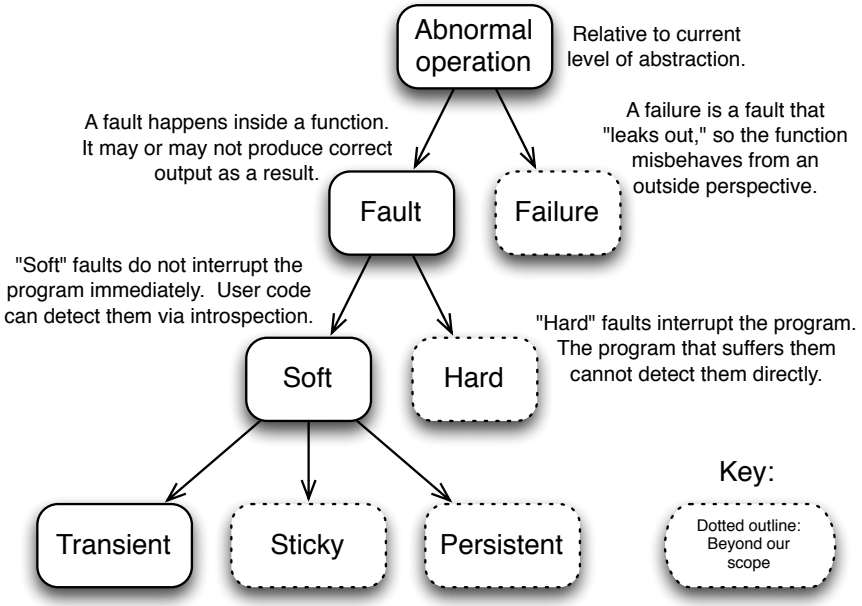


FIG. 1.1. *Taxonomy of faults and scope of this work.*

Transient faults occur once, and while the bit flip is transient the effect of the flip may be persistent, which we discuss at length in § 4.

1.2. Problem Statement. The intent of our work is, ultimately, to guard against silent, transient, soft faults. We pose the problem as follows: *Assume that a single transient bit flip silently corrupts the data used in an algorithm. Can we understand how the error introduced by this bit flip impacts the solution of algorithm?* In the context of this work, the bit flip is silent, and we assume that any detection mechanisms failed. Based on this premise, we seek to quantify how a bit flip in data *could* impact the solution, and we consider all possible perturbations to the data.

1.3. Assumptions and Justification. Our assumption that a single bit flip impacts data is fundamental, and we justify this decision as follows:

1. Hardware employs techniques to ensure that so-called "single event upsets" (SEUs) – that is, bit flips – do not occur. Therefore, it is expected that SEUs will be rare events.
2. If we can understand the best- and worst-case scenarios for error that a single bit flip can contribute, we will have a baseline to conjecture about multiple bit flips.
3. We lack any knowledge of a statistical distribution indicating the rate at which bit flips can occur. Therefore, speculation about flip rates may or may not prove useful.
4. Stochastic models are of limited value given that the expected behavior of the algorithm is dependent on the data and parameters used in the algorithms.

We have a strong reason to believe that SDC is a rare event. Hardware incorporates a fairly large amount of safeguards in-place to protect data and instructions. In example, Intel provides the Machine Check Architecture, which provides reporting of bit errors at the register, cache (L1-L3), QuickPath Interconnect, and DRAM (via ECC) layers. We do not attempt to conjecture about the likelihood of bit flips, rather we turn

to the theoretical basis that an algorithm is built on, and study how the algorithm behaves when perturbed within the bounds imposed by mathematical analysis. By following this research path, we are able to avoid the pitfalls presented in bullets #3 and #4, and we are able to isolate the impact of a bit flip in data without other factors polluting our analysis.

2. Motivation. Energy and peak power increasingly constrain modern computer hardware, yet hardware approaches to protect computations and data against errors cost energy. This holds at all scales of computation, but especially for the largest parallel computers being built and planned today. This results from a confluence of factors:

- Increasing parallelism (and therefore more components to fail) [3,4]
- Decreasing transistor feature sizes, making individual components more vulnerable
- Extremely tight peak power requirements [13], limiting the use of hardware redundancy to increase reliability

As these trends continue, hardware vendors may succumb to the temptation to expose incorrect arithmetic or memory corruption to application codes [12, 13, 15]. Some studies already indicate that this behavior is appearing at the user level [9].

Much of the prior work on fault-tolerant iterative solvers has taken the approach of focusing on specific linear algebra operations, in particular on the sparse matrix-vector multiply (SpMV) operation [18,21]. Other work has treated algorithms as black boxes [7]. A common feature among [7,18,19,21] is a methodology of first proposing a fault-tolerance technique, and then evaluating the effectiveness of the technique by random sampling. The focus in this type of research has been to detect errors, and then to perform some response – e.g., correct the tainted values, or roll back and resume computation from an assumed valid state – while hoping the fault does not occur frequently enough to cause stagnation. In addition, all prior work on sparse iterative methods is based on a fault model that assumes multiple faults injected at some rate. Studies are carried out with little care for whether the bit flipped is a $0 \rightarrow 1$ or a $1 \rightarrow 0$, and most studies flip bits randomly. In summary, the above work makes the following assumptions:

1. Numerical algorithms are simply a collection of operations, or a "black box" subroutine executing opaquely.
2. Techniques are validated via random sampling.
3. Fault models and experimental methodology presume multiple faults that happen at a specified "rate."

We question these assumptions.

If SDC is a rare event, a finding backed by years of correct solutions as well as current research [14], then why has current SDC research focused on failure rates? Michalak et al. found that when a Roadrunner node was placed in front of a neutron cannon and subsequently bombarded with charged particles, this resulted in a startlingly low occurrence of SDC (and primarily resulted in outright node failure) [14]. Clearly, Exascale does not imply that nodes will be built with particle cannons attached to the boards to guarantee a high rate of SEUs. Instead, SDC should remain a very rare event, and currently, little, if any, research has attempted to explain how a single bit flip impacts an algorithm and ultimately the solution. The studying of multiple bit flips has had an adverse affect on our ability to glean information from the studies. Instead of characterizing SDC, we have studies that propose solutions to a problem we have very little understanding of. It is our goal to first analyze SDC, and then to propose both specific algorithmic techniques and general heuristics that minimize the impact of SDC should it occur. With this ability, mathematicians, scientists, and engineers can take quantifiable steps to develop algorithms and applications that have an inherent resilience to SDC.

Also, a bit flip (or many bit flips) have a mathematical meaning, given that they impact data used by a numerical method. Research demonstrating that a proposed scheme can detect or correct errors is only as valid as the fault model that injects said errors. We advocate a drastically different approach, namely that bit

flips impact the underlying mathematical assumptions that are used to guarantee the algorithm is convergent. Rather than focusing on detecting binary errors, we treat bit flips as numerical error and evaluate how these errors relate to the theoretical basis that the algorithm is built on. In this sense, we filter values that are theoretically impossible, while accepting variations that are allowable by the theory. While our approach does not "solve" the SDC problem, we are able to use our findings to isolate clear, well posed problems for further research.

Based on the above arguments, first we seek to define the semantics of a bit flip in floating-point data, and then propose a reliability scheme rooted in mathematical analysis. These are disjoint tasks. Quantifying the impact of a single bit flip provides many insights, and has allowed us to relate how the inputs to the algorithm dictate the types of errors we can see from a bit flip.

Numerical algorithms (that use floating-point arithmetic to approximate continuous mathematical problems) can be particularly sensitive to incorrect arithmetic or data corruption. Even a single fault may cause *silently incorrect answers*. For example, if a sparse matrix A gets permanently corrupted, a linear solver for $Ax = b$ will compute the "right answer to the wrong problem." Fortunately, many numerical algorithms only need reliability for certain data and phases of computation. If the system can guard just those parts of the algorithm in space and time, then the algorithm can compute the right answer – or at least be able to detect failure and report it "loudly" – despite faults in unreliable phases of execution. This suggests a "layered" approach to the design of reliable numerical algorithms. A reliable outer layer can recover from faults in a less reliable inner layer. If the solver can spend most of its time in unreliable mode, it can mitigate the cost of reliable computation in the outer mode. Bridges et al illustrate such a numerical algorithm – the "Fault-Tolerant GMRES" (FT-GMRES) iterative linear solver – in [5]. It was also shown that such an algorithm can cooperate with a run-time system to recover from memory corruption [6].

3. Project Overview. To quantify the possible effects of a silent bit flip in GMRES, we propose a multifaceted approach. We combine modeling based on the IEEE-754 specification of a floating-point number [10] and standard mathematical analysis to determine the semantics of a bit flip in specific numerical routines. First, by analyzing the effects of data corruption on the representation of floating-point numbers, we can show that most hardware faults either result in bounded error, or cause an obvious error that an algorithm can easily detect. This excludes the most damaging errors, and allows us to have predictable error should SDC occur. Second, we develop a fault detection technique for the Modified Gram-Schmidt (MGS) component of the Arnoldi process in the GMRES algorithm based on mathematical analysis of the norm bounds on the intermediate MGS vectors. Third, we illustrate how the scaling of the data used in GMRES impacts the distribution of potential numerical errors that could arise from a bit flip in data. Fourth, we conclude by evaluating the future work necessary to bound the error of the entire inner solve, and how such a bound could be used to enhance the FT-GMRES algorithm.

In summary, the overarching goal of this work is to investigate algorithms in the presence of a silent bit flip by defining and correlating the semantic meaning of a bit flip in floating point data represented using the IEEE-754 specification to the analytical bounds derived from the mathematical theory. Using this analysis approach, we ultimately show that the error introduced by a bit flip can be bounded.

4. Fault Model. The premise of our work is that a silent, transient bit flip impacts data. Before we can perform any analysis or experimental work, we must define how such a bit flip would impact an algorithm, and how we enforce that the bit flip was transient. To achieve this goal, we build our model around the basic concept that when data is used by the algorithm, this translates into some set of operations being performed on the data. Should a bit flip perturb our data, some operation will use a corrupt value, rather than the correct value. The output of this single operation will then contain a tainted value, and this tainted value could cause the solution to be incorrect. Note that a transient bit flip may cause a persistent error in the output depending

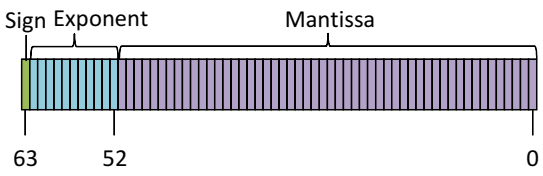


FIG. 4.1. *Graphical representation of data layout in the IEEE-754 Binary64 specification.*

on how the value is used.

A side effect of an operation-centric model is that we naturally avoid a pitfall that arbitrary memory fault injection succumbs to, which is that if a bit flip impacts data (or memory) that is never used (read) then this fault can *not lead to a failure*. Our fault model allows a bit flip to perturb the input to an operation performed on the data, while not persistently tainting the storage of the inputs, which we feel mimics how a transient bit flip would manifest itself, e.g., the data that experiences the bit flip need not show signs that it was perturbed. This model allows us to observe the impact of transient flips on the inputs, which results in sticky or persistent error in the result. We then utilize mathematical analysis to model how this persistent error propagates through the algorithm.

4.1. Fault Characterization via Semantic Analysis. To establish a fault model we must first understand what a fault is. Since floating-point numbers approximate real numbers and most numerical algorithms use real numbers, we start from the definition of a real-valued scalar $\gamma \in \mathbb{R}$. The range of *possible* values that γ can take is

$$\gamma \in [-\infty, +\infty].$$

We assume that the IEEE-754 specification for double-precision numbers, called *Binary64*, is used to represent these numbers. This means that γ can take a fixed set of numeric values, and these values lie in the range

$$\gamma \in [-1.80 \times 10^{308}, +1.80 \times 10^{308}],$$

or using base two for the exponent

$$\gamma \in [-1.\bar{9} \times 2^{1023}, +1.\bar{9} \times 2^{1023}],$$

where $1.\bar{9}$ indicates the largest possible fractional component, and 1.0 indicates the smallest fractional component. A more informative range is that of $|\gamma|$, excluding 0 and denormalized numbers,

$$|\gamma| \in [2.23 \times 10^{-308}, 1.80 \times 10^{308}], \tag{4.1}$$

and in semi base two

$$|\gamma| \in [1.0 \times 2^{-1022}, 1.\bar{9} \times 2^{1023}]. \tag{4.2}$$

To approximate real numbers, *Binary64* uses 64 bits, of which 11 are devoted to the exponent, 52 for the fractional component (we refer to as the mantissa), and one bit for the sign. Figure 4.1 shows how these bits are laid out. In addition to numeric values, Binary64 includes two non-numeric values, Not-a-Number (NaN) and Infinity (Inf), which may be signed, to account for infinity and values that result in undefined

operations, e.g., division by zero. The range of values in Equations (4.1) and (4.2) is not continuous, and has non-uniform gaps due to the discrete precision, which is a consequence of having a fixed number of bits in the fractional component.

We can further discretize the range of possible values by recognizing that there is a finite number of exponents that are possible given IEEE-754 double precision, e.g.,

$$\gamma \in \{0, \pm\text{Inf}, \pm\text{NaN}, \pm 2^{-1022} \times 1.x, \pm 2^{-1021} \times 1.x, \ldots, \pm 2^{0} \times 1.x, \ldots, \pm 2^{1023} \times 1.x\},$$

where $1.x$ indicates some fractional component.

Analytically, this is expressed as

$$\gamma = (-1)^{sign} \left(1 + \sum_{i=0}^{51} b_i 2^{i-52}\right) \times 2^{e-1023}, \tag{4.3}$$

for IEEE-754 *Binary64*. Note, the specification does not include a sign bit for the exponent. Rather, IEEE floating point numbers utilize a *bias* to allow the exponent to be stored without a sign bit, which we will later exploit for fault-resilience. Another important characteristic that stems from the general approach of expressing numbers in exponential notation, is that we can characterize numbers by their order of magnitude. Of particular interest is the following relation

$$\begin{aligned} &|2^{-1022}| \leq |2^{-1022} \times 1.x| \\ &< |2^{-1021}| \leq |2^{-1021} \times 1.x| < \ldots \\ &< |2^{0}| \leq |2^{0} \times 1.x| < \ldots \\ &< |2^{1023}| \leq |2^{1023} \times 1.x|. \end{aligned} \tag{4.4}$$

This means that we can use the next order of magnitude as an upper bound for errors in the fractional component of a number — which is practically achieved by incrementing the exponent or multiplying by two. We can also analytically model the number of fractional bits that could contribute error larger than some tolerance, since the error that *could* arise from each mantissa bit is relative to the exponent of the number. This final step is necessary since the fractional term can take values in the range $[1,2)$, where the left parenthesis indicates that 2 is not a member of this interval. We can also characterize the error that a perturbed sign bit can contribute, and, like the fractional component, this error is relative to the exponent of the number. Suppose the sign is perturbed in a scalar γ, then we have $\tilde{\gamma} = -\gamma$, the absolute error is $|\gamma - \tilde{\gamma}| = |\gamma - (-\gamma)| = 2\gamma$. Which means we can bound the error from a sign bit perturbation by incrementing the exponent of the resulting value.

In summary, we have demonstrated that errors in IEEE-754 floating point numbers can be characterized using the exponent of the numbers. This property allows us to reduce the number of bits we need consider in a fault model, since we know that a large number of errors are bounded by the relatively small set of possible exponents.

4.2. Fault Characteristics of Perturbed Exponents. In the context of IEEE-754 double precision numbers and silent data corruption, we do not model the exponents directly. Instead, we model the biased exponents, as they are the interesting portion of the *data* that allows us to characterize the errors that the majority of the bits present in the data can produce. For instance, in double precision data we can characterize the errors from 53 of the 64 bits using our approach. This type of fault-characterization is impossible if bit flips are injected randomly into the data's memory, as that approach loses the semantic information that is implicitly present in the data.

The Binary64 specification does not store exponents directly, instead it uses a bias of 1023. From § 4.1 this means we can characterize *all* faults in double precision data by analyzing perturbations to the possible biased exponents

$$\{0, 1, 2, \ldots, 1023, \ldots, 2046\}.$$

Note that zero is not a biased exponent, and has special meaning. In IEEE-754, a zero pattern in the exponent with zeros in the mantissa is used to represent the scalar zero, and while a non-zero pattern in the mantissa is used to represent subnormal numbers. We also assume the user does not perform computation on the two non-numeric values NaN and Inf, which are represented using the biased exponent 2047 (all ones). We do include zero in our analysis because it is a valid real number.

Since we are concerned with bit perturbations in the exponent, we express the biased exponents in their binary form, e.g., 11-bit unsigned integers presented in binary. We can further expand Figure 4.2 to show

$$\underset{\text{Exponent}}{\begin{Bmatrix} 2^{-1} \\ 2^{0} \\ 2^{1} \end{Bmatrix}} \Rightarrow \underset{\text{Biased}}{\begin{Bmatrix} 1022 \\ 1023 \\ 1024 \end{Bmatrix}} \Rightarrow \underset{\text{Storage}}{\begin{Bmatrix} 01111111110 \\ 01111111111 \\ 10000000000 \end{Bmatrix}}$$

FIG. 4.2. *Relation of exponent, IEEE-754 double precision bias, and what data are actually stored.*

the potential change to the original exponent should a bit flip occur, which will form the basis for our fault model and analytic models.

In the context of bit flips, we can view a bit flip as adding or subtracting from the biased exponent, which in turn translates to multiplying or dividing the number by some power of two. We can model the impact of a bit flip in the exponent as the original scalar being magnified or minimized by a specific powers of two. We illustrate this in Figure 4.3, where reading left-to-right, we have some initial exponent, which is represented using a bias of 1023, which translate to a discrete binary pattern. We then consider all bit flips in this binary pattern and compute the actual perturbed exponent. We then show how the perturbation can be modeled independent of the original exponent. Table 4.1 summarizes how a bit upset on a double precision scalar can be modeled, and Table 4.2 summarizes how we can characterize the error that the fault introduces.

By characterizing the error introduced, we recognize that the mantissa flips introduce error that has the same exponent as the original number, and a sign flip introduces error that is only one order of magnitude larger than the original number. Furthermore, the exponent bits can either introduce large error, or a flip introduces error roughly equivalent to the order of magnitude of the original number. Suppose we can enforce that all numbers used in calculations are less than 1.0, then we know that the majority of the bits will produce error that is also less than one, since 51 of the total 52 mantissa bits will contribute error less than 1.0. We also see that some of the exponent bits have the potential to contribute error less than 1.0, which indicates if we can enforce or assume some properties of the data used in the calculations e.g., data less than one, then we can greatly increase the likelihood that a bit flip introduces error no greater than 1.0. This phenomena is shown in Figure 4.3, where we can see empirically that numbers with exponent 2^0 and 2^{-1} introduce small error, compared with the errors introduced with the exponent 2^1.

In summary, we can characterize the impact of a bit flip in the exponent, because the sign and mantissa bits produce error that is relative to the exponent. As discussed in § 4.1 and analytically presented in Eq. (4.4), we are able to relate bit upsets to numerical error in terms of the exponent of the original number. Table 4.2 summarizes how a bit upset impacts a single value, and expresses how the order of magnitude

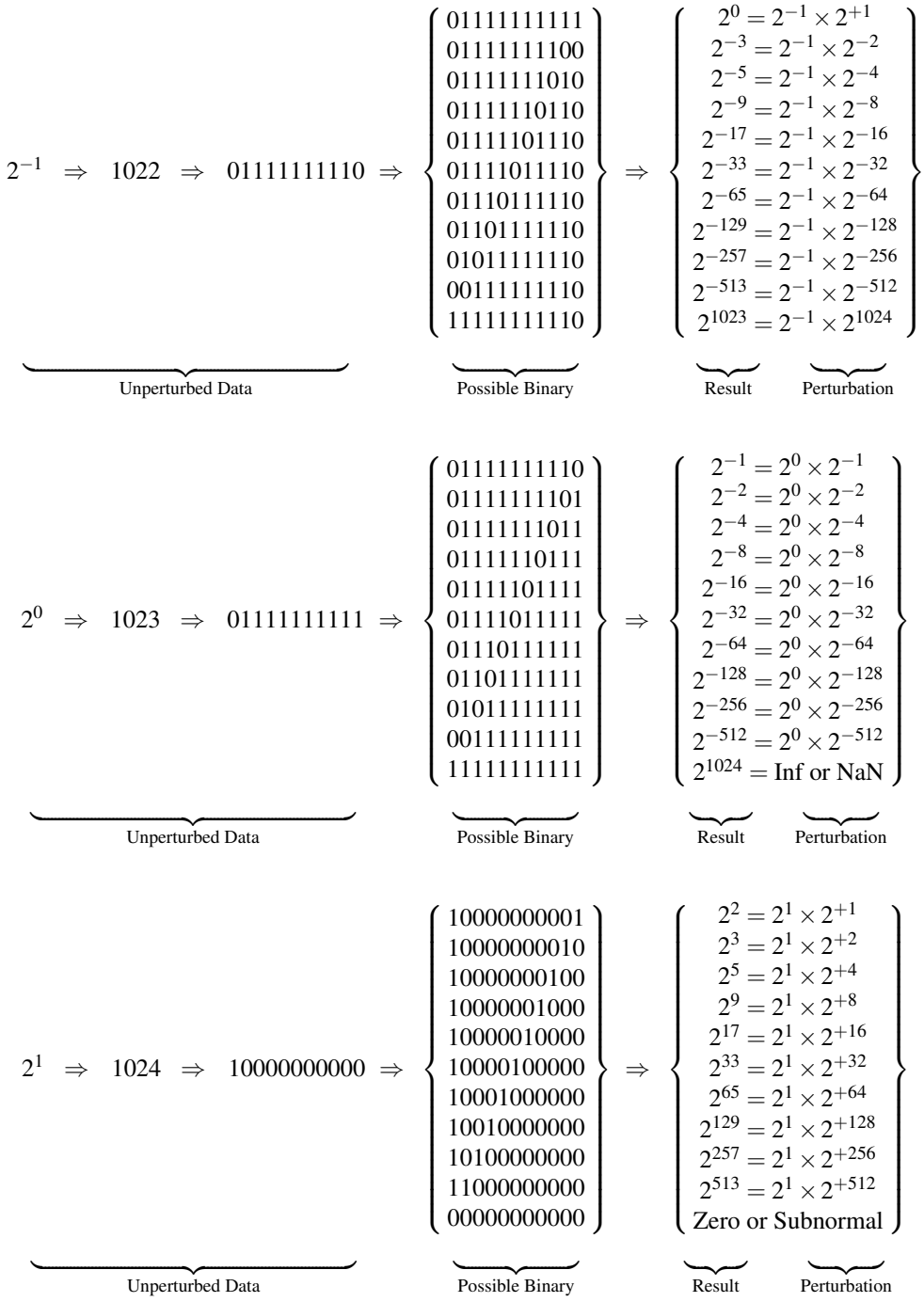


FIG. 4.3. *Examples of how a bit flip can impact an exponent represented using the IEEE-754 Binary64 specification.*

TABLE 4.1
Perturbations possible to IEEE-754 double precision value λ, producing the perturbed value $\tilde{\lambda}$.

Bit Location	Perturbed Scalar	Bit Range
Mantissa	$\tilde{\lambda} = \lambda + \lambda_{\text{exp}}(1+2^{j-52})$,	for $j = 0,\ldots,51$
$\text{Exponent}_{1\to 0}$	$\tilde{\lambda} = 2^{-2^j}\lambda$,	for $j = 0,\ldots,10$ and $bit_{j+52} = 1$
$\text{Exponent}_{0\to 1}$	$\tilde{\lambda} = 2^{2^j}\lambda$,	for $j = 0,\ldots,10$ and $bit_{j+52} = 0$
Sign	$\tilde{\lambda} = (-1)^{(bit_{63}+1)}\lambda$,	

TABLE 4.2
Bit flip absolute error for a scalar λ represented using IEEE-754 double precision, with $\lambda = \lambda_{\text{exp}} \times \lambda_{\text{frac}}$. Where λ_{exp} is the exponent 2^x, and λ_{frac} is the fractional component.

Bit Location	Absolute Error: $\left\lvert\lambda - \tilde{\lambda}\right\rvert$		Δ Order[†]
Mantissa	$\left\lvert\lambda_{\text{exp}}(1+2^{j-52})\right\rvert$,	for $j = 0,\ldots,51$	0
$\text{Exponent}_{1\to 0}$	$\left\lvert\lambda(1-2^{-2^j})\right\rvert$,	for $j = 0,\ldots,10$ and $bit_{j+52} = 1$	-2^j
$\text{Exponent}_{0\to 1}$	$\left\lvert\lambda(1-2^{2^j})\right\rvert$,	for $j = 0,\ldots,10$ and $bit_{j+52} = 0$	$+2^j$
Sign	$\lvert 2\lambda\rvert$,		1

[†] The change in order of magnitude.

changes. Now that we have characterized a fault in a scalar, we will present a fault model centered around operations on scalars, where we assume one will be perturbed.

4.3. Operation Centric Fault Model. This work distinguishes itself from related work in the field of silent data corruption by developing a fault model that is not based on perturbing arbitrary memory locations. We seek a fault model and experimental methodology that expresses all possible errors, and not the expected error, which is what is obtained through random sampling. We also remove a strong assumption that is present in many memory hardening approaches, namely that perturbations to read-only data constitute a reasonable fault model. "Perturbations to read-only data" include, for example, a persistent change to the matrix A or right-hand side b of a linear system $Ax = b$. These types of faults will obviously produce incorrect solutions, given that the wrong problem was solved. Moreover, a fault model based only on DRAM does not capture how data behave in the system. For instance, the data might have been read from cache, rather than from DRAM, or a computation might be pipelined in such a way that intermediate results never leave the processor die.

4.3.1. Fault Model for Dot Products. We now describe a realization of our fault model that describes the error that could be injected if an operation in a dot product experiences a single bit upset. We choose the dot product because it is a common operation, and because we will use this model in § 6.1 to model the worst-case errors that could be injected into a phase of the GMRES algorithm.

Given two real-valued n-dimensional vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$, then the dot product is defined to be

$$c = \sum_{i=1}^{n} c_i, \quad \text{where} \quad c_i = a_i b_i. \tag{4.5}$$

If we allow a single bit flip to impact the i-th element of the dot product, then we have a perturbed solution

$\tilde{c}$, which is the result of a perturbation to either a_i, b_i, or c_i. In the context of our fault model, this captures a bit upset impacting the inputs to the multiplication operator, and it captures a bit upset in the intermediate value c_i, which is the input to the addition operator.

Using Table 4.1, we compose the above cases for a dot product. A bit flip in the result of the multiply will produce error as described in Table 4.2, and we need only construct a model for the absolute error should a bit flip perturb a multiplication. Table 4.3 summarizes the analytic models for a bit flip in a multiply based off our previous models of a perturbed scalar. We then determine the absolute error and potential change in magnitude, which is presented in Table 4.4.

TABLE 4.3
Perturbations possible given the multiplication of two IEEE-754 double precision scalars, where one operand is perturbed. $\lambda = \alpha\beta$ *and* $\tilde{\lambda} = \tilde{\alpha}\beta$.

Bit Location	Perturbed Product	Bit Range
Mantissa	$\tilde{\lambda} = \alpha\beta + \alpha_{\exp}(1+2^{j-52})\beta,$	for $j = 0,\ldots,51$
$\text{Exponent}_{1\to0}$	$\tilde{\lambda} = 2^{-2^j}\alpha\beta,$	for $j = 0,\ldots,10$ and $bit_{j+52} = 1$
$\text{Exponent}_{0\to1}$	$\tilde{\lambda} = 2^{2^j}\alpha\beta,$	for $j = 0,\ldots,10$ and $bit_{j+52} = 0$
Sign	$\tilde{\lambda} = (-1)^{(bit_{63}+1)}\alpha\beta,$	

TABLE 4.4
Bit flip absolute error for a perturbed multiplication of two IEEE-754 double precision scalars, where one operand is perturbed. $\lambda = \alpha\beta$ *and* $\tilde{\lambda} = \tilde{\alpha}\beta$.

Bit Location	Absolute Error: $\left\lvert\lambda - \tilde{\lambda}\right\rvert$		$\Delta\text{Order}^{\dagger}$
Mantissa	$\left\lvert\alpha_{\exp}(1+2^{j-52})\beta\right\rvert,$	for $j = 0,\ldots,51$	0 or 1
$\text{Exponent}_{1\to0}$	$\left\lvert\alpha\beta(1-2^{-2^j})\right\rvert,$	for $j = 0,\ldots,10$ and $bit_{j+52} = 1$	-2^j
$\text{Exponent}_{0\to1}$	$\left\lvert\alpha\beta(1-2^{2^j})\right\rvert,$	for $j = 0,\ldots,10$ and $bit_{j+52} = 0$	$+2^j$
Sign	$\lvert 2\lambda\rvert,$		1

† Potential change in order of magnitude relative to unperturbed result $\left\lvert\lambda_{mag} - \tilde{\lambda}_{mag}\right\rvert$.

The potential change in order of magnitude is paramount. Consider an exponent flip from $1 \to 0$. These types of exponent bit flips produce error that is bounded above by the original magnitude of the result, which can be viewed as "zeroing out" the term if a perturbation occurs. Similar to a perturbed scalar, the mantissa can contribute either no change in the order of magnitude, or in the worst case a bit flip causes a carry which will increment the order of magnitude by one. The order of magnitude for a sign bit flip is exactly the same as that of a perturbed scalar, which introduces error one order of magnitude larger than the result. These error models can be thought of as the largest additive error that we can inject into a dot product from a bit flip, e.g.,

$$\tilde{c} = \sum_{i=1}^{n} a_i b_i + (\text{error term}). \tag{4.6}$$

In summary, we have composed analytic models for the the absolute error that could be introduced into a dot product. Our models are initially constructed from the IEEE-754 Binary64 model, which we extended

to express how a bit upset impacts a single double precision scalar. We then composed a model for the multiplication operator, and analytically expressed the absolute error. Using the absolute error, we have a model that explains *how wrong* a dot product can be, assuming a bit flip in one of the input vectors or in an intermediate value. Next, we refine these models to construct strict upper bounds on the error introduced by a bit flip in a dot product.

4.3.2. Error Bounds for a Bit Flip in a Dot Product. The models presented in Table 4.4 make no assumptions about the bits present in the mantissa of the operands. This is problematic if we want to consider *all* possible errors that could be introduced into a dot product. To account for the mantissa, and in doing so create strict upper bounds on the error, we will use the relation presented in Eq. (4.4). From this relation, we know that $\alpha\beta < 2^{\alpha_{\text{exponent}}+1}2^{\beta_{\text{exponent}}+1}$, or we can write this as

$$\alpha\beta < 4\alpha_{\text{exp}}\beta_{\text{exp}}, \tag{4.7}$$

where $*_{\text{exp}} = 2^{*\text{exponent}}$. Using Eq. (4.7), we are able to account for the mantissa bits, but we can also show that a bit flip in the sign is bounded by Eq. (4.7). The sign bit introduces error equivalent to incrementing the exponent of the result

$$\alpha\beta < 2\alpha\beta < 4\alpha_{\text{exp}}\beta_{\text{exp}}, \tag{4.8}$$

where $2\alpha\beta$ is the potential error introduced should the sign bit be perturbed, which must be smaller than the bound constructed for the mantissa.

By utilizing Eq. (4.7), we are able to account for all possible mantissas and their potential faults, as well as a perturbation to the sign bit. We will now discuss how to use this model to understand the relationship between the data used in an algorithm and the distribution of potential errors that could occur should a bit flip in the data.

5. Fault Model Evaluation. In Section 4 we proposed analytic models for error introduced should a bit flip occur in IEEE-754 double precision data. We now illustrate how data can impact the size of errors that bit flips can create should they occur. Consider the follow sample vectors

$$\mathbf{u}_{\text{small}} = \begin{bmatrix} 0.5 \\ 0.25 \end{bmatrix}, \ \mathbf{u}_{\text{large}} = \begin{bmatrix} 2 \\ 4 \end{bmatrix}, \text{ and } \mathbf{v}_{\text{small}} = \begin{bmatrix} 0.25 \\ 0.5 \end{bmatrix}, \ \mathbf{v}_{\text{large}} = \begin{bmatrix} 4 \\ 2 \end{bmatrix}. \tag{5.1}$$

If we compute the dot product $\lambda = \mathbf{u}_{\text{large}} \cdot \mathbf{v}_{\text{large}}$, we have a finite number of potential errors should a bit flip in the data of $\mathbf{u}_{\text{large}}, \mathbf{v}_{\text{large}}$, or in an intermediate value in the summation. We can experience either $\tilde{2} \times 4 + 4 \times 2$, $2 \times \tilde{4} + 4 \times 2$, or $\tilde{8} + 8$. We have previously shown what $\tilde{2}$ can be in Figure 4.3, but for clarity we will state what the perturbed values could be

$$\tilde{2} = \left\{ \begin{array}{c} 2^2 \\ 2^3 \\ 2^5 \\ 2^9 \\ 2^{17} \\ 2^{33} \\ 2^{65} \\ 2^{129} \\ 2^{257} \\ 2^{513} \\ \text{Zero} \end{array} \right\}, \quad \tilde{4} = \left\{ \begin{array}{c} 2^1 \\ 2^4 \\ 2^6 \\ 2^{10} \\ 2^{18} \\ 2^{34} \\ 2^{66} \\ 2^{130} \\ 2^{258} \\ 2^{514} \\ 2^{-1020} \end{array} \right\}, \quad \tilde{8} = \left\{ \begin{array}{c} 2^4 \\ 2^1 \\ 2^7 \\ 2^{11} \\ 2^{19} \\ 2^{35} \\ 2^{67} \\ 2^{131} \\ 2^{259} \\ 2^{515} \\ 2^{-1018} \end{array} \right\}. \tag{5.2}$$

By inspection it is clear that substituting any of the above perturbed scalars into the dot product will produce absolute error greater than one in all cases, and in the event you choose to substitute the near zero perturbed values, the absolute error of the dot product still has magnitude 8, e.g., $|16-(0+8)|$.

Alternatively, consider the vectors $\mathbf{u}_{\text{small}}$ and $\mathbf{v}_{\text{small}}$. If we compute the dot product, $\lambda = \mathbf{u}_{\text{small}} \cdot \mathbf{v}_{\text{small}} = 0.25$. Then we have possible perturbations $\widetilde{0.5}$, $\widetilde{0.25}$, and $\widetilde{0.125}$, which we can construct from our model of a perturbed scalar, or we could directly compute the dot product error from our multiplication model.

$$\widetilde{0.5} = \left\{ \begin{matrix} 2^{0} \\ 2^{-3} \\ 2^{-5} \\ 2^{-9} \\ 2^{-17} \\ 2^{-33} \\ 2^{-65} \\ 2^{-129} \\ 2^{-257} \\ 2^{-513} \\ 2^{1022} \end{matrix} \right\}, \quad \widetilde{0.25} = \left\{ \begin{matrix} 2^{-3} \\ 2^{0} \\ 2^{-6} \\ 2^{-10} \\ 2^{-18} \\ 2^{-34} \\ 2^{-66} \\ 2^{-130} \\ 2^{-258} \\ 2^{-514} \\ 2^{1019} \end{matrix} \right\}, \quad \widetilde{0.125} = \left\{ \begin{matrix} 2^{-2} \\ 2^{-1} \\ 2^{-7} \\ 2^{-11} \\ 2^{-19} \\ 2^{-35} \\ 2^{-67} \\ 2^{-131} \\ 2^{-259} \\ 2^{-515} \\ 2^{1017} \end{matrix} \right\}. \tag{5.3}$$

By inspection, $\widetilde{0.5}$ can contribute absolute error to the dot product larger than one only once, e.g., $\left|0.25-(2^{1022}\times 0.25+0.125)\right|$. Likewise, $\widetilde{0.25}$ and $\widetilde{0.125}$ can perturb the result of the dot product with error greater than one, only once, and for all 3 cases the perturbation will change the result by hundreds of orders of magnitude.

Returning to Figure 4.3 explains what causes bit flips in the exponent to produce either a majority of large or small error, where we consider error less than one to be small. The binary pattern of the stored biased exponent contains predominantly zeros for numbers greater than one, and predominantly ones for numbers less than one.

You can also obtain primarily ones in the exponent as you approach the extrema of the biased exponents, e.g., numbers larger than 2^{512}. In this case, the biased exponent does contain many ones, however, because the number is sufficiently large, the absolute error will remain considerably large. This is because if one "zeroes out" a perturbed element in the dot product, the error is proportional to the magnitude of the result.

If one chooses to take the extrema route to guarantee many ones in the biased exponent, the side consequence is a substantial loss in precision, since one will want to ensure most numbers have magnitude larger than 2^{512}. Clearly, this would be a poor decision numerically. By operating on numbers approximately 1 in magnitude, one gains the benefit of optimal precision, i.e., 1×10^{-16}, while still maintaining predictable error should SDC occur.

5.1. Faults in the Mantissa or Sign. The error generated by the mantissa or sign bits is relative to the exponent of the number that the flip occurred in. If the exponent is larger than one, then clearly the mantissa or sign bits will generate an error larger than one. Alternatively, if the values all are less than one, then mantissa errors will produce errors less than one because $2^{-1}\times 1.x < 1.0 \leq 2^{0}$. The errors from the sign bit can not exceed 2, since we have $2\times 2^{-1}\times 1.x < 2^{1}$.

It is unreasonable not to consider the mantissa generating a carry, as discussed in § 4.3.2. To account for this we construct a strict upper bound by incrementing the exponent of each element of the vectors analyzed, similar to Eq. (4.7). For example,

$$\mathbf{u}_{\text{original}} = \begin{bmatrix} 2.12332 \\ 1.24568 \end{bmatrix} \Rightarrow \mathbf{u}_{\text{upper bound}} = \begin{bmatrix} 4 \\ 2 \end{bmatrix}. \tag{5.4}$$

We then can evaluate our models on these vectors to determine a strict upper bound on the errors we can experience in a dot product.

5.2. Modeling Large Vectors. We have shown how to exhaustively examine each element in a vector, and from this analysis we can determine precisely which absolute errors we could experience. Given large vectors, where the dimension n may have millions or billions of elements, exhaustively searching each element would be time consuming, but it would also be a waste of time. As stated previously, there is a discrete number of exponents supported by the IEEE-754 Binary64 specification. As we have previously shown, the exponent characterizes the faults we can observe, so we only need consider the 2046 possible biased exponents and the special case of zero. The perturbations that are possible can be determined independent of knowing the data, e.g., we can precompute the perturbations and absolute error because we know the relation stated in Eq. (4.4) and Eq. (4.7).

To analyze arbitrarily large vectors, we will construct a lookup table for the absolute error in whatever operation we choose to model (we have chosen products and addition). The table size is 2047×2047, and allows us to consider the error introduced by performing an operation on two exponents, which will map to a unique ij location.

For example, consider the vectors

$$\mathbf{u} = \begin{bmatrix} 1.0 \\ 1.2 \\ 8.0 \\ 0.125 \end{bmatrix}, \text{ and } \mathbf{v} = \begin{bmatrix} 0.125 \\ 0.125001 \\ 0.125002 \\ 1.0 \end{bmatrix}. \tag{5.5}$$

We first extract the biases from the vectors

$$\mathbf{u}_{\text{Double Prec.}} = \begin{bmatrix} 1.0 \\ 1.2 \\ 8.0 \\ 0.125 \end{bmatrix} \Rightarrow \mathbf{u}_{\text{exponent}} = \begin{bmatrix} 2^0 \times 1.0 \\ 2^0 \times 1.x \\ 2^3 \times 1.0 \\ 2^{-3} \times 1.0 \end{bmatrix} \Rightarrow \mathbf{u}_{\text{biased}} = \begin{bmatrix} 1023 \\ 1023 \\ 1026 \\ 1020 \end{bmatrix} \tag{5.6}$$

Now, we determine an interval of possible values, and then account for the mantissa values which may have been truncated

$$u_i \in [1020, 1026] \in [1020, 1027] \text{ for } i = 1, \ldots, 4. \tag{5.7}$$

The range of biased exponents $[1020, 1027]$ will contain all possible values that the original vector contained, and include one value that was larger than any in the vector, the number corresponding to the biased exponent 1027. Similarly, we can compute the interval for $\mathbf{v}$

$$\mathbf{v} = \begin{bmatrix} 0.125 \\ 0.125001 \\ 0.125002 \\ 0.25 \end{bmatrix} \Rightarrow \begin{bmatrix} 2^{-3} \times 1.0 \\ 2^{-3} \times 1.x \\ 2^{-3} \times 1.x \\ 2^{-2} \times 1.0 \end{bmatrix} \Rightarrow \begin{bmatrix} 1020 \\ 1020 \\ 1020 \\ 1021 \end{bmatrix}, \tag{5.8}$$

and then we determine the interval for which we consider errors

$$v_i \in [1020, 1021] \in [1020, 1022] \text{ for } i = 1, \ldots, 4. \tag{5.9}$$

We now compute the lookup table,

	1020	1021	1022	1023	1024	1025	1026	1027
1020	×	×	×	×	×	×	×	×
1021		×	×	×	×	×	×	×
1022			×	×	×	×	×	×

,

where each entry computes the relevant perturbations and absolute error for the operations being modeled. In the case of multiplication, the table has symmetry because multiplication is commutative. In practice, computing the full table $(0,\ldots,2046)$ is simple and allows one to model errors for arbitrary vectors.

A caveat of the above approach is that we must know the range of values that the vector contains. This can be achieved by directly computing the min and max values for each vector. Alternatively, an approximate range can be determined if the "length" of vector is known, e.g., the two-norm. One weakness to the proposed approach is that we do not consider a flip in the accumulating sum, which we have left to future work.

5.3. Summary. We have shown that the range of values used in the dot product has a direct impact on the size of the errors that can be observed. A general rule in floating point algorithms has been to perform operations on numbers as close to the same magnitude as possible, as doing so minimizes the loss of precision. We have now shown that following this rule-of-thumb also gives the benefit of making bit upsets generate relatively small error when the numbers are no larger than one. We will show in § 6.1 how we utilize this concept of scaling to ensure that bit flips generate either small or very large error.

5.4. Vector and Matrix Norms. A fundamental tool in mathematical analysis is the concept of a norm, which provides a way to assign length (or size) to a vector. Using a norm, we are able to compare vectors, which allows us to bound the result of certain operations that utilize matrices and vectors.

5.4.1. Vector Norms. The Cauchy-Schwarz inequality states that

$$|\langle \mathbf{u}, \mathbf{v} \rangle| \leq \|\mathbf{u}\| \, \|\mathbf{v}\| , \tag{5.10}$$

which is an abstract result for any inner product space. In our analysis we consider only real-valued vectors, and so the bound becomes

$$|\mathbf{u} \cdot \mathbf{v}| \leq \|\mathbf{u}\|_2 \, \|\mathbf{v}\|_2 , \text{ for } \mathbf{u}, \mathbf{v} \in \mathbb{R}^n. \tag{5.11}$$

A consequence of the Cauchy-Schwarz inequality is the triangle inequality

$$\|\mathbf{u} + \mathbf{v}\| \leq \|\mathbf{u}\| + \|\mathbf{v}\| . \tag{5.12}$$

For real-valued vectors, the ℓ^2 norm $\|\cdot\|_2$ is defined to be

$$\|\mathbf{u}\|_2 = \sqrt{\mathbf{u} \cdot \mathbf{u}}, \tag{5.13}$$

which is often referred to as the Euclidean distance. Another norm we will utilize is the infinity norm $\|\cdot\|_\infty$, which in this context can be thought of as the 'max' of the vector

$$\|\mathbf{u}\|_\infty = \max\{|u_1|, \ldots, |u_n|\}. \tag{5.14}$$

The final norm we will utilize is the one-norm, which is commonly referred to as the taxicab norm or Manhattan distance.

$$\|\mathbf{u}\|_1 = \sum_{i=1}^{n} |u_i| \tag{5.15}$$

Because we consider only vectors that are in $\mathbb{R}^n$, these norms are equivalent in the following ways

$$\|\mathbf{u}\|_2 \leq \|\mathbf{u}\|_1 \leq \sqrt{n}\,\|\mathbf{u}\|_2\,, \tag{5.16}$$

$$\|\mathbf{u}\|_\infty \leq \|\mathbf{u}\|_2 \leq \sqrt{n}\,\|\mathbf{u}\|_\infty\,, \tag{5.17}$$

$$\|\mathbf{u}\|_\infty \leq \|\mathbf{u}\|_1 \leq n\,\|\mathbf{u}\|_\infty\,. \tag{5.18}$$

5.4.2. Matrix Norms. We also wish to make meaningful comparisons involving matrices. To do so we will use matrix analogs of the norms defined for vectors, referred to as induced norms. Suppose we have an $m \times n$ matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, then the infinity norm is defined as

$$\|\mathbf{A}\|_\infty = \max_{1\leq i\leq m} \sum_{j=1}^{n} \left|a_{ij}\right|, \tag{5.19}$$

which is maximum absolute row sum. The one-norm,

$$\|\mathbf{A}\|_1 = \max_{1\leq j\leq n} \sum_{i=1}^{m} \left|a_{ij}\right|, \tag{5.20}$$

is the maximum absolute column sum. The two-norm, also called the spectral norm, is defined to be

$$\|\mathbf{A}\|_2 = \sigma_{\max}(\mathbf{A}), \tag{5.21}$$

, where $\sigma_{\max}$ is the largest singular value of A. The Frobenius norm, which is not an induced norm, takes the form

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m} \left|a_{ij}\right|^2}. \tag{5.22}$$

We also will use the norm of a matrix-vector product, , e.g., $\|\mathbf{Ax}\|$. We have the following relation, were p denotes one of the norms defined above

$$\|\mathbf{Ax}\|_p \leq \|\mathbf{A}\|_p\,\|\mathbf{x}\|_p \tag{5.23}$$

Similar to vector norms, these norms are equivalent since we consider all values in $\mathbb{R}^{m\times n}$. We will use the following relations

$$\|\mathbf{A}\|_2 \leq \|\mathbf{A}\|_F \leq \sqrt{r}\,\|\mathbf{A}\|_2 \text{ for } r = \operatorname{rank}(\mathbf{A}), \tag{5.24}$$

$$\frac{1}{\sqrt{n}}\,\|\mathbf{A}\|_\infty \leq \|\mathbf{A}\|_2 \leq \sqrt{m}\,\|\mathbf{A}\|_\infty\,, \tag{5.25}$$

$$\|\mathbf{A}\|_2 \leq \sqrt{\|\mathbf{A}\|_\infty\,\|\mathbf{A}\|_1}. \tag{5.26}$$

6. GMRES. The Generalized Minimum Residual method (GMRES) of Saad and Schultz [17] is a Krylov subspace method for solving large, sparse, possibly nonsymmetric linear systems $Ax = b$. GMRES is based on the Arnoldi process [2], which can also be used to approximate a matrix's eigenvalues and eigenvectors. GMRES has the convenient property that the residual norm of the approximate solution at each iteration is monotonically nonincreasing, assuming correct arithmetic and storage. Its use of orthogonal projections and normalized (to length one) basis vectors also has advantages, that we will discuss below.

We begin this section by explaining how to use properties of the Arnoldi process to detect faults in an iteration of GMRES. We then apply the SDC models we developed above to show how to scale the linear system in a way that enhances fault detection and bounds the possible error of the major computational kernels. We will show in future work that these bounds by themselves do not suffice to bound the solution error. Nevertheless, they can, if one makes inexpensive changes to how GMRES computes the solution update coefficients.

6.1. Fault detection via projection coefficients. The norms and inner products that occur in each iteration of the Arnoldi process in GMRES have bounded absolute value. These bounds depend on the norm of the preconditioned matrix, which is inexpensive to estimate. We can use them to detect faults in all the major computational kernels in GMRES. Furthermore, we show that equilibrating or otherwise scaling the matrix so that its largest absolute value is one reduces the possible exponent range of these norms and inner products. This excludes more possible faults, thereby enhancing detection.

Algorithm 1 Restarted GMRES

1: **for all** $l = 1 to$ **do**
2: $\mathbf{r} \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}^{(j-1)}$
3: $\mathbf{q}_1 \leftarrow \mathbf{r} / \|r\|_2$
4: **for all** $j = 1 to restart$ **do**
5: $\mathbf{w}_0 \leftarrow \mathbf{A}\mathbf{q}_j$
6: **for all** $i = 1 to j$ **do**
7: $h_{i,j} \leftarrow \langle \mathbf{q}_i, \mathbf{w}_{i-1} \rangle$
8: $\mathbf{w}_i \leftarrow \mathbf{w}_{i-1} - h_{i,j}\mathbf{q}_i$
9: **end for**
10: $h_{j+1,j} \leftarrow \left\|\mathbf{w}_j\right\|_2$
11: $\mathbf{q}_{j+1} \leftarrow \mathbf{w}_j / h_{j+1,j}$
12: Find $\mathbf{y} = \min \left\|\mathbf{H}_j\mathbf{y} - \|\mathbf{b}\| \mathbf{e}_1\right\|_2$
13: Evaluate convergence criteria
14: Optionally, compute $\mathbf{x}_j = \mathbf{Q}_j\mathbf{y}$
15: **end for**
16: **end for**

6.1.1. Bounds on the Arnoldi Process. We start our analysis by bounding the dot product which determines the i-th upper Hessenberg entry, $h_{i,j}$ of the j-th Arnoldi iteration. The Arnoldi process is expressed on Lines 4–11 in Algorithm 1. At its core is the Modified Gram-Schmidt (MGS) process, which constructs a vector orthogonal to all previous basis vectors $\mathbf{q}_i$. The MGS process begins on Line 6 and completes on Line 9. To bound $h_{i,j}$ on Line 7, we will use the Cauchy-Schwarz inequality presented in Eq. (5.11) in conjunction with various norms presented in § 5.4. We begin with the vector which will become the $j+1$ orthonormal basis vector $\mathbf{w}_0$.

$$\|\mathbf{w}_0\| = \left\|\mathbf{A}\mathbf{q}_j\right\|$$
$$\|\mathbf{w}_0\| \leq \|\mathbf{A}\| \left\|\mathbf{q}_j\right\|$$

Our goal is to bound the length of $\mathbf{w}_0$, and as was stated when we presented the Cauchy-Schwarz inequality, the distance measure in $\mathbb{R}^n$ is the ℓ^2 norm. This leads to the following bound on the first intermediate vector

$$\begin{aligned} \|\mathbf{w}_0\|_2 &\leq \|\mathbf{A}\|_2 \left\|\mathbf{q}_j\right\|_2, \\ &\leq \sigma_{\max}(\mathbf{A}). \end{aligned} \tag{6.1}$$

We seek to bound the i-th entry of the upper Hessenberg on Line 7, which is bounded by

$$\left|h_{i,j}\right| \leq \|\mathbf{q}_i\| \|\mathbf{w}_{i-1}\|. \tag{6.2}$$

We know what $\mathbf{q}_i$ is a unit vector, e.g., $\|\mathbf{q}_i\|_2 = 1$, and therefore we only need to know a bound on the intermediate basis vector $\mathbf{w}_i$.

Recognizing that the Modified Gram-Schmidt process is isometric, that is it preserves the length of vectors, we have

$$\begin{aligned} \|\mathbf{w}_i\|_2 &\leq \|\mathbf{w}_0\|_2, \\ &\leq \sigma_{\max}(\mathbf{A}). \end{aligned} \tag{6.3}$$

Therefore, we may bound the upper Hessenberg entries by

$$\begin{aligned} \left|h_{i,j}\right| &\leq 1 \cdot \|\mathbf{w}_{i-1}\| \\ &\leq \sigma_{\max}(\mathbf{A}). \end{aligned} \tag{6.4}$$

The bound presented in Eq. (6.4) is crucial, as it demonstrates that the upper Hessenberg entries are bounded entirely by the input matrix. This means that if the matrix has a large maximum singular value, then the upper Hessenberg values can take large values. Moreover, we will now relate this bound to potential errors as illustrated in § 5.2.

6.1.2. Bound Application. We have shown what the theoretical upper limit is for the values in the upper Hessenberg. This essentially tells us what is *theoretically possible* inside the Arnoldi process, and we will use the theory that drives the algorithm to bound the error should a bit flip occur. Using this approach to construct an SDC detector is significant. By building a detection scheme in this way, we know precisely what errors we can detect, and more importantly we know what is not detectable. In § 5, we demonstrated how to analyze the data provided to the algorithm, and model the absolute error that bit flips could introduce should they impact that data. To begin, we will explain the relationship between the norm bounds presented in § 6.1.1 and § 5.

The important factor to keep in mind is, that exactly how an error is committed is irrelevant, the norm bounds allow us to filter out values that are invalid by theory. The intent of modeling the actual numerical errors that could be observed is because we wish to understand the distribution of possible errors — we either detect a large error or commit a small error.

6.1.3. Interpreting the Arnoldi Bounds. The bounds presented are theoretical, but we can draw several conclusions from the inherent properties of the Arnoldi process. Our goal is to translate the mathematics into information that can be used in our fault model.

1. The vectors $\mathbf{q}$ are unit vectors, and therefore have elements in the range $[0,1]$.
2. The two-norm presents an upper bound on the infinity norm, from Eq. (5.17). Which means we can state what the largest value of the intermediate basis vector $\mathbf{w}$ is, and construct the range of possible values to be $[0, \sigma_{\max}]$.

3. Rather than compute the largest singular value of $\mathbf{A}$, we can use upper bounds on the two-norm of a matrix, e.g., Eq. (5.24) and Eq. (5.26).

The upper Hessenberg matrix $\mathbf{H}$ is what drives the subsequent step of the GMRES algorithm, which is to solve a projected least squares problem, and ultimately compute a solution $\mathbf{x}$. Based on our premise that a single bit flip occurs, should this bit flip impact the MGS process, then we can use our bound on the Hessenberg entries to understand the types of error we can expect in the least squares problem. We will address the least squares problem further in § 6.2.

6.1.4. Error Detection. In the context of error detection, we can only detect an error that exceeds the bound on the upper Hessenberg entry h_{ij}. To do this, we insert a conditional between Lines 7 and 8, and test whether $|h_{ij}| \leq \sigma_{\max}$. Should this condition be invalid, then we assume that we have committed an error at some point, and we can halt the algorithm. If we use the sandbox model with FT-GMRES, we need not halt and can choose whether to reject this inner iteration and start a new inner solve.

6.1.5. Model Evaluation and Potential Errors. We now use our fault model, in conjunction with two example linear systems to demonstrate how:

1. Data inputs impact the range of detectable values.
2. Data scaling impacts the distribution of possible errors.

In the following examples, we have implemented Algorithm 1, and instrumented the code to compute the actual range of values present in vectors. From these ranges we computed all possible bit flip perturbations in the biased exponent before the dot product that computes the upper Hessenberg entries. We then classify the absolute error that would be committed in the dot product into four classes:

1. Absolute error less than 1.0
2. Absolute error greater than or equal to 1.0, but less than or equal to $\|\mathbf{A}\|_2$.
3. Absolute error greater $\|\mathbf{A}\|_2$.
4. Error that is non-numeric, e.g., Inf or NaN.

Classes 1 and 2 are *undetectable*, while Classes 3 and 4 are detectable. Our goal is to ensure that should a bit flip, the error falls into Classes 1, 3, and 4. While minimizing or eliminating the occurrence of Class 2 errors. We refer to Class 2 errors as the *grey area*, as they are undetectable errors that we consider to be large.

6.1.6. Sample Problems. We have chosen two sample matrices to demonstrate our technique. To ensure reproducability, we did not create either of these matrices from scratch, rather we used readily available matrices. The first matrix is fairly common, and arises from the finite difference discretization of the Poisson equation. This matrix is symmetric and positive definite, meaning that it could be solved using the Conjugate Gradient method. We generated this matrix using Matlab's built-in Gallery functionality. The second matrix chosen presents a more realistic linear system. The CoupCons3D matrix comes from the University of Florida Sparse Matrix Collection [8]. It arises from a fully coupled poroelastic problem. The matrix is symmetric in pattern, but not symmetric in values. The system is also not positive definite, meaning Conjugate Gradient could not be used to solve the system. The matrix is also fairly large, and has zero values explicitly stored. The system is also poorly scaled, with a mixture of large and small values. We have summarized the characteristics of each matrix in Table 6.1.

One of our findings from § 4 is that scaling can have a direct impact on the range of possible errors. To investigate this, we will scale the input matrix and right-hand side vector such that the matrix is equilibrated. To *scale* a matrix, we use a sparse matrix implementation of LAPACK's equilibration routine DGEEQU [1]. Equilibration does not cause fill. Table 6.2 summarizes the norms for each of our test matrices.

TABLE 6.1
Sample Matrices

Properties	Poisson Equation	CoupCons3D
number of rows	10,000	416,800
number of columns	10,000	416,800
nonzeros	49,600	17,277,420
structural full rank?	yes	yes
explicit zero entries	0	5,044,916
nonzero pattern symmetry	symmetric	symmetric
type	real	real
structure	symmetric	unsymmetric
positive definite?	yes	no

TABLE 6.2
Norms of Sample Matrices

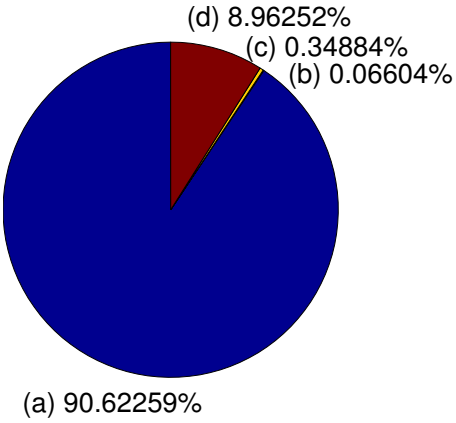
Norm	Poisson Equation		CoupCons3D	
	No Scaling	Scaling	No Scaling	Scaling
$\|\mathbf{A}\|_\infty$	8.0	2.0	1.30×10^6	1.0
$\|\mathbf{A}\|_2$	7.999	1.999	1.20×10^6	1.0
$\|\mathbf{A}\|_F$	4.46×10^2	1.12×10^2	2.75×10^6	2.91×10^2

6.1.7. Results. We ran Algorithm 1 for 1000 total iterations, using a restart value of 25. By instrumenting the code, we determined the numerical range of values each vector contained, and then computed the possible absolute error that a bit flip could introduce. We classified the absolute error according to § 6.1.5, and counted each class of error for the duration of the algorithm.

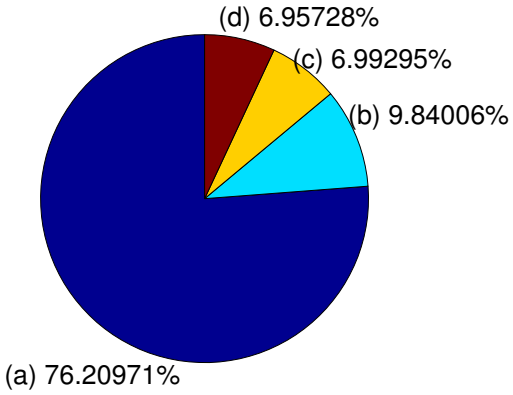
The pie charts presented in Figure 6.1 are not probabilistic, e.g., they do not convey the likelihood of observing such an error. Rather, these charts characterize the possible errors when given specific data. Consider an arbitrary length vector $\mathbf{x}$, we can determine the range of values in the vector, e.g., $x_i \in [a,b]$, but we do not know how many of each value, or in what order they occur. Obtaining fine-grained statistics would involve evaluating every element of the vector, or constructing a probabilistic model that captures the distribution of values in each vector.

Since we consider the impact of a single bit flip, it is sufficient to follow the methodology presented in § 5. That is, we may not know the distribution and order of numbers in the vectors, but we can model every possible error, by assuming that each value in the interval, *could* be used in an operation with *every* value of the other interval. This Cartesian product (or outer product) guarantees that we have counted all possible errors for IEEE-754 double precision numbers in an interval, including errors that may not occur because the vector does not contain that specific number, or because the ordering.

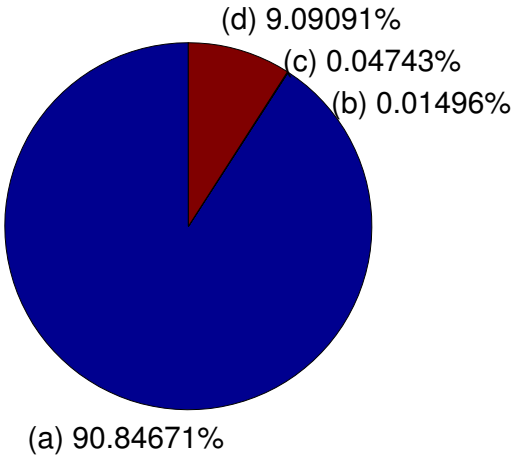
What these results indicate is that there is a clear benefit to having good scaling. We can not enforce this, since some linear systems may be inherently poorly scaled. We can advocate that scaling, while typically used to improve numerical stability and reduce the loss of precision, can benefit fault resilience. We can also conclude that the upper Hessenberg entries are a viable location to perform fault detection, and this is particularly true when data is scaled, since it reduces the number of errors that fall into the undetectable and large region.



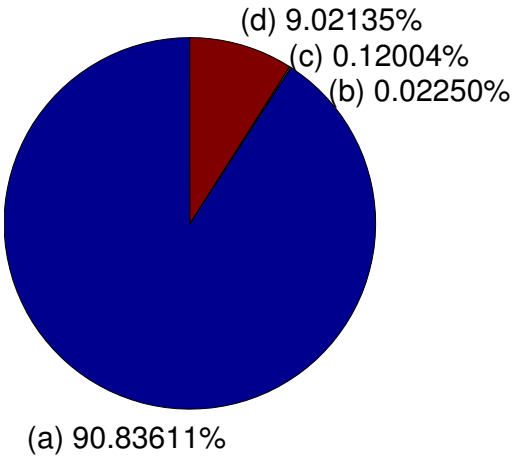
(a) Poisson, no scaling.



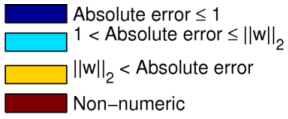
(b) CoupCons3D, no scaling.



(c) Poisson, equilibrated.



(d) CoupCons3D, equilibrated.



(e) Legend.

FIG. 6.1. *Number of possible errors falling into (a) Class 1: err* < 1.0 *, (b) Class 2:* $1.0 \geq err \leq \|\mathbf{A}\|_2$*, (c) Class 3:* $\|\mathbf{A}\|_2 > err$*, and (d) Class 4: Non-numeric*

6.2. Inexpensive robustness improvements. In this section, we show that if the errors in the major computational kernels of a single iteration of GMRES are bounded, then simple, inexpensive changes to GMRES can make the resulting solution error bounded as well. In particular, we need to ensure that as long as the error in each of the entries of the last column of the upper Hessenberg matrix is bounded, the resulting solution of the projected least-squares problem is bounded. The conventional way to solve this least-squares

problem is first to use Givens rotations to reduce the upper Hessenberg matrix to upper triangular, and then to use backward substitution to solve the resulting upper triangular system. This can result in unbounded or even undefined solution coefficients if the upper triangular matrix is singular or nearly so. However, we believe that the minimum-norm least-squares solution to that triangular system is bounded with respect to its input, if one truncates zero or small singular values. We plan to prove this in future work. If our assertion is true, then it ensures boundedness of the solution coefficients with respect to the condition number of the truncated system. Since GMRES computes the update to the solution vector as a linear combination of the basis vectors, and since the basis vectors are all unit length and linearly independent, this means that the effects of an error on the solution are also bounded.

Truncation of zero or small singular values is a particular kind of *regularization*. This regularization of the projected least-squares problem requires computing a rank-revealing decomposition. It's possible to update a rank-revealing decomposition of a $k+1$ by k matrix (when adding a new column and row) in $\Theta(k^2)$ time [22]. This is asymptotically no more computation than would be required to solve the least-squares problem anyway. Furthermore, a rank-revealing decomposition of the upper Hessenberg matrix is useful for Flexible GMRES in detecting abnormal conditions that can arise even with correct arithmetic and storage [5].

6.2.1. Future Work. Our next step is to prove that bounded errors in an iteration of the Arnoldi process translate into

1. bounded errors in that GMRES iteration, that
2. do not cause unbounded errors in subsequent iterations.

We outline in the discussion above how to show the first of these assertions. It will depend on a slight change to GMRES, in order to regularize the least-squares problem used to solve for the solution update coefficients. We will begin proving the second assertion by showing that the corrupted Arnoldi process is still either an *oblique projection method* (in the sense of Saad [16, Chapter 5]), or what we call an *inexact oblique projection method*, in which "oblique" has the sense above, and "inexact" has the sense of "inexact Krylov methods." We will use this, in combination with GMRES' "memory" of previous correct iterations, to show that the additional corrupted basis vectors and projection coefficients cannot make the residual norm grow by more than a certain bound more than its value in the last correct iteration.

The above assertions, if true, ensure that bounded errors in the Arnoldi process translate into bounded GMRES solution errors. This will give us the piece missing from our previous work on FT-GMRES [5]: namely, that the outer solver may need to bound inner solves' errors in order to ensure convergence, as governed by inexact Krylov theory [20]. FT-GMRES assumes that the outer solver computes reliably, so it can evaluate each inner solve's residual norm and use the inexact Krylov bound as an acceptance criterion for that inner solve. If an inner solve result does not satisfy the bound, the outer solver will repeat the inner solve. However, this is a very coarse-grained error detection mechanism, given that FT-GMRES expects the inner solve to consume most of the iterations and do most of the work. Rolling back an entire inner solve can be expensive. Suppose instead that we could *detect* most errors in the inner GMRES solve, using the criteria developed earlier in this paper, and *bound* the errors that we could not detect. If we could do this, then we might only ever need to roll back a single iteration of the inner solver. We might not even need to do this, if we apply inexact Krylov bounds to individual iterations of the *inner* solver, as well as to the outer solver. If we succeed in this task, we could finally make good on our promise to deliver an iterative linear solver that *always converges* in reasonable time despite faults, as long as it would have converged in reasonable time without faults.

7. Conclusions. The key findings we have presented are

- The semantics of a bit flip provides useful insight into how the inputs of the GMRES algorithm

relate to the potential errors that a bit flip in data can produce.
- Studying the effect of a single bit flip can provide enormous insight into how algorithms behave in the presence of faulty hardware, and produces findings that are not based on conjectures about failure rates that are not measurable.

REFERENCES

[1] E. ANDERSON, Z. BAI, C. BISCHOF, S. BLACKFORD, J. DEMMEL, J. DONGARRA, J. D. CROZ, A. GREENBAUM, S. HAMMARLING, A. MCKENNEY, AND D. SORENSEN, *LAPACK Users' Guide*, SIAM, Philadelphia, PA, USA, third ed., 1999.

[2] W. E. ARNOLDI, *The principle of minimized iterations in the solution of the matrix eigenvalue problem*, Quarterly of Applied Mathematics, 9 (1951), pp. 17–29.

[3] K. ASANOVIC, R. BODIK, B. C. CATANZARO, J. J. GEBIS, P. HUSBANDS, K. KEUTZER, D. A. PATTERSON, W. L. PLISHKER, J. SHALF, S. W. WILLIAMS, AND K. A. YELICK, *The Landscape of Parallel Computing Research: A View from Berkeley*, Tech. Rep. UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec 2006. See also the journal article Asanovic et al. [4].

[4] K. ASANOVIC, R. BODIK, J. W. DEMMEL, T. KEAVENY, K. KEUTZER, J. KUBIATOWICZ, N. MORGAN, D. A. PATTERSON, K. SEN, J. WAWRZYNEK, D. WESSEL, AND K. A. YELICK, *A View of the Parallel Computing Landscape*, Communications of the ACM, 52 (2009), pp. 56–67.

[5] P. G. BRIDGES, K. B. FERREIRA, M. A. HEROUX, AND M. HOEMMEN, *Fault-tolerant linear solvers via selective reliability*, ArXiv e-prints, (2012).

[6] P. G. BRIDGES, M. HOEMMEN, K. B. FERREIRA, M. A. HEROUX, P. SOLTERO, AND R. BRIGHTWELL, *Cooperative application/OS DRAM fault recovery*, in Proceedings of the 2011 International Conference on Parallel Processing - Volume 2, Euro-Par'11, Berlin, Heidelberg, 2012, Springer-Verlag, pp. 241–250.

[7] G. BRONEVETSKY AND B. DE SUPINSKI, *Soft error vulnerability of iterative linear algebra methods*, in Proceedings of the 22nd annual international conference on Supercomputing, ICS '08, New York, NY, USA, 2008, ACM, pp. 155–164.

[8] T. A. DAVIS AND Y. HU, *The University of Florida Sparse Matrix Collection*, ACM Transactions on Mathematical Software, 38 (2011), pp. 1:1–1:25. `http://www.cise.ufl.edu/research/sparse/matrices`.

[9] I. S. HAQUE AND V. S. PANDE, *Hard data on soft errors: A large-scale assessment of real-world error rates in GPGPU*, in Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, CCGRID '10, Washington, DC, USA, 2010, IEEE Computer Society, pp. 691–696.

[10] *IEEE standard for floating-point arithmetic*, IEEE Std 754-2008, (2008), pp. 1–70.

[11] INTEL, *FDIV replacement program: Description of the flaw*, July 2004. Available online: `http://support.intel.com/support/processors/pentium/sb/CS-013007.htm` [last accessed 04 Sep 2013].

[12] T. KARNIK, P. HAZUCHA, AND J. PATEL, *Characterization of soft errors caused by single event upsets in CMOS processes*, IEEE Trans. Dependable Secur. Comput., 1 (2004), pp. 128–143.

[13] P. M. KOGGE ET AL., *ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems*, Tech. Rep. TR-2008-13, University of Notre Dame CSE Department, September 2008.

[14] S. MICHALAK, A. DUBOIS, C. STORLIE, H. QUINN, W. RUST, D. DUBOIS, D. MODL, A. MANUZZATO, AND S. BLANCHARD, *Assessment of the impact of cosmic-ray-induced neutrons on hardware in the Roadrunner supercomputer*, Device and Materials Reliability, IEEE Transactions on, 12 (2012), pp. 445–454.

[15] N. MISKOV-ZIVANOV AND D. MARCULESCU, *Soft error rate analysis for sequential circuits*, in Proceedings of the Conference on Design, Automation and Test in Europe, DATE '07, San Jose, CA, USA, 2007, EDA Consortium, pp. 1436–1441.

[16] Y. SAAD, *Iterative Methods for Sparse Linear Systems*, SIAM, Philadelphia, PA, USA, second ed., 2003.

[17] Y. SAAD AND M. H. SCHULTZ, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 856–869.

[18] M. SHANTHARAM, S. SRINIVASMURTHY, AND P. RAGHAVAN, *Characterizing the impact of soft errors on iterative methods in scientific computing*, in Proceedings of the international conference on Supercomputing, ICS '11, New York, NY, USA, 2011, ACM, pp. 152–161.

[19] ———, *Fault tolerant preconditioned conjugate gradient for sparse linear system solution*, in Proceedings of the 26th ACM international conference on Supercomputing, ICS '12, New York, NY, USA, 2012, ACM, pp. 69–78.

[20] V. SIMONCINI AND D. B. SZYLD, *Theory of inexact Krylov subspace methods and applications to scientific computing*, SIAM J. Sci. Comput., 25 (2003), pp. 454–477.

[21] J. SLOAN, R. KUMAR, AND G. BRONEVETSKY, *Algorithmic approaches to low overhead fault detection for sparse linear algebra*, in Proceedings of the 2012 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), DSN '12, Washington, DC, USA, 2012, IEEE Computer Society, pp. 1–12.

[22] G. W. STEWART, *Updating a rank-revealing ULV decomposition*, SIAM J. Matrix Anal. Appl., 14 (1993), pp. 494–499.

INVESTIGATING ITERATIVE METHODS FOR SYSTEMS WITH MULTIPLE RIGHT-HAND SIDES USING GRAPHICS PROCESSING UNITS

SARAH V. OSBORN* AND ERIC T. PHIPPS[1]

Abstract. Numerically solving large, sparse linear systems is of great importance in many applications and often a great computational bottleneck. In this paper, we investigate the potential improvement in performance of solving linear systems on different threaded architectures, in particular graphics processing units (GPUs). GPUs are an important component in high-performance computing and allow for great parallelism which preconditioned Krylov iterative solvers can take advantage. In this paper, we discuss some challenges in solving linear systems with multiple right-hand sides. Then results will be presented for performing solves using existing linear solvers and preconditioners from the Trilinos Project on CPU and GPU architectures.

1. Introduction. The numerical solution of large, sparse linear systems is fundamental for many applications in computational science and engineering. This often requires a significant computational effort so finding faster, more efficient ways to solve $A\mathbf{x} = \mathbf{b}$ allows for the solution of larger and more complex problems. Of particular interest in this paper is the solution of $A\mathbf{X} = \mathbf{B}$ where $\mathbf{X}$ and $\mathbf{B}$ are multi-vectors. Using preconditioned block iterative methods as solvers allows for the potential to speed-up the process by exploiting the parallelism of these methods on parallel computer platforms like multicore-CPU and manycore-accelerator (e.g., NVIDIA GPU) devices. In order to take advantage of the potential improvement in performance, careful consideration of algorithms and implementation is essential. In this paper, we first consider the CUSP library for sparse linear algebra and implement our own kernels for performing block solves for multiple right-hand sides on a GPU using the CUDA platform. Of particular interest are kernels adapted to a large number of right-hand sides. This problem arises when using a mean-based preconditioner in the stochastic Galerkin method [13]. Then, we use existing CPU-based solvers and preconditioners in Trilinos and port them to GPU architectures and present timing results. We present an overview of iterative methods and preconditioners for solving $A\mathbf{X} = \mathbf{B}$ in Section 2. Implementation of the necessary routines in CUSP and Trilinos is discussed in Section 3 with particular focus on parallel programming using CUDA. Numerical results are given in Section 4.

In this paper, except when otherwise specified, upper case letters (A, B, etc.) will denoted matrices, lower case bold letters ($\mathbf{x}$, $\mathbf{y}$, etc.) will denote vectors, and upper case bold letters ($\mathbf{X}$, $\mathbf{Y}$, etc.) will denote multi-vectors.

2. Preconditioned Iterative Methods. We consider solving the linear system $A\mathbf{X} = \mathbf{B}$ where A is a symmetric, positive definite (SPD) matrix and $\mathbf{X}$ and $\mathbf{B}$ are the solution and right-hand side multi-vectors. In general, an iterative method starts with an initial guess and performs a series of steps to find more accurate approximations to the solution. We focus on the conjugate gradient (CG) method due to the structure of the matrix. The convergence of an iterative method depends on the spectrum of A and can be significantly improved with preconditioning. The basic idea of preconditioning is to transform a linear system $A\mathbf{x} = \mathbf{b}$ into an equivalent linear system $\tilde{A}\mathbf{x} = \tilde{\mathbf{b}}$ which will have the same solution as the original system. The transformation is chosen so that $\tilde{A}$ will have better convergence properties than A resulting in faster convergence. Typically, this is done by the multiplication of the inverse of a non-singular matrix M so then $\tilde{A} = M^{-1}A$ and $\tilde{\mathbf{b}} = M^{-1}\mathbf{b}$. Another way to precondition is to define $\tilde{A}\mathbf{x} = \tilde{\mathbf{b}}$ to be a simplified version of the original system.

Since we are solving for multiple solution vectors, we consider the pseudo-block CG algorithm which applies the standard single-vector CG algorithm simultaneously to multiple right-hand sides [12]. The pseu-

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

docode for the preconditioned pseudo-block CG method is shown in Algorithm 2 where $\mathbf{z} =< \mathbf{X}, \mathbf{Y} >$ is the multi-vector dot product defined by $\mathbf{z}[i] = \mathbf{X}[i]^T \mathbf{Y}[i]$ where $\mathbf{X}[i]$ is the i-th column of $\mathbf{X}$. The multiplication and division operators are performed component-wise.

Algorithm 2 Pseudo-block preconditioned CG $(A, \mathbf{B}, \mathbf{X}_0)$

$\mathbf{R}_0 = \mathbf{B} - A\mathbf{X}_0$
Solve $M\mathbf{Z}_0 = \mathbf{R}_0$
$\mathbf{P}_0 \leftarrow \mathbf{Z}_0$
for all $i \leftarrow 1, 2, \ldots$ until converged **do**
  $\alpha_i = \frac{<\mathbf{R}_i, \mathbf{Z}_i>}{<\mathbf{P}_i, A\mathbf{P}_i>}$
  $\mathbf{X}_{i+1} \leftarrow \mathbf{X}_i + \alpha_i \mathbf{P}_i$
  $\mathbf{R}_{i+1} \leftarrow \mathbf{R}_i - \alpha_i A\mathbf{X}_i$
  Solve $M\mathbf{Z}_{i+1} = \mathbf{R}_{i+1}$
  $\beta_i = \frac{<\mathbf{R}_{i+1}, \mathbf{Z}_{i+1}>}{<\mathbf{R}_i, A\mathbf{Z}_i>}$
  $\mathbf{P}_{i+1} \leftarrow \mathbf{R}_{i+1} + \beta_i \mathbf{P}_i$
end for

Now as for the choice of the preconditioner M, we consider three different preconditioning strategies: incomplete factorizations, polynomial preconditioners, and multigrid preconditioners.

2.0.2. Incomplete Factorizations. An incomplete LU factorization (ILU) preconditioner is based on Gaussian elimination which given a matrix A yields an upper-triangular matrix U and a lower-triangular matrix L so that $A = LU$ [14]. The key idea in an incomplete factorization is to produce approximate $\tilde{L}$ and $\tilde{U}$ by discarding some of the fill-in which often takes place during the factorization so then $A \approx \tilde{L}\tilde{U}$. An application of an ILU preconditioner to a multi-vector involves solving two triangular systems with multiple right-hand sides.

2.0.3. Polynomial Preconditioner. A polynomial preconditioner calculates a series of matrix-valued polynomials which approximates the inverse of a matrix [14]. For Chebyshev preconditioning, a linear combination of Chebyshev polynomials are used as the preconditioner matrix. In contrast to preconditioning methods based on Gaussian elimination, such as ILU from above, the Chebyshev method only uses matrix-multi-vector multiplication and multi-vector addition.

2.0.4. Algebraic Multigrid Overview. Algebraic multigrid methods are an effective preconditioning strategy that construct a hierarchy of grids and intergrid transfer operators without explicit knowledge of the underlying problem [5, 14]. A typical multigrid cycle starts at a finest level where the fine level solution vector is then transferred to a next coarser level, called restriction. After some relaxation cycles on the coarse level, the solution is then restricted to the next coarser level until the coarsest level is reached. The solution obtained at the coarsest level is then interpolated back to the finer level, called prolongation. The solution from this finer level is interpolated to next finer level after some relaxation iterations. The basic idea is to damp high-energy components which are reduced through a simple smoothing procedure, then transfer the low-energy (or smooth) components to a coarser level where they are likewise reduced.

The setup phase involves restricting the original matrix A to a coarser (smaller) matrix by performing the triple-matrix product $A_{l+1} = R_l A_l P_l$ where P_l is the interpolation operator, R_l is the restriction operator, and A_l is the representation of the matrix operator for a particular level l where $A_0 = A$. In this work, the setup of the preconditioner is done on the CPU and is not discussed in detail. Instead we focus on the apply phase of the preconditioner. The multigrid cycling or solve phase is detailed in Algorithm 3 where m is

the total number of levels and l is the current level. Several computations are executed at each level in the algorithm, but the operations are largely sparse matrix- multi-vector multiplications.

Algorithm 3 AMGCycle$(A, \mathbf{X}, \mathbf{B}, l, m)$

if $l == m$ **then**
  Solve $A\mathbf{X} = \mathbf{B}$ using coarse grid solver
else
  $\mathbf{X} \leftarrow Presmooth(A, \mathbf{X}, \mathbf{B})$
  $\mathbf{R} = \mathbf{B} - A\mathbf{X}$
  $\mathbf{E} = 0$
  AMGCycle$(A_{l+1}, \mathbf{E}, R_{l+1}\mathbf{R}, l+1, m)$
  $\mathbf{X} = \mathbf{X} + P_{l+1}\mathbf{E}$
  $\mathbf{X} \leftarrow Postsmooth(A, \mathbf{X}, \mathbf{B})$
end if

There are many options for smoother approaches: Gauss-Seidel, incomplete factorizations, polynomial smoothers, etc. We choose to use a polynomial smoother as it is implemented almost entirely with sparse matrix - multi-vector products. In particular, we use a Chebyshev iteration as the smoother in our implementation. For the coarsest grid solver, a direct method is often used like a LU factorization or an iterative method, but another smoother can be used or an incomplete factorization. In our implementation, we use the Chebyshev iteration as the coarse grid solver. Once the multigrid hierarchy is formed, to apply it as a preconditioner for each level l the matrices R_l, A_l, and P_l are necessary as well as the smoothers for each level and the coarse grid solver for the final level.

3. Software Implementation. Our goal was to test the performance of pseudo-block CG preconditioned with AMG on GPU architectures. Existing software implementations were used with iterative linear solver algorithms and preconditioners as a starting point. In this section, we first describe some basic terminology and give an overview of programming using CUDA [10], a parallel computing platform and programming model invented by NVIDIA. Then we explain our implementation in CUSP [4]. CUSP is a library of parallel algorithms for sparse linear algebra on CUDA GPU architectures. Next, we move to the Trilinos framework and use existing software for the preconditioned CG and port it to a GPU.

3.1. Overview of Parallel Programming with CUDA. In the CUDA parallel programming model [9, 10], an application consists of a sequential host program that may execute parallel programs known as *kernels* on a parallel device. A kernel is executed using a potentially large number of parallel threads where each thread runs the same sequential program. The programmer organizes the threads of a kernel into a grid of thread blocks. The threads of a given block can cooperate among themselves using barrier synchronization and a per-block shared memory space that is private to that block. To manage this large population of threads efficiently, the GPU employs a SIMT (Single Instruction Multiple Thread) architecture in which the threads of a block are executed in groups of 32 called *warps*. A warp executes a single instruction at a time across all its threads.

In order to write efficient kernels for our applications, memory access is an important factor. When accessing a portion of global memory, all threads in a warp access a bank of memory at one time. If the locations of the global memory are sufficiently close together, the device *coalesces* all memory accesses into a consolidated access to consecutive memory locations. When concurrent threads simultaneously access memory addresses that are very far apart in physical memory, the device is unable to combine the accesses which leads to non-coalesced memory access. This results in a decrease to memory bandwidth efficiency

and therefore a decrease to the overall performance of the kernel and should be avoided if possible.

3.2. Implementation of Sparse Linear Algebra Routines on a GPU in CUSP. CUSP provides a framework for the CG method preconditioned with AMG for a single right-hand side on a GPU. In order to implement pseudo-block CG and to apply the preconditioner to a linear system with multiple right-hand sides, three main kernels are necessary: sparse matrix times a multi-vector (SpM-MV), dot product of two multi-vectors, and scalar multiplication and addition of multi-vectors. With these kernels in place, implementing our method simply involves adding the pseudo-block CG algorithm code and an apply function for the preconditioner. Special consideration must be made when implementing SpM-MV on a GPU as this operation is necessary in the solver and preconditioner and ultimately determines the overall performance of a method. One SpM-MV involves computing the dot products of each row of the sparse matrix with each column of the multi-vector.

There are many different sparse matrix representations, but we will only consider the *compressed row storage* (CRS) format [3]. This format stores the column indices and nonzero values in arrays with a third array of row pointers which stores the locations of the value array that start a row. This is an efficient format in that it does not store any unnecessary elements, but needs an indirect addressing step for vector/multi-vector element access.

First, let us consider a SpM-MV but with a single column in the multi-vector. One possible implementation is to assign one thread per matrix row to compute the dot product. This kernel suffers from lack of performance due to the way it accesses the data in the CRS matrix. The column indices and nonzero values for a given row are stored contiguously in the CRS data structure, but these values are not accessed by consecutive threads. This leads to uncoalesced memory access causing poor performance. A better implementation assigns a warp to each matrix row so threads access the data in the CRS matrix format contiguously, called the vector kernel in [3]. There is still inefficiency in this kernel due to the uncoalesced vector accesses.

Since we are interested in an efficient kernel for a large number of right-hand sides, let us consider a multi-vector with a large number of columns and investigate accessing the elements in the multi-vector. The multi-vector is stored as a one-dimensional array with the entries stored in a row-wise form. Since the matrix is sparse and the multi-vector is only accessed when the matrix has a nonzero, we don't have coalesced memory access for the multi-vector using the above approach. To correct this we have consecutive threads access consecutive columns for a given row of the multi-vector.

Let us consider the kernel for a particular thread block. Each thread block iterates over the matrix rows assigned to that block processing seven rows in parallel for each iteration. In order to maintain coalesced reads of the matrix values, shared memory is used to store the values and column indices for each row. Then the matrix row and multi-vector dot product is computed in parallel with each thread in a warp accessing a different column of the multi-vector, processing 128 columns at a time. It should be noted that due to the number of columns processed in parallel our kernel is not optimized for fewer columns in the multi-vector.

Timing results comparing our kernel and CUSP's vector kernel for a single vector are shown in Figure 3.1. Our experiments show that with large number of right-hand sides we achieve significantly higher floating point throughput over the vector kernel processing each column in the multi-vector serially. The other kernels that were implemented on the GPU, multi-vector dot-product and scalar multiplication and addition of multi-vectors, were implemented in a similar way taking the same considerations of coalesced memory access and distribution of warps into account.

3.3. Trilinos. Now, we consider using the Trilinos Project to solve a linear system on a GPU [8]. Currently, matrix and preconditioner assembly are not available on a GPU. Instead, our strategy is to assemble the linear problem (matrix and right-hand side) and the preconditioner on the CPU in serial. Then the as-

sembled items are transfered to the GPU and the linear algebra kernels that are available on the GPU are used to compute the solution.

Trilinos' ability to support multicore computing is provided by the Kokkos shared-memory API [6]. The software implementations of the linear algebra algorithms and necessary data objects are templated on a Kokkos node type. This allows a single code-base to support serial, threaded, CPU or GPU, and hybrid architectures. The node type specifies how to transfer data in memory as well as an abstraction for how the parallel architecture works. Tpetra is used for the linear algebra objects [1]. Belos' pseudo-block CG was used as the iterative solver method [2]. The MueLu package provided the algebraic multigrid which in turn uses Ifpack2's Chebyshev iteration as a smoother and coarse grid solver [7].

Once the linear problem and preconditioner are assembled on the CPU, both must be transferred to the GPU. To accomplish this, we implemented a clone function to copy all necessary data to the new node type (i.e. the GPU). In order to clone a matrix, vector, or multi-vector to the GPU, an exact copy of the data is made on the device. To clone a multigrid preconditioner, we must clone the restrictor (R), prolongator (P), and coarse grid (A) matrices for each level of the muligrid hierarchy as well as the smoother (pre and/or post) and coarse grid solver. To clone an ILU preconditioner, we perform the factorization of the coarse grid on the CPU first and then clone the factors (stored as matrices) to the new node type. For the smoother, we use a Chebyshev iteration and all necessary data is copied to the GPU when it is cloned.

After the linear problem and preconditioner are cloned to the GPU, all that is necessary to solve on a GPU is the appropriate linear algebra routines (most importantly matrix-vector (or multi-vector) multiply, matrix-matrix multiplication, dot products, etc.). Kokkos provides the flexibility to specify the linear algebra kernels to be used for a particular node type. In our implementation, the NVIDIA CUDA Sparse Matrix library (CUSPARSE) is used for the linear algebra kernels on the GPU [11].

4. Numerical Results. In this section we present results to show the performance of solving linear systems arising from the finite difference discretization of a linear two-dimensional diffusion equation with varying number of mesh points. Pseudo-block CG was used as the solver with a convergence tolerance of 10^{-12}, preconditioned with smoothed aggregation algebraic multigrid. A Chebyshev polynomial smoother was used as the smoother and coarse grid solver with polynomial degree of 5. The CPUs used for the testing were two Intel Xeon E5-2670 CPUs with 8 cores per CPU and the GPU was a Tesla K20Xm of compute capability 3.5. For each problem size, the solver ran on the CPU with 1 core and with 16 cores using MPI parallelism, on a single GPU, and on two GPUs. The timings for the matrix-vector multiply, preconditioner apply, and the total solve are shown in Figures 4.1 - 4.3. The number of mesh points in each direction is 2^N yielding 2^{2N} degrees-of-freedom.

The most useful comparison is the timing results for the CPU with 16 cores and the single GPU. For the matrix-vector multiplication, the GPU outperforms the CPU with 16 cores when the mesh size reaches 128 and for the preconditioner apply, it is faster when the mesh size reaches 2048.

Currently, solving with 2 GPUs is hindered by the large amount of transfer time between the CPU and the GPU. This is due to known inefficiencies in the implementation where the entire multi-vector is copied between the CPU and GPU each time communication between GPUs is required. This is demonstrated in Figure 4.3 plotting the transfer time between the CPU and GPU.

5. Conclusions. In this paper, we investigated linear solver performance on a CPU and a GPU. This was first accomplished using CUSP where we implemented pseudo-block CG for solving a system with multiple right-hand sides. Several linear algebra kernels were written for the GPU. It should be noted that these kernels are not fully optimized particularly for smaller numbers of right-hand sides. Then we moved to the Trilinos framework where we cloned a linear problem and an existing multigrid preconditioner to a different node type, specifically a GPU. We presented some numerical results that show promising results for

solving linear systems with a GPU and multi-GPU scheme. There is still work to be done to more efficiently implement a multi-GPU solver scheme, yet when those issues are resolved the results look very promising.

In the future, we would like to implement the linear algebra kernels that we implemented in CUSP into the Trilinos framework to be used on the GPU.

REFERENCES

[1] C. BAKER AND M. HEROUX, *Tpetra, and the use of generic programming in scientific computing*, Scientific Programming, 20 (2012), pp. 115–128.

[2] 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.

[3] N. BELL AND M. GARLAND, *Implementing sparse matrix-vector multiplication on throughput-oriented processors*, in SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, New York, NY, USA, 2009, ACM, pp. 1–11.

[4] N. BELL AND M. GARLAND, *CUSP: Generic parallel algorithms for sparse matrix and graph computations*, 2012. Version 0.3.0.

[5] W. L. BRIGGS, V. E. HENSON, AND S. F. MCCORMICK, *A Multigrid Tutorial, Second Edition*, SIAM, Philadelphia, 2000.

[6] H. C. EDWARDS, D. SUNDERLAND, V. PORTER, C. AMSLER, AND S. MISH, *Manycore performance-portability: Kokkos multidimensional array library*, Scientific Programming, 20 (2012), pp. 89–114.

[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] 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.

[9] D. B. KIRK AND W. MEI W. HWU, *Programming Massively Parallel Processors*, Morgan Kaufmann Publishers, Burlington, MA, 2010.

[10] NVIDIA, *CUDA C programming guide*, 2013. Version 5.5.

[11] ———, *CUSPARSE library*, 2013. Version 5.5.

[12] D. P. O'LEARY, *The block conjugate gradient algorithm and related methods*, Linear Algebra and Its Applications, 29 (1980), pp. 293–322.

[13] C. E. POWELL AND H. C. ELMAN, *Block-diagonal preconditioning for spectral stochastic finite-element systems*, IMA Journal of Numerical Analysis, 29 (2009), pp. 350–375.

[14] Y. SAAD, *Iterative Methods for Sparse Linear Systems*, SIAM, Philadelphia, 2 ed., 2003.

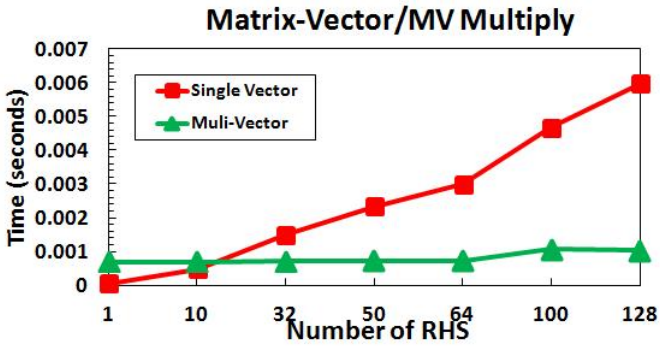


FIG. 3.1. *Timing results for matrix-vector and matrix-multi-vector multiply*

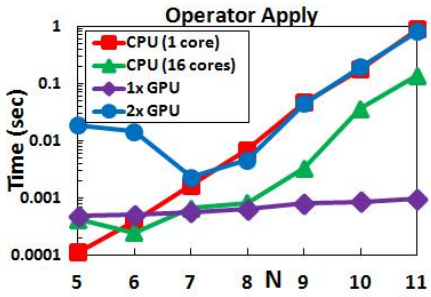


FIG. 4.1. *Timing results for operator apply*

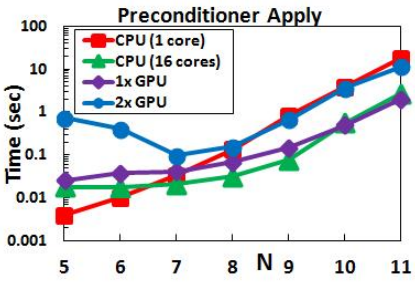


FIG. 4.2. *Timing results for preconditioner apply*

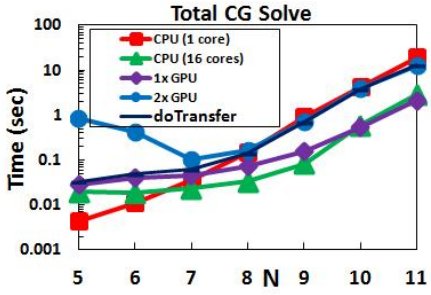


FIG. 4.3. *Timing results for solve*

THE $\mathscr{H}$-MATRIX-BASED FAST SOLVER FOR LARGE-SCALE CIRCUIT SIMULATION

BANGDA ZHOU*, SIVASANKARAN RAJAMANICKAM[1], AND HEIDI K. THORNQUIST[2]

Abstract. In this work, we proposed and developed an $\mathscr{H}$-matrix-based linear matrix solver for large-scale scientific simulation including large-scale complex integrated circuit simulation. In addition to geometrical approach to determine admissibility condition and build $\mathscr{H}$-matrix data structure which is widely applied in conventional $\mathscr{H}$-matrix algorithm, algebraic admissibility condition is developed in this work. By algebraic admissibility condition, the proposed solver can handle arbitrary sparse symmetric matrix with an $\mathscr{H}$-matrix representation.

1. Introduction. In recent years, there has been an increased demand for high performance and low power very large scale integration circuit design. High performance is achieved by technology scaling, increased functionality and competitive designs. On the other hand, a common technique used to obtain low-power design is to scale down the supply voltage. This stands to reason, since the chip power P is proportional to the square of the supply voltage V_{DD}. Thus, the demand for high performance and low power has led to modern VLSI designs being characterized by reduced feature size, increased functionality and lower supply voltage.

Increased chip functionality results in the need for complex and large-scale design. Lower supply voltage, on the other hand, makes the voltage variation across circuit terminals very critical since it may lead to chip failures. Consequently, efficient and accurate analysis of circuit is necessary for predicting the performance and improving the performance if necessary.

Among both DC and transient simulation of large circuit, numerical kernel always takes large portion of total simulation time. With the advancement of modern circuit design, large bandwidth and high speed signals require system level simulation which arise new challenges to numerical kernel. Therefore it is critical to improve the solution speed to facilitate the design of large-scale circuit.

The inverse M^{-1} of a sparse matrix $M \in \mathbb{R}^{n\times n}$ is fully populated in general. The computation of the exact inverse, which requires $O(n^3)$ operations, is therefore avoided in large-scale computations . Hierarchical matrix ($\mathscr{H}$-matrix) approximate matrix in a data-sparse way, and the approximate arithmetic for $\mathscr{H}$-matrix is almost optimal. Data sparse representation of matrix, LU factors and inverse is proved for PDE problem due to kernel degeneration of integral operator [3,4,15,20]. For PDE-like problem, like large-scale powergrid analysis, compared by conventional sparse matrix solvers [1, 5, 6, 14], $\mathscr{H}$-matrix for solving sparse matrix generated from circuit analysis is also shown to be efficient [13, 19]. In conventional $\mathscr{H}$-matrix algorithm, admissibility condition [3,11] is based on geometric information, which is not always achievable in common sparse matrix computation. Therefore algebraic admissibility condition [2,17,19] is introduced to determine the $\mathscr{H}$-matrix representation based on graph generated from sparse matrix itself [9, 10, 12, 17, 18].

In this paper, we developed an $\mathscr{H}$-matrix-based sparse matrix solver for large-scale simulation using algebraic admissibility condition and its superior performance and controllable accuracy is demonstrated by simulating two examples , of which the maximum number of unknown is beyond 12 million and total simulation time is within 1 hour.

2. $\mathscr{H}$-Matrix-based Fast Solver. Hierarchical matrices provide approximations to the matrix with almost linear complexity in both operation counts and memory consumption. The inverse and LU factorization complexity of a sparse matrix generated from finite element method (FEM) are proved to be $O(N\log N^2)$ [15] and solution and memory consumption complexity are $O(N\log N)$, compared with $O(N^2)$ in exact arithmetic.

*Purdue University, zhou136@purdue.edu
[1]Sandia National Laboratories, srajama@sandia.gov
[2]Sandia National Laboratories, hkthorn@sandia.gov

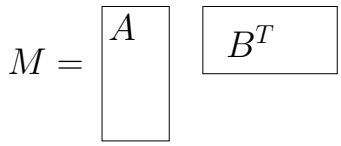


FIG. 2.1. *Rank-k approximation of M*

2.1. Low Rank Approximation. Low rank approximation of a matrix M not only reduces the operation count but also save the storage cost.

THEOREM 1 (Best Approximation with Fixed Rank [3]). *Let*

$$M = U\Sigma V^T \tag{2.1}$$

be a SVD of $M \in \mathbb{R}^{n\times m}$. *Let*

$$\tilde{M} := \tilde{U}\tilde{\Sigma}\tilde{V}^T \tag{2.2}$$

with matrices

$$\tilde{U} := U|_{n\times k}, \tilde{\Sigma} := diag(\Sigma_{1,1}, \Sigma_{2,2}, \ldots, \Sigma_{k,k}), \tilde{V}^T := V|_{m\times k}. \tag{2.3}$$

Then $\tilde{M}$ *is a best approximation to* M *in the sense that*

$$\|M-\tilde{M}\|_2 = \Sigma_{k+1,k+1}, \|M-\tilde{M}\|_F = \sqrt{\sum_{i=k+1}^{\min(n,m)} \Sigma_{i,i}^2}. \tag{2.4}$$

holds in the Frobenius and spectral norm.

Proof. For the spectral norm the proof is contained in [8]. The extension to the Frobenius norm can be achieved by induction. □

According to Theorem 1, by given any accuracy requirement ε, numerical rank of M can be adaptively determined.

To perform low-rank truncation on matrix with full rank is straightforward by Theorem 1. But in $\mathscr{H}$-matrix-based matrices multiplication and addition, low-rank truncation of matrices in rank-k approximated form are always encountered. In order to handle low-rank truncation efficiently, reduced Singular Value Decomposition (rSVD) is introduced.

Let $M = AB^T \in \mathbb{R}^{m\times n}$ be a matrix in rank-k approximation form as shown in Figure 2.1. We compute an rSVD $M = U\Sigma V^T$ by

1. Computing (reduced) QR-factorisations of A, B: $A = Q_A R_A$, $B = Q_B R_B$ with matrices $Q_A \in \mathbb{R}^{n\times k}$, $Q_B \in \mathbb{R}^{m\times k}$, $R_A, R_B \in \mathbb{R}^{k\times k}$.
2. Computing an rSVD of $R_A R_B^T = U'\Sigma V'^T$.
3. Computing $U := Q_A U', V := Q_B V'$.

The operation complexity is reduced from $O(nm^2)$ (if $n \geq m$) to $O((m+n)k^2)$ by using rSVD.

2.2. Admissibility Condition. The partition of the index set $I \times I$ is critical by $\mathscr{H}$-matrix. It is usually generated by recursive subdivision of computational domain $I \times I$. The recursion stops in the blocks $t \times s$ that satisfy a so-called admissibility condition or that are small enough, i.e. $\min\{|t|, |s| \leq n_{min}\}$, where n_{min} is the predefined constant, denoted as *leafsize*.

DEFINITION 1 (Admissibility Condition). *Consider clusters $t, s \in I$, following condition is defined as admissibility condition,*

$$\min\{diam(t), diam(s)\} \leq \eta dist(t,s) \tag{2.5}$$

where $diam(t)$ is the diameter of cluster t, $dist(t,s)$ is the distance between cluster t and cluster s and η is a constant

DEFINITION 2 (Admissible Block). *Consider matrix block $M_{t\times s}$, $M_{t\times s}$ is admissible if admissibility condition between row cluster t and column cluster s is satisfied.*

DEFINITION 3 (Inadmissible Block). *Consider matrix block $M_{t\times s}$, $M_{t\times s}$ is inadmissible if following condition is satisfied,*

$$\#(t) \leq n_{min} \quad and \quad \#(s) \leq n_{min}. \tag{2.6}$$

where $\#(\cdot)$ is the unknown size and n_{min} is leafsize *as mentioned previously.*

The complexity of arithmetic operations for a good cluster tree should make sure that blocks become admissible as soon as possible. In [3], geometry bisection is proved efficient to generate $\mathscr{H}$-matrix structure. However, geometric information is not always available especially for circuit simulation. In [2, 17, 19], algebraic admissibility condition is introduced for $\mathscr{H}$-matrix construction without geometric information.

2.2.1. Geometric Approach. The geometric admissibility condition depends on the Euclidean diameters of the supports of the involved basis functions and on their Euclidean distance. For each $i \in I$, we denote the support of the corresponding basis function ϕ_i by $\Omega_i := supp(\phi_i)$. A point x_i is chosen for each index $i \in I$. The construction starts with the full index set I, which is the root of the cluster tree by definition. Then, we apply a suitable technique to find a disjoint partition of the index set and use this partition to create son clusters. We apply the procedure recursively to the sons until the index sets are small enough, namely when the size of index set reaches *leafsize*.

2.2.2. Algebraic Approach. For most sparse matrices from circuit problems and many other scientific applications, there is no geometric information available or geometric information is not achievable. In order to compute diam$(\cdot)$ and dist$(\cdot,\cdot)$ in (3), a graph $G_M(E_M, V_M)$ is generated from sparse matrix $M \in \mathbb{R}^{n\times n}$. Cluster Tree T_I [3] is constructed based on nested dissection ordering [7] provided by graph partition packages in [12, 18]. In the implementation of proposed $\mathscr{H}$-matrix based solver, we choose Scotch [18] for more balanced cluster tree compared with smaller separator by Metis [12]. Balanced tree structure will ease the construction of block cluster tree $T_{I\times I}$, which determine the recursive $\mathscr{H}$-matrix block data structure.

By nested dissection ordering, computational domain is recursively divided into subdomains and one separator, namely $t_0^0 = \{t_0^1, t_1^1, t_2^1\}$ as shown in Fig 2.2(a). t_0^0 is the computational domain at upper level. t_0^1 and t_1^1 are subdomains which are completely separated by separator t_2^1, which means in sparse matrix M matrix block $M(t_0^1, t_1^1)$ and $M(t_1^1, t_0^1)$ are zero block and they will be preserved during LU factorization, so that we can save time and memory by avoiding operations on those zero blocks. Here $M(r,c)$ denotes a matrix block in matrix M in which row cluster is r and column cluster is c. The nested dissection will be recursively done on subdomains and for separator normal bisection will be applied. Domain subdivision for both domain and separator stop when size of domain or separator reaches certain predefined value, namely *leafsize*. If upper domain is chose to be the parent of subdomains and its separator, a tree structure can be built, which is the cluster tree T_I. Generally the size of separator is smaller than the size of subdomains, the tree structure will become imbalanced. So if (2.2.2) is satisfied, separator S_i^k is pushed one level down as shown in Fig 2.2(b).

$$\alpha\#(S_i^k) < \#(D_j^k) \tag{2.7}$$

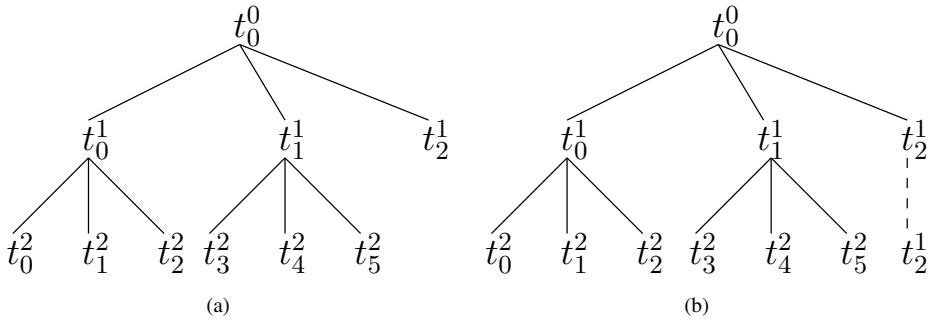


FIG. 2.2. *Cluster tree T_I separator relocation*

Here, $\#(S_i^k)$ denotes the size of the separator i at level k and D_j^k is the domain j at level k, α is a control constant, usually $\alpha \in [1,2]$. After pushing down separator when it is necessary, the cluster tree T_I is truncated at its minimum depth. We can have the following,

1. The size of each cluster at the same level is in comparable size.
2. The union of all the clusters at same level is equal to the whole computational domain I.

The remaining problem is to calculate the distance between two clusters and the diameter of a given cluster in (2.2.2). A graph $G_i(E_i,V_i)$ is constructed for each cluster $t_i \in T_I$. The diameter of a cluster t_i is estimated by calculating the radius of corresponding graph $G_i(E_i,V_i)$. Radius of one graph can be estimated by breadth first search. In implementation of proposed solver, breadth first search is done twice on one graph in order to achieve more accurate result. One is started from vertex with maximum degree and the other one is started from the root from the first search. The maximum depth represents the radius, namely the diameter of one cluster. For graph $G_i(E_i,V_i)$, time complexity is $O(|E_i|+|V_i|)$ in the worst case. So overall complexity of calculating diameter for all clusters in cluster tree is $O(N\log N)$, since the depth of cluster tree is $\log N$. To determine the distance of two clusters, one way is to calculate the shortest path between two set of vertices. However, this cannot be finished in polynomial time complexity. In order to keep the merit of $\mathscr{H}$-matrix-based solver, almost linear complexity, an approximated way [2] to determine the distance is implemented.

DEFINITION 4 (Neighbored Clusters). *Two cluster $t_1,t_2 \subset I$ are denoted as* neighbored clusters *if there exists an edge in $G_I(E_I,V_I)$ connecting indices of t_1 and t_2, i.e. $\exists i \in t_1$, $\exists j \in t_2$ such that $(i,j) \in E_I$.*

A cluster t is called *contiguous cluster* if there are two indices $i_{\min}$ and $i_{\max}$ such that $t = \{i : i_{\min} \leq i < i_{\max}\}$. Note that checking whether two contiguous cluster t_1 and t_2 are neighbored can be done with $O(\min(|t_1|,|t_2|))$ operations. It is shown in the following, if we assume $\#(t_1) \leq \#(t_2)$

1. Find the neighbor vertices of t_1, denoted as N_{t_1}
2. If $\exists k \in N_{t_1}$ such that $i_{\min}^{t_2} \leq k < i_{\max}^{t_2}$, t_1 and t_2 are neighbored clusters.

All clusters in cluster tree T_I are contiguous clusters if the index of vertices in each cluster is ordered. To achieve this, we just need to change the index of the graph $V_I \in G_I$ according to the ordering results from nested dissection, which can be done in $O(|E_I|+|V_I|)$, namely $O(N\log N)$ logarithmic linear complexity. And if t_1 and t_2 are neighbored clusters, the distance in between is 0.

If t_1 and t_2 are not neighbored, approximated distance is estimated by the length of shortest path in a newly constructed graph G_D using Dijkstra's algorithm, which can be done with $O(|V_{G_D}|^2)$ operations. The edges in G_D are based on the neighbored clusters from the same level. Each edge has a weight w that is

calculated by the following,

$$w = |(l_{t_1} - l'_{t_1}) - (l_{t_2} - l'_{t_2})| + 1 \tag{2.8}$$

where l denotes the current level and l' is the virtual level, the level where cluster t locates before pushing down. Only for separator clusters l and l' are different. And distance is calculated between domain cluster and separator cluster, so either $l_{t_1} - l'_{t_1}$ or $l_{t_2} - l'_{t_2}$ is 0. It is proved that vertices V_{G_D} in the graph G_D is bounded [2]. Hence, the approximate distance can be computed with a bounded number of operations.

The previous approach can be refined by the following iterative procedure. Dijkstra's algorithm not only computes the distance between t_1 and t_2, but also the nodes of the shortest path. We construct a new graph consisting of vertices from level $l+m$ for some m rooted at the shortest path nodes. Its edge are defined as in the previous graph G_D. The computation of the shortest path between t_1 and t_2 in this graph will lead to an improved approximation of $\mathrm{diam}(t_1, t_2)$.

3. Numerical Experiments. To demonstrate the efficiency and accuracy of proposed solver, two set of examples are simulated. One is from finite element method and the other one is from circuit analysis, namely modified nodal analysis (MNA).

3.1. Wathen Examples. Wathen examples are two dimensional finite element problem generated from Matlab gallery function. There is no geometric information available in Matlab, so it would not be feasible for conventional $\mathscr{H}$-matrix solver. The total number of unknown ranges from $30,401$ to $12,008,001$. In proposed solver, *leafsize* is 16, η in admissibility condition is 5 and truncation error ε is 10^{-5}. Other state-of-the-art sparse direct solver are also tested, like KLU [6], UMFPACK [5], SuperLU [14] and MUMPS [1].

Simulation time of proposed solver and other sparse direct solvers is shown in Fig 3.1. Compared with other solvers, superior performance of proposed solver is demonstrated. All other sparse solver cannot solve the largest 12 million problem but proposed solver can factorize and solve it within 1 hour. And time complexity from Fig 3.1 is close to linear. And in Fig 3.2, solution error of proposed solver is shown. Error is measured in relative residue of solving one random right-hand-side. Error across different test cases is well controlled, which is close to 10^{-9}.

In Fig 3.3, different configurations of proposed solver are tested. By letting η be zero, there are no low-rank approximated block generated during LU factorization, all blocks are stored in exact format. At this time, proposed solver acts like an exact sparse matrix solver. By comparison of two different configurations, we can conclude that $\mathscr{H}$-matrix is very efficient and play a key role in proposed solver.

3.2. IBM Powergrid Examples. IBM power grid examples [16] of different number of unknowns are tested. The sparse matrices are generated from DC analysis and transient analysis using modified nodal analysis. As shown in Table 3.2, the performance of proposed solver is competitive to other sparse solvers. For examples with small number of unknowns, proposed solver is slower than KLU, which is specifically designed to solver circuit problem. But for examples with large number of unknowns, proposed solver is faster. And another observation is that due to very sparse property of circuit problem, there are very few blocks in the low-rank approximated form. Even for the largest example, proposed solver only takes close to 1 minute to finish the simulation. Due to very few low-rank approximated blocks, proposed solver acts as an exact direct sparse matrix solver, so the solution error is in the same order to other direct sparse solvers.

4. Conclusions and Future Works. In this work, we proposed and developed an $\mathscr{H}$-matrix-based linear sparse matrix solver with logarithmic linear complexity. Also, we adopt algebraic admissibility condition in our proposed solver to handle the problem without geometric information. By simulating two set of examples, superior performance and good accuracy is achieve by comparing with other state-of-the-art direct sparse solvers.

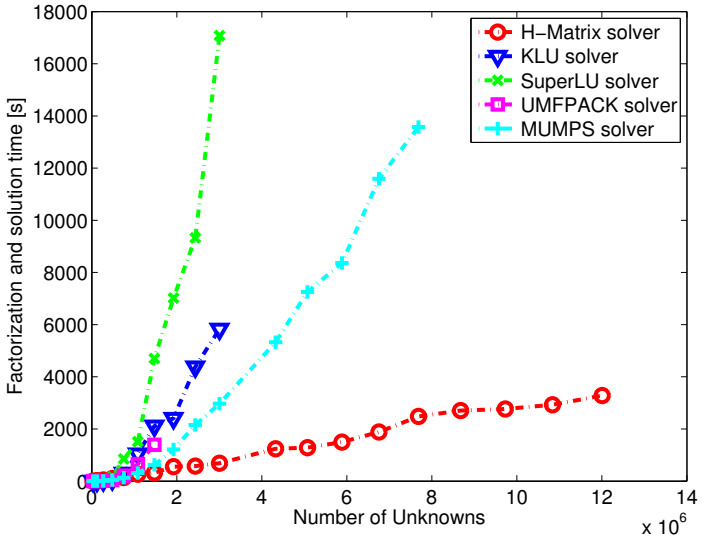


FIG. 3.1. *Simulation time*

Currently, the proposed solver is only capable of handling symmetric sparse matrix. In the future, the application of proposed solver could be further expanded by making it able to handle asymmetric matrix. And for circuit problem, the very sparse property of matrix itself should be considered in order to improve the ratio of the low-rank approximated blocks.

REFERENCES

[1] P. R. AMESTOY, I. S. DUFF, AND J.-Y. L'EXCELLENT, *Multifrontal parallel distributed symmetric and unsymmetric solvers*, Computer methods in applied mechanics and engineering, 184 (2000), pp. 501–520.

[2] M. BEBENDORF AND T. FISCHER, *On the purely algebraic data-sparse approximation of the inverse and the triangular factors*

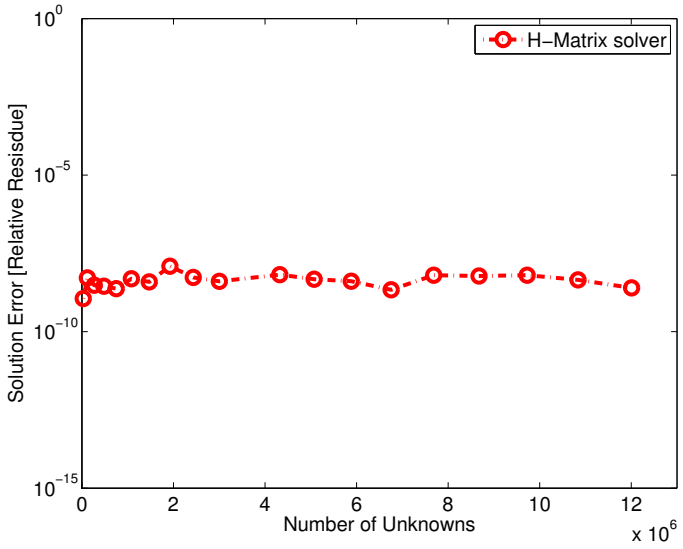


FIG. 3.2. *Solution Error of proposed solver*

of sparse matrices, Numerical Linear Algebra with Applications, 18 (2011), pp. 105–122.
[3] S. BÖRM, L. GRASEDYCK, AND W. HACKBUSCH, *Hierarchical matrices*, Lecture notes, 21 (2003).
[4] W. CHAI AND D. JIAO, *An $\mathscr{H}^2$ -matrix-based integral-equation solver of reduced complexity and controlled accuracy for solving electrodynamic problems*, Antennas and Propagation, IEEE Transactions on, 57 (2009), pp. 3147–3159.
[5] T. A. DAVIS, *Algorithm 832: Umfpack v4. 3—an unsymmetric-pattern multifrontal method*, ACM Transactions on Mathematical Software (TOMS), 30 (2004), pp. 196–199.
[6] T. A. DAVIS AND E. PALAMADAI NATARAJAN, *Algorithm 907: Klu, a direct sparse solver for circuit simulation problems*, ACM Transactions on Mathematical Software (TOMS), 37 (2010), p. 36.
[7] A. GEORGE, *Nested dissection of a regular finite element mesh*, SIAM Journal on Numerical Analysis, 10 (1973), pp. 345–363.
[8] G. H. GOLUB AND C. F. VAN LOAN, *Matrix computations*, vol. 3, JHU Press, 2012.
[9] L. GRASEDYCK, W. HACKBUSCH, AND R. KRIEMANN, *Performance of h-lu preconditioning for sparse matrices*, Comput. Methods Appl. Math., 8 (2008), pp. 336–349.

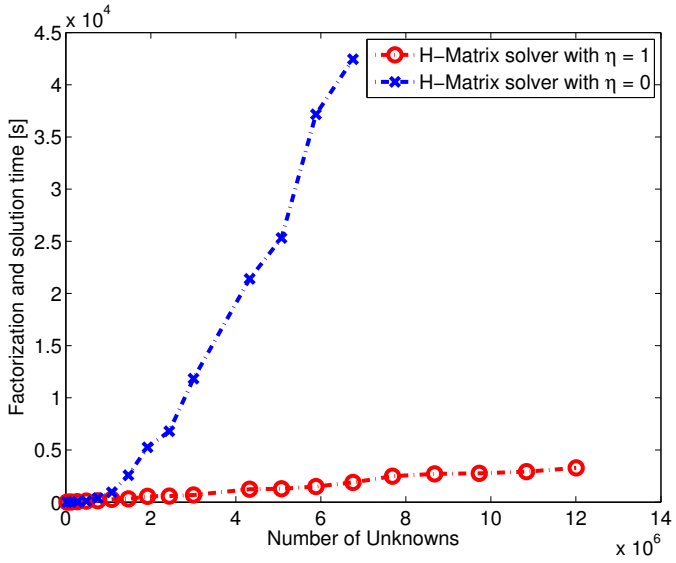


FIG. 3.3. *Comparison of different configurations*

[10] L. GRASEDYCK, R. KRIEMANN, AND S. LE BORNE, *Domain decomposition based $\mathscr{H}$-lu preconditioning*, Numerische Mathematik, 112 (2009), pp. 565–600.

[11] W. HACKBUSCH, B. N. KHOROMSKIJ, AND R. KRIEMANN, *Hierarchical matrices based on a weak admissibility criterion*, Computing, 73 (2004), pp. 207–243.

[12] G. KARYPIS, *Multi-constraint mesh partitioning for contact/impact computations*, in Proceedings of the 2003 ACM/IEEE conference on Supercomputing, ACM, 2003, p. 56.

[13] J. N. KOZHAYA, S. R. NASSIF, AND F. N. NAJM, *A multigrid-like technique for power grid analysis*, Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 21 (2002), pp. 1148–1160.

[14] X. S. LI, *An overview of superlu: Algorithms, implementation, and user interface*, ACM Transactions on Mathematical Software (TOMS), 31 (2005), pp. 302–325.

[15] H. LIU AND D. JIAO, *Existence of $\mathscr{H}$-matrix representations of the inverse finite-element matrix of electrodynamic problems and-based fast direct finite-element solvers*, IEEE Transactions on Microwave Theory and Techniques, 58 (2010), p. 3697.

TABLE 3.1
Simulation time of IBM Powergrid examples in second

	Unknowns	$\mathscr{H}$-Matrix	KLU	SuperLU	UMFPACK
IBMpg1sf	44,285	0.28	0.1	0.17	0.23
IBMpg1t_ls	53,643	0.41	0.11	0.19	0.25
IBMpg2sf	126,905	3.62	2.38	4	1.89
IBMpg2t_ls	164,237	4.4	2.5	3.82	2.11
IBMpg3sf	850,626	68.79	146.27	211.86	78.46
IBMpg4sf	952,618	93.43	222.59	267.55	84.96
IBMpg5sf	1,617,840	41.28	88.36	-	63.04
IBMpg5t_ls	2,091,594	54.15	75.18	-	106.35

[16] S. NASSIF, *Ibm power grid benchmarks*, 2008.
[17] S. OLIVEIRA AND F. YANG, *An algebraic approach for $\mathscr{H}$-matrix preconditioners*, Computing, 80 (2007), pp. 169–188.
[18] F. PELLEGRINI AND J. ROMAN, *Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs*, in High-Performance Computing and Networking, Springer, 1996, pp. 493–498.
[19] J. M. SILVA, J. R. PHILLIPS, AND L. M. SILVEIRA, *Efficient simulation of power grids*, Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 29 (2010), pp. 1523–1532.
[20] B. ZHOU, H. LIU, AND D. JIAO, *A direct finite element solver of linear complexity for large-scale 3-d circuit extraction in multiple dielectrics*, in Proceedings of the 50th Annual Design Automation Conference, ACM, 2013, p. 140.

KKT PRECONDITIONERS FOR NON-HERMITIAN INDEFINITE PDE SYSTEMS

STEPHEN D. SHANK*, PAUL TSUJI[1], RAY TUMINARO[2], AND DENIS RIDZAL[3]

Abstract. This paper is concerned with KKT preconditioning in the context of PDE-constrained optimization. An all-at-once approach for quadratic minimization problems with linear constraints as well as a more general trust region SQP method are considered with an emphasis on situations where the associated PDE problem has an operator which is indefinite. We analyze the effectiveness of an approximate Schur complement method to preconditioning KKT systems, extending existing results to non-Hermitian and indefinite systems. The performance of the proposed preconditioner is illustrated on problems from acoustic control.

1. Introduction and Motivation. This paper considers the iterative solution of linear systems associated with equality-constrained optimization

$$\begin{aligned} &\min_{z,u} \mathscr{F}(z,u) \\ \text{subject to} \quad &\mathscr{G}(z,u) = 0, \end{aligned} \tag{1.1}$$

where $\mathscr{F}$ is the objective function, $\mathscr{G}$ specifies constraints representing a PDE that characterizes the physics of the model, u represents a physical state such as displacement or velocity field, and z is a variable of interest related to the specific type of optimization. In parameter estimation problems, z might represent material parameters such as the bulk modulus, while in optimal control problems z might correspond to an input into the system. Problems of this type arise in a wide range of different contexts. In this paper, we focus on Helmholtz operators and contrast this with the better studied case where $\mathscr{G}(z,u)$ is associated with a Poisson operator. Our analysis is also useful for optimization problems with other non-Hermitian operators, such as those arising in convection-diffusion and nonlinear elasticity.

The main goal of this paper is to extend results for a general KKT preconditioning strategy proposed by Rees, Dollar, and Wathen [16] (both theoretically and computationally) from symmetric positive-definite PDE systems to non-Hermitian and indefinite systems. The KKT preconditioner in [16] is motivated by an all-at-once approach applied to a class of distributed control problems with linear constraints and a quadratic objective function. The underlying linear systems have block structure

$$\begin{pmatrix} 2\beta M & 0 & -M \\ 0 & M & \mathscr{G}_u^* \\ -M & \mathscr{G}_u & 0 \end{pmatrix} \tag{1.2}$$

where M is a finite element mass matrix, $\mathscr{G}_u$ is the matrix operator associated with the linear PDE constraints, and $*$ denotes the conjugate transpose. This KKT system arises from a specific quadratic minimization problem which includes a penalty parameter β; the minimization problem along with a preconditioner are discussed in [16] (and further described in Section 2). In addition to computational results, a convergence proof demonstrates the optimality of the preconditioner for a class of problems where the underlying discrete PDE operator is a symmetric positive-definite (SPD) matrix. Our key result centers on extending these results theoretically and computationally to systems which are not necessarily SPD. In particular, we illustrate that when the resulting preconditioner is applied to indefinite problems, the corresponding convergence behavior is relatively insensitive to changes in the mesh spacing.

*Temple University, Philadelphia, PA 19122
[1]Sandia National Laboratories, Livermore, CA 94550
[2]Sandia National Laboratories, Livermore, CA 94550
[3]Sandia National Laboratories, Albuquerque, NM 87123

We then apply this preconditioning strategy within a more general inexact sequential quadratic programming (SQP) trust region framework [7, 8, 17], applicable to nonlinear functions $\mathscr{F}$ and $\mathscr{G}$. A natural requirement in many applications is that the objective function and control may only involve quantities within a subregion of the physical domain. An important feature of the inexact trust-region SQP algorithm is that it remains applicable in this setting. The underlying linear subsystems that must be solved have structure

$$\begin{pmatrix} I^{(z)} & 0 & \mathscr{G}_z^* \\ 0 & I^{(u)} & \mathscr{G}_u^* \\ \mathscr{G}_z & \mathscr{G}_u & 0 \end{pmatrix} \tag{1.3}$$

where $\mathscr{G}_u$ and $\mathscr{G}_z$ are matrices associated with a linearization of $\mathscr{G}(z,u)$ corresponding to the Jacobian of the governing PDE, and $I^{(u)}$ and $I^{(z)}$ are identity matrix operators. It is important to notice that these subproblems do not depend on $\mathscr{F}$, which might include regularization or penalty parameters. Furthermore, the 2×2 upper-left block matrix is always well-conditioned regardless of the specific nature of the objective function or control (e.g., whether these are defined within a subregion or globally). This means that a general preconditioning strategy can be employed over a wide range of different optimization scenarios. This situation is in contrast to Equation (1.2) which includes the β term, and where the natural generalization of the M terms to locally defined (subregion) control and/or optimization may lead to rank deficient operators. Thus, the inexact SQP trust-region framework is widely applicable and generally leads to better conditioned KKT systems.

2. KKT Preconditioners for SPD systems. Rees, Dollar, and Wathen [16] consider the control problem

$$\begin{aligned} \min_{u,f}\ & \frac{1}{2}||u-\hat{u}||^2_{L_2(\Omega)} + \beta||f||^2_{L_2(\Omega)} \\ \text{subject to}\quad & -\Delta u = f \quad \text{in } \Omega \\ & u = 0 \quad \text{on } \partial\Omega. \end{aligned} \tag{2.1}$$

The discrete version of this problem is of the form

$$\begin{aligned} \min_{\mathbf{u},\mathbf{f}}\ & \frac{1}{2}\mathbf{u}^* M\mathbf{u} - \mathbf{u}^*\mathbf{b} + \beta\mathbf{f}^* M\mathbf{f} \\ \text{subject to}\quad & K\mathbf{u} = M\mathbf{f} \end{aligned} \tag{2.2}$$

where $K \in \mathbb{R}^{n\times n}$ is a finite element discretization matrix of Δ on Ω, $M \in \mathbb{R}^{n\times n}$ is the mass matrix, and $\mathbf{u}$, $\mathbf{b}$, and $\mathbf{f}$ are n-vectors of coefficients for u, $M\hat{u}$, and f. For the Lagrange multipliers $\lambda \in \mathbb{R}^n$, the discrete Lagrangian for this problem is

$$\mathscr{L}(\mathbf{u},\mathbf{f},\lambda) = \frac{1}{2}\mathbf{u}^* M\mathbf{u} - \mathbf{u}^*\mathbf{b} + \beta\mathbf{f}^* M\mathbf{f} + \lambda^*(K\mathbf{u} - M\mathbf{f})\ . \tag{2.3}$$

To satisfy the first-order optimality conditions, $\mathbf{u}$, $\mathbf{f}$, and λ must satisfy

$$\begin{pmatrix} 2\beta M & 0 & -M \\ 0 & M & K^* \\ -M & K & 0 \end{pmatrix} \begin{pmatrix} \mathbf{f} \\ \mathbf{u} \\ \lambda \end{pmatrix} = \begin{pmatrix} 0 \\ \mathbf{b} \\ 0 \end{pmatrix}. \tag{2.4}$$

We denote this 3×3 block matrix as $\mathscr{A}$. In [16], the authors propose the preconditioners

$$\mathscr{P}_1 = \begin{pmatrix} 2\beta M & 0 & 0 \\ 0 & M & 0 \\ 0 & 0 & \frac{1}{2\beta}M + KM^{-1}K^* \end{pmatrix} \quad \& \quad \mathscr{P}_2 = \begin{pmatrix} 2\beta M & 0 & 0 \\ 0 & M & 0 \\ 0 & 0 & KM^{-1}K^* \end{pmatrix}, \tag{2.5}$$

where $\mathscr{P}_1$ uses the exact Schur complement, $S = \frac{1}{2\beta}M + KM^{-1}K^*$ while $\mathscr{P}_2$ uses an approximation, $\tilde{S} \approx KM^{-1}K^*$. From a practical perspective, $\mathscr{P}_2$ is much more attractive, as one typically wants to replace inverses within the preconditioner by an easy-to-compute approximation. In the case of $\mathscr{P}_2$, one might apply a single multigrid V-cycle for the matrices K and K^* as a replacement for K^{-1} and K^{-*}. As Rees/Dollar/Wathen demonstrate, this replacement typically does not degrade the overall convergence rate associated with using the exact solves.

For $\mathscr{P}_1$, the preconditioned linear system given by $\mathscr{P}_1^{-1}\mathscr{A}$ has three distinct eigenvalues, which implies that GMRES converges in at most three iterations; this is given by [13], as the matrix in (2.4) can be written as a block 2×2 saddle-point system. Further, [16] shows that the condition number of the preconditioned linear system $\mathscr{P}_2^{-1}\mathscr{A}$ does not grow as the mesh is refined under some assumptions on the discrete PDE operator. These assumptions include that the discrete PDE operator K is SPD, and that certain bounds exist on the eigenvalues of M and K. We now formally summarize Proposition 3.2 in [16]. Specifically, let μ be an eigenvalue of $\mathscr{P}_2^{-1}\mathscr{A}$, then

$$\begin{aligned} &\text{either} \quad \mu = 1, \\ &\quad \text{or} \quad \frac{1}{2}(1+\sqrt{1+4\tilde{\mu}_1}) \le \mu \le \frac{1}{2}(1+\sqrt{1+4\tilde{\mu}_n}), \\ &\quad \text{or} \quad \frac{1}{2}(1-\sqrt{1+4\tilde{\mu}_1}) \le \mu \le \frac{1}{2}(1-\sqrt{1+4\tilde{\mu}_n}) \end{aligned} \tag{2.6}$$

where $0 \le \tilde{\mu}_1 \le \ldots \le \tilde{\mu}_n$ are the eigenvalues of the preconditioned Schur complement, $\frac{1}{2\beta}(KM^{-1}K^*)^{-1}M + I$. Furthermore, if the stiffness and mass matrices satisfy

$$c_1 h^2 \le \frac{\mathbf{x}^* M \mathbf{x}}{\mathbf{x}^* \mathbf{x}} \le c_2 h^2, \quad d_1 h^2 \le \frac{\mathbf{x}^* K \mathbf{x}}{\mathbf{x}^* \mathbf{x}} \le d_2 \tag{2.7}$$

with constants c_1, c_2, d_1, and d_2 independent of h and β, then

$$\begin{aligned} &\text{either} \quad \mu = 1, \\ &\quad \text{or} \quad \frac{1}{2}\left(1+\sqrt{5+\frac{2a_1h^4}{\beta}}\right) \le \mu \le \frac{1}{2}\left(1+\sqrt{5+\frac{2a_2}{\beta}}\right), \\ &\quad \text{or} \quad \frac{1}{2}\left(1-\sqrt{5+\frac{2a_2}{\beta}}\right) \le \mu \le \frac{1}{2}\left(1-\sqrt{5+\frac{2a_1h^4}{\beta}}\right). \end{aligned}$$

with constants a_1 and a_2 independent of h and β. Thus, the eigenvalues of the preconditioned system $\mathscr{P}_2^{-1}\mathscr{A}$ are clustered in a few distinct intervals.

The eigenvalue bound assumptions given by (2.7) typically hold for standard linear finite elements and second-order finite differences when the underlying mesh is shape regular, and when M corresponds to a mass matrix defined over the entire domain. This result certainly holds true for the Poisson problem, if the discretization is a standard finite element discretization with linear basis functions on a quasi-uniform mesh. There are two limitations with this particular eigenvalue clustering result. First, the control problem given by (2.2) assumes that one is interested in evaluating the state misfit and the control penalty terms over the entire domain. This leads to the appearance of the mass matrices on the diagonal of the 2×2 upper left block of (2.4). In many situations, it is natural to only minimize the data fit or apply control within a subregion. In this case, M must be replaced with rank deficient versions where only subregions are populated with entries. Here, the preconditioning method is not well defined. Second, as one considers smaller β, the clusters

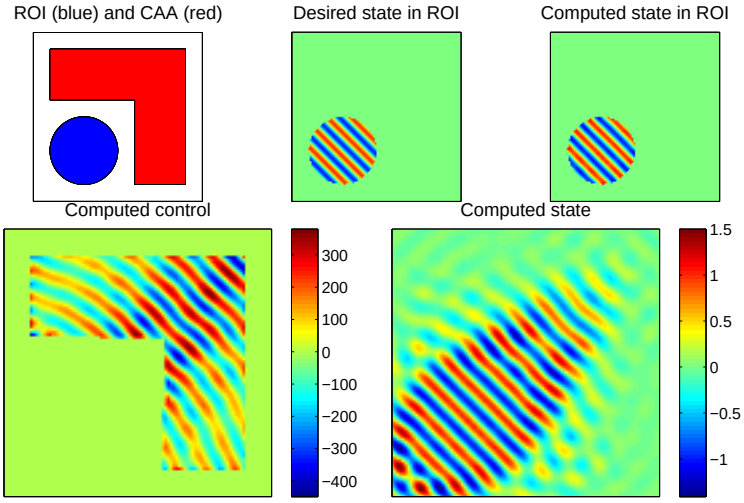


FIG. 3.1. *Localized control where one seeks to create a plane wave in a given direction in the region of interest (ROI), while only allowing a nonzero source in the control allowable area (CAA).*

become less tight and one observes that the convergence of the associated preconditioner tends to degrade. As small β is of interest, this is a bit problematic. It should, however, be noted that a recent proposed preconditioning improvement by Pearson/Wathen in [15] appears to be significantly less dependent on β, allowing for solver convergence which is almost independent of both mesh size and regularization. Finally, while the above preconditioning strategy appears limited to optimization problems of the form (2.2), one can instead consider a class of SQP methods [7, 8, 17], applicable to a much broader range of problems. As shown in Section 4, the same preconditioner strategy can then be used on the associated linear subsystems generated by these SQP methods.

3. KKT Preconditioners for non-SPD systems. We now consider preconditioners when the discrete PDE operator is non-Hermitian or indefinite (e.g., Helmholtz and advection-diffusion equations). For the Helmholtz equation, the underlying control problem of interest is

$$\begin{aligned} \min_{u,f} \frac{1}{2}||u-\hat{u}||^2_{L_2(\Omega_I)} + \beta||f||^2_{L_2(\Omega_C)} & \\ \text{subject to} \quad -\Delta u - \omega^2 u = f \quad & \text{in } \Omega,\ \text{supp}(f) \subset \Omega_C \\ \frac{\partial u}{\partial n} - i\omega u = 0 \quad & \text{on } \partial\Omega, \end{aligned} \tag{3.1}$$

where $\Omega_I \subseteq \Omega$ is a region of interest in which one seeks to match the computed state u to some desired state $\hat{u}$. $\Omega_C \subseteq \Omega$ is a control allowable area, i.e., the portion of the domain where the source term may be nonzero. We refer to the case $\Omega_C = \Omega$ as distributed control, while $\Omega_C \subsetneq \Omega$ is termed localized control. Such a localized control problem is illustrated in Figure 3.1.

In Equation (3.1), the last equation is a first-order approximation to the Sommerfeld radiation condition; this boundary condition is used to simulate an infinite domain where wave energy propagates outward from Ω. In practice, other boundary conditions such as perfectly matched layers can be used. The stiffness matrix K in the previous section is now replaced with the discretization of the Helmholtz operator taking the form

$A = K + i\omega C - \omega^2 M$; here, the damping matrix C only has nonzero entries on the boundary resulting from the $i\omega$ term of the radiation condition, and K is equivalent to the discretization of the Laplacian with Neumann boundary conditions. This matrix is complex symmetric and, even for small values of ω, indefinite.

In this section we first consider a distributed control problem. The case of localized control is deferred to Section 4. The corresponding KKT system is

$$\begin{pmatrix} 2\beta M & 0 & -M \\ 0 & M & A^* \\ -M & A & 0 \end{pmatrix} \begin{pmatrix} \mathbf{f} \\ \mathbf{u} \\ \lambda \end{pmatrix} = \begin{pmatrix} 0 \\ \mathbf{b} \\ 0 \end{pmatrix}. \tag{3.2}$$

To distinguish the different KKT systems, the 3×3 matrix with the Helmholtz operator will be denoted as $\mathscr{A}_H$. The proposed preconditioner is

$$\mathscr{P}_H = \begin{pmatrix} 2\beta M & 0 & 0 \\ 0 & M & 0 \\ 0 & 0 & AM^{-1}A^* \end{pmatrix}. \tag{3.3}$$

As we consider general systems, our analysis is based on singular values instead of eigenvalues and relies on the following assumption.

ASSUMPTION 3.1. *Solutions of the discrete Helmholtz equation satisfy the following stability estimate*

$$\langle u_h, u_h \rangle \;\leq\; \frac{c_3}{\omega} \langle f_h, f_h \rangle$$

where c_3 is a constant independent of mesh spacing or frequency, $\langle \cdot\,,\, \cdot \rangle$ denotes a standard L_2 inner product, and the functions u_h and f_h correspond to standard finite element representations. That is,

$$u_h(\mathbf{x}) = \sum_j \bar{u}_j \psi_j(\mathbf{x}) \qquad \textit{and} \qquad f_h(\mathbf{x}) = \sum_j \bar{f}_j \psi_j(\mathbf{x})$$

where $\psi_j(\mathbf{x})$ are nodal finite element basis functions, the $\bar{u}_j$'s (and $\bar{f}_j$'s) are scalars, and $\mathbf{x}$ denotes the vector position in $\mathbb{R}^2$. Helmholtz stability estimates of this form can generally be found [2,10–12]. Typically, these results include technical assumptions (e.g. regularity) and limitations on the shape of the domain and on the type of boundary conditions. In general, the Helmholtz case is much harder to address than elliptic Poisson-like problems. For example, one must exclude resonant cavities which give rise to singular systems. We do not focus on these any further and point the interested reader to the references.

LEMMA 3.2. *Assume that Assumption 3.1 holds with all the associated function and operator definitions. Then, the corresponding Helmholtz discrete matrices satisfy*

$$\|M^{1/2} A^{-1} M^{1/2}\|_2 \leq \frac{c_3}{\omega}.$$

Proof. Applying the definition of u_h and the inner product, direct substitution leads to

$$\langle u_h, u_h \rangle \;=\; \sqrt{\sum_i \sum_j \int_\Omega \bar{u}_i \bar{u}_j \psi_j(\mathbf{x}) \psi_j(\mathbf{x}) d\mathbf{x}} = \sqrt{\bar{u}^* M \bar{u}}$$

where $\bar{u}$ is the vector comprised of the $\bar{u}_i$'s. Likewise,

$$\langle f_h, f_h \rangle \;=\; \sqrt{\bar{f}^* M \bar{f}},$$

and so Assumption 3.1 implies that

$$\sqrt{\bar{u}^* M \bar{u}} \;\leq\; \frac{c_3}{\omega}\sqrt{\bar{f}^* M \bar{f}}$$

or equivalently

$$\sqrt{\bar{f}^* M A^{-*} M A^{-1} M \bar{f}} \;\leq\; \frac{c_3}{\omega}\sqrt{\bar{f}^* M \bar{f}} \tag{3.4}$$

using the fact that $\bar{u} = A^{-1} M \bar{f}$. Letting $v = M^{-\frac{1}{2}} \bar{f}$ and applying the definition of the vector 2-norm, (3.4) can be transformed to

$$\|M^{1/2} A^{-1} M^{1/2} v\|_2 \leq \frac{c_3}{\omega}\|v\|_2.$$

Recognizing that this inequality must hold for all v, the Lemma is completed simply by applying the definition of the matrix 2-norm. □

COROLLARY 3.3. *Assume that Assumption 3.1 holds. Then, the discretized Helmholtz operator A satisfies*

$$c_4 h^2 \leq \Sigma(M^{1/2} A^{-1} M^{1/2}) \leq \frac{c_3}{\omega}$$

where $\Sigma(M^{1/2} A^{-1} M^{1/2})$ *are the singular values of* $M^{1/2} A^{-1} M^{1/2}$*, and* c_3*,* c_4 *are positive constants independent of h and* β*.*

Proof. The upper bound is a direct consequence of Lemma 3.2. The lower bound is proved by first recognizing that one can instead develop an upper bound for the largest singular value of the inverse of $M^{1/2} A^{-1} M^{1/2}$ (under an assumption that A and M are invertible) and then use matrix norm inequalities and properties of the mass/stiffness matrices for standard nodal finite elements. In particular,

$$\begin{aligned}\sigma_{min}(M^{\frac{1}{2}} A^{-1} M^{\frac{1}{2}}) &= \sigma_{max}(M^{-\frac{1}{2}} A M^{-\frac{1}{2}})^{-1} \\ &= \|M^{-\frac{1}{2}} A M^{-\frac{1}{2}}\|_2^{-1}.\end{aligned}$$

Decomposing the matrix A into its individual components, we have

$$\begin{aligned}\|M^{-\frac{1}{2}} A M^{-\frac{1}{2}}\|_2^{-1} &= \|M^{-\frac{1}{2}}(K + i\omega C - \omega^2 M) M^{-\frac{1}{2}}\|_2^{-1} \\ &\geq \frac{1}{\|M^{-\frac{1}{2}}\|_2(\|K\|_2 + \omega\|C\|_2 + \omega^2\|M\|_2)\|M^{-\frac{1}{2}}\|_2} \\ &= \frac{1}{\|M^{-1}\|_2(\|K\|_2 + \omega\|C\|_2 + \omega^2\|M\|_2)} \\ &\geq c_1 h^2 \frac{1}{\|K\|_2 + \omega\|C\|_2 + \omega^2\|M\|_2}\end{aligned}$$

where the last line is obtained via Equation (2.7). Since the damping matrix C is comprised only of mass matrix terms on the boundary, there exist constants b_1 and b_2 such that $b_1 h \leq \|C\|_2 \leq b_2 h$. Additionally, we can use the previously established upper bounds given in Equation (2.7) on the norms of K and M to obtain

$$\|M^{-\frac{1}{2}} A M^{-\frac{1}{2}}\|_2^{-1} \geq c_1 h^2 \frac{1}{d_2 + \omega h b_2 + \omega^2 h^2 c_2}\ .$$

Finally, assuming that there is a constant number of discretization points per wavelength N_{ppw} and using the relation $\omega = \frac{2\pi}{N_{ppw}h}$, we have

$$\|M^{-\frac{1}{2}}AM^{-\frac{1}{2}}\|_2^{-1} \geq h^2 \frac{c_1}{d_2 + \frac{2\pi}{N_{ppw}}b_2 + \frac{4\pi^2}{N_{ppw}^2}c_2} = c_4 h^2$$

where c_4 is a constant independent of frequency. This requires that d_2 be chosen sufficiently large so that it dominates other terms in the denominator for realistic values of N_{ppw} (needed for the discretization stability and accuracy). □ In Section 6, we give further numerical evidence for this result. With the above corollary bounding the singular values, we now prove our main result about the preconditioned linear system.

THEOREM 3.4. *Consider μ, an eigenvalue of the preconditioned system $P_H^{-1}\mathcal{A}_H$. Given that Assumption 3.1 holds, there are positive constants a_1, a_2 independent of h and β such that*

$$\begin{aligned}
&\textit{either } \mu = 1, \\
&\quad \textit{or } \frac{1}{2}\left(1 + \sqrt{5 + \frac{2a_1h^4}{\beta}}\right) \leq \mu \leq \frac{1}{2}\left(1 + \sqrt{5 + \frac{2a_2}{\beta}}\right), \\
&\quad \textit{or } \frac{1}{2}\left(1 - \sqrt{5 + \frac{2a_2}{\beta}}\right) \leq \mu \leq \frac{1}{2}\left(1 - \sqrt{5 + \frac{2a_1h^4}{\beta}}\right).
\end{aligned}$$

Proof. Consider the singular value decomposition

$$M^{1/2}A^{-1}M^{1/2} = U\Sigma V^*.$$

If we right-multiply with the transpose, we obtain

$$M^{1/2}A^{-1}MA^{-*}M^{1/2} = U\Sigma^2U^*,$$

which implies

$$M^{1/2}(A^*M^{-1}A)^{-1}M^{1/2} = U\Sigma^2U^*.$$

So the eigenvalues of $M^{1/2}(A^*M^{-1}A)^{-1}M^{1/2}$ are equivalent to the singular values of $M^{1/2}A^{-1}M^{1/2}$ squared, i.e. $\Lambda(M^{1/2}(A^*M^{-1}A)^{-1}M^{1/2}) = \Sigma(M^{1/2}A^{-1}M^{1/2})^2$. Corollary 3.3 implies that

$$c_4^2h^4 \leq \Lambda(M^{1/2}(A^*M^{-1}A)^{-1}M^{1/2}) \leq \frac{c_3}{\omega},$$

Making a similarity transformation and shifting, we get

$$1 + c_4^2h^4 \leq \Lambda(I + (A^*M^{-1}A)^{-1}M) \leq 1 + \frac{c_3}{\omega}. \tag{3.5}$$

The proof is completed by plugging these bounds into Equation (2.6). □

The key is that both the exact Schur complement, $\frac{1}{2\beta}M + AM^{-1}A^*$, and the approximate Schur complement, $AM^{-1}A^*$, are both positive-definite even though A is indefinite. Thus, the preconditioner is well-behaved even if A is not positive-definite. In essence, what matters is that the singular values of $A^{-1}M$ are bounded independent of the mesh resolution.

4. Inexact SQP Methods. Sequential quadratic programming methods have been popularized in [1, 14]. Strategies for solving nonlinear equality-constrained optimization problems with inexact trust-region SQP methods are proposed in [7–9, 17, 21, 22]. These works discuss several aspects: basic strategy, robustness, theoretical properties, inexact iterative methods for linear subproblems, parameter choices, solver tolerances, and computational efficiency. The purpose of the current paper is to focus on the iterative linear solution strategy and preconditioning, which ultimately determines the efficiency of the overall strategy. Before doing this, however, we briefly mention a few features of the inexact SQP method that are relevant to our context.

Roughly, the derivation of an SQP algorithm for the equality-constrained optimization problem (1.1) comes from considering Newton's method applied to the optimality conditions associated with minimizing a Lagrangian functional

$$\mathscr{L}(z,u,\lambda) = \mathscr{F}(z,u) + \langle \lambda, \mathscr{G}(z,u) \rangle$$

where $\langle \cdot\,,\,\cdot \rangle$ denotes a suitable inner product. Each Newton step (applied to the optimality conditions) requires the solution of a KKT system which in turn is equivalent to resolving a quadratic minimization problem. This naturally leads to an approach based on solving a sequence of nonconvex quadratic trust-region subproblems. A composite step scheme can then be considered in which one step focuses on reducing linear infeasibility in what is termed a quasi-normal step. A second step, the tangential step, improves optimality while inexactly staying in the null space of the linearized constraints. This tangential step in turn involves solving a related optimization subproblem which can addressed by an inexact projected Steihaug-Toint CG (ST-CG) algorithm. The details of the algorithm specifics are not important for this paper. The key points to keep in mind are that the ST-CG algorithm handles the nonconvex trust-region subproblems and is primarily responsible for minimization of the objective function. The ST-CG algorithm only utilizes an iterative linear solver to perform null-space projections associated with the PDE constraints and these linear systems have the form (1.3). Further, the method is well-defined with a corresponding convergence theory when these null-space projection linear systems are solved inexactly (assuming appropriate stopping conditions are used).

To summarize, PDE preconditioning techniques are only needed for (1.3). Optimization parameters (e.g., a β regularization term) do not appear in these linear subproblems and the method is completely defined even if the minimization or control is only considered locally within subregions. All of this means that the KKT preconditioning strategy of Rees, Dollar, and Wathen can be applied to a broad class of nonlinear problems via this inexact SQP methodology. Of course, generality comes with trade-offs. Now, the objective function is addressed by the ST-CG algorithm which requires several inexact iterative linear solves (along with a couple of additional (1.3) system solves associated with pre- and post-processing) as opposed to a single linear system of the form (3.2) when considering the more restrictive optimization problem (3.1). Thus, the choice of optimization approach depends on whether one is interested in the more restrictive case or not. Finally, it is worth pointing out that inexact SQP methodology is attractive from a software perspective as a wide range of optimization problems can be addressed with the same software utilizing an iterative linear algebra package for systems which only involve PDE equations.

5. Solution of the Forward Problem. We use a shifted Laplacian preconditioner [4, 5] for the forward Helmholtz solves within $\mathscr{P}_H^{-1}$. This consists of applying a solver to a damped system $P_V = K - (1 + .5i)\omega^2 M$, where the complex frequency perturbation introduces damping that smooths out solutions. Multigrid methods, which often have issues on undamped Helmholtz problems (see for instance [3, 6]) are much more effective on these damped problems. Solves were carried out using a slightly modified smoothed aggregation AMG, where the prolongation and restriction operators are constructed using a discretization of the Laplacian, and the damping parameter varies between multigrid levels [19].

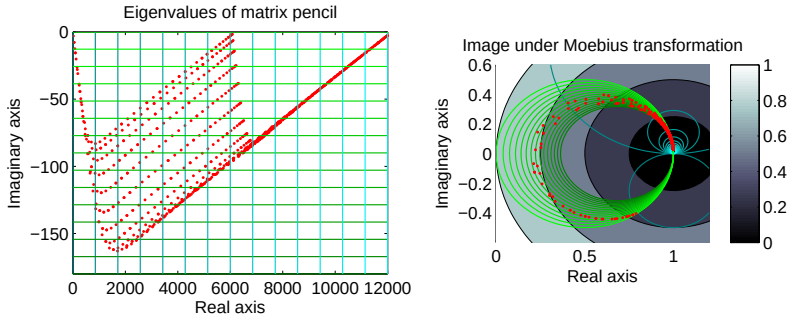


FIG. 5.1. *Eigenvalues of the matrix pencil $\Lambda(K,M)$ for a finite difference discretization of the Helmholtz equation, plotted in red (left). Green (blue) lines with constant imaginary (real) part are drawn to cover this pencil, changing from light to dark as the imaginary (real) part grows. These eigenvalues and lines are mapped by the Moebius transformation (right). The background contour plot represents the distance from 1.*

An analysis of shifted Laplacians is given in [20] illustrating how eigenvalues of the preconditioned system depend on eigenvalues of the matrix pencil $\Lambda(K,M)$. More precisely, $\Lambda(P_V^{-1}A) = m(\Lambda(K,M))$ where $m(z) = \frac{z-\omega^2}{z-(1+.5i)\omega^2}$. Results are provided describing circles in the complex plane containing the eigenvalues of the preconditioned system. Loosely speaking, this is explained by observing that A is near-Hermitian, so its eigenvalues may be near a line, and $m(z)$ is a Moebius transformation that maps lines to circles. When eigenvalues lie exactly on a circle as in Figure 5.1, approximation theory results indicate that the convergence of Krylov subspace solvers may degrade to that of classical methods. Such details are outside the scope of this paper, but we refer readers to [18] and provide some numerical evidence in Section 6.

6. Numerical Results. We begin by clarifying some remarks in the previous section concerning shifted Laplacian preconditioning and convergence degradation in Krylov subspace methods. Figure 6.1 contains a Helmholtz solution for $\omega = 40\pi$ as well as convergence curves comparing full GMRES, GMRES(5), GMRES(25), and preconditioned steepest descent, which is defined as the iteration $x_{k+1} = x_k + \alpha_k P_V^{-1} r_k$; here the scalar α_k is chosen to minimize the norm of the current residual. We observe that steepest descent

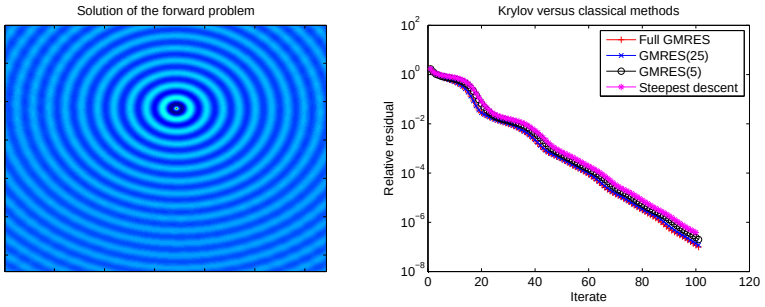


FIG. 6.1. *Residual convergence plots of GMRES & steepest descent on the forward problem.*

attains nearly the same *optimal* convergence as GMRES, at a fraction of the overall storage cost and work per iteration; thus, we believe it is a viable option within the overall KKT preconditioner as a replacement for the forward solves.

We now consider Assumption 3.1 which leads to the upper bound in Corollary 3.3. This upper bound is equivalent to

$$\|M^{1/2}A^{-1}M^{1/2}\|_2 \leq \frac{c_3}{\omega}$$

for c_3 independent of the mesh width h and frequency ω. Table 6.1 displays values of $\|A^{-1}M\|_2$ which is a related (more easily computed) quantity for varying ω and mesh width. It is clear that for a fixed frequency

ω / n	10	20	40	80	160	320
2π	5.64E-02	5.79E-02	5.82E-02	5.82E-02	5.82E-02	5.81E-02
4π		2.83E-02	2.88E-02	2.90E-02	2.91E-02	2.91E-02
8π			1.53E-02	1.56E-02	1.57E-02	1.57E-02
16π				7.94E-03	8.02E-03	8.05E-03
32π					4.08E-03	4.22E-03
64π						2.06E-03

TABLE 6.1

Values of $\|A^{-1}M\|_2$ for varying ω and n, the number of subintervals in a given direction.

$\|A^{-1}M\|_2$ approaches an asymptotic value (and so is bounded) as the mesh is refined. Further, the actual asymptotic value appears to be inversely proportional to the frequency (consistent with the Corollary).

Now, convergence curves corresponding to residual reduction are provided for solving the distributed control KKT system (3.2) via GMRES with preconditioner (3.3) to a tolerance of $\tau_{\text{out}} = 10^{-5}$. Figure 6.2 contains GMRES convergence curves for varying ω and mesh widths. Inner forward solves corresponding to A^{-1} and A^{-*} required to apply $\mathscr{P}_2^{-1}$ are accomplished via MATLAB's backslash. These results clearly

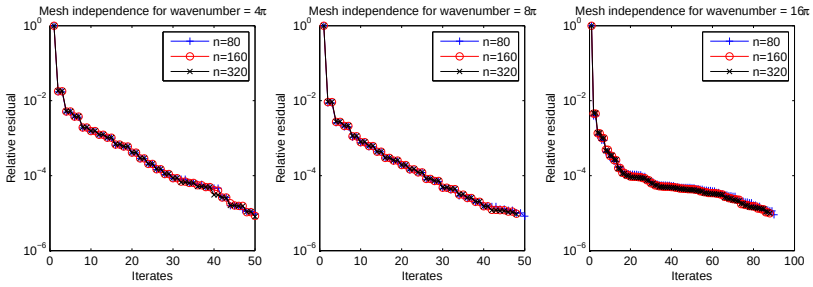


FIG. 6.2. *Convergence curves for KKT systems. In each plot the ω is fixed while mesh width varies; here n denotes the number of subintervals in a given direction.*

demonstrate mesh insensitivity and only a modest sensitivity to frequency. Interestingly, the number of iterations required for a residual reduction of 10^{-4} drops as the frequency increases while the trend is reversed if one considers a residual reduction of 10^{-5}. Clearly, this needs further investigation.

We also study sensitivity to inexactly solving the forward problem; In particular, A^{-1} and A^{-*} are replaced by shifted Laplacian preconditioners. For the fully inexact case, the forward solution is approximated by 5 steps of preconditioned steepest descent, with the preconditioner being one V-cycle of algebraic multigrid on a damped Helmholtz problem. α_k is computed initially and then reused for subsequent computations; what results is a stationary iteration that does not require use of a flexible Krylov method. Again, KKT systems are solved to $\tau_{\text{out}} = 10^{-5}$. Though there is some sensitivity to the inner solver, the AMG inner solver

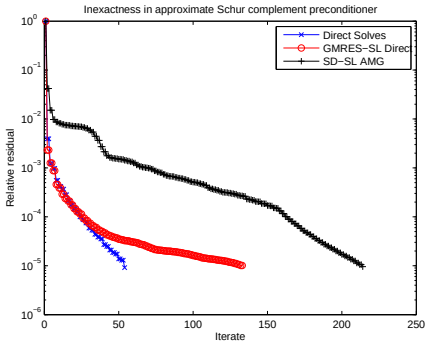


FIG. 6.3. *Inexact forward solves ($\omega = 16\pi$).* `Direct Solves` *uses A^{-1} & A^{-*}.* `GMRES-SL Direct` *uses GMRES preconditioned by shifted-Laplacian (direct solves for the shifted problem) to a 10^{-1} tolerance.* `SD-SL AMG` *uses 5 steepest descent steps (AMG preconditioning on shifted problem).*

is by far the least expensive. These results are illustrated in figure 6.3.

Finally, Table 6.2 contains data relative to the full inexact-SQP algorithm in the case of control and minimization being restricted to subregions of the domain. We again experiment with different linear solvers and varying mesh widths. Forward solves are now performed using preconditioned BiCGStab to a tolerance $\tau = 10^{-5}$. We note that forward problem solves for SL-Direct and SL-AMG typically took less than 20 and 35 iterations, respectively. It is observed that the number of forward solves, ST-CG iterations within the SQP

n	Solver	Forward solves	ST-CG iterations	SQP iterations
100	Direct	2065	278	6
200	Direct	1923	263	6
100	SL-Direct	4011	275	6
100	SL-AMG	2577	164	4

TABLE 6.2

SQP algorithm convergence for $\omega = 5.0265$; here n is the number of subintervals in each direction, solver corresponds to the forward problem, and SL stands for shifted Laplacian.

approach, and outer SQP iterations do not vary largely as different options are exercised.

7. Conclusions. We have extended a KKT preconditioning strategy originally developed for PDE constraints associated with symmetric positive definite operators to the indefinite Helmholtz case. We have shown both theoretically and practically that this strategy is insensitive to refinement in the mesh resolution and modestly sensitive to frequency changes. When the strategy is incorporated within an SQP framework, it can be used to solve a broad class of nonlinear equality constrained optimization problems. One key requirement for the KKT preconditioner is a *forward PDE solver*. Some preliminary results using a shifted Laplacian strategy combined with AMG indicate that these inner forward solves can be performed inexactly without a large degradation in the performance of the outer iterative procedure. Additional work is needed to better quantify this relationship between inner iteration accuracy to the overall robustness/efficiency of the outer iterations.

REFERENCES

[1] P. BOGGS AND J. TOLLE, *Sequential quadratic programming*, Acta Numerica, 4 (1995), pp. 1–51.

[2] J. Douglas, J. Santos, D. Sheen, and L. Bennethum, *Frequency domain treatment of one-dimensional scalar waves*, Mathematical Models and Methods in Applied Sciences, 3 (1993), pp. 171–194.
[3] Y. A. Erlangga, *Advances in iterative methods and preconditioners for the Helmholtz equation*, Archives of Computational Methods in Engineering, 15 (2008), pp. 37–66.
[4] Y. A. Erlangga, C. W. Oosterlee, and C. Vuik, *A novel multigrid based preconditioner for heterogeneous Helmholtz problems*, SIAM Journal on Scientific Computing, 27 (2006), pp. 1471–1492.
[5] Y. A. Erlangga, C. Vuik, and C. W. Oosterlee, *On a class of preconditioners for solving the Helmholtz equation*, Applied Numerical Mathematics, 50 (2004), pp. 409–425.
[6] O. G. Ernst and M. J. Gander, *Why it is difficult to solve helmholtz problems with classical iterative methods*, in Numerical Analysis of Multiscale Problems, Springer, 2012, pp. 325–363.
[7] M. Heinkenschloss and D. Ridzal, *An inexact trust-region SQP method with applications to PDE-constrained optimization*, in Numerical Mathematics and Advanced Applications, Springer-Verlag, 2008, pp. 613–620.
[8] ———, *A matrix–free trust–region SQP method for equality constrained optimization*, tech. rep., CAAM Technical Report TR11-17, Department of Computational and Applied Mathematics, Rice University, 2011. In review.
[9] M. Heinkenschloss and L. Vicente, *Analysis of inexact trust-region SQP algorithms*, SIAM Journal on Optimization, 12 (2002), pp. 283–302.
[10] U. Hetmaniuk, *Stability estimates for a class of helmholtz problems*, Communications in Mathematical Sciences, 5 (2007), pp. 665–678.
[11] F. Ihlenburg, *Finite element analysis of acoustic scattering*, Springer-Verlag, 1998.
[12] F. Ihlenburg and I. Babuska, *Finite element solution of the Helmholtz equation with high wave number, part I: the h-version of the FEM*, Computers and Mathematics with Applications, 30 (1995), pp. 9–37.
[13] M. Murphy, G. Golub, and A. Wathen, *A note on preconditioning for indefinite linear systems*, SIAM Journal on Scientific Computing, 21 (2000), pp. 1969–1972.
[14] J. Nocedal and S. Wright, *Numerical Optimization*, Springer, 2006.
[15] J. Pearson and A. Wathen, *A new approximation of the Schur complement in preconditioners for PDE-constrained optimization*, Numerical Linear Algebra with Applications, 19 (2012), pp. 816–829.
[16] T. Reese, H. Dollar, and A. Wathen, *Optimal preconditioners for PDE-constrained optimization*, SIAM Journal on Scientific Computing, 32 (2010), pp. 271–298.
[17] D. Ridzal, M. Aguilo, and M. Heinkenschloss, *Numerical study of a matrix–free trust–region SQP method for equality constrained optimization*, tech. rep., SAND2011-9346, 2011.
[18] Y. Saad, *Iterative methods for sparse linear systems*, Siam, 2003.
[19] P. Tsuji and R. Tuminaro, *Augmented AMG-Shifted Laplacian Preconditioners for Indefinite Helmholtz Problems*, submitted to Numerical Linear Algebra with Applications.
[20] M. B. van Gijzen, Y. A. Erlangga, and C. Vuik, *Spectral analysis of the discrete Helmholtz operator preconditioned with a shifted Laplacian*, SIAM Journal on Scientific Computing, 29 (2007), pp. 1942–1958.
[21] A. Walther, *A first order convergence analysis of trust-region methods with inexact Jacobians*, SIAM Journal on Optimization, 19 (2008), pp. 307–325.
[22] J. Ziems and S. Ulbrich, *Adaptive multilevel inexact SQP methods for PDE-constrained optimization*, SIAM Journal on Optimization, 21 (2011), pp. 1–40.

A GALERKIN RADIAL BASIS FUNCTION METHOD FOR NONLOCAL DIFFUSION

STEPHEN D. BOND*, RICHARD B. LEHOUCQ[1], AND STEPHEN T. ROWE[2]

Abstract. We introduce a meshfree Galerkin method for solving nonlocal diffusion problems. Radial basis functions are used to construct an approximation scheme that requires only scattered nodes with no triangulation. A quadrature scheme specific to radial basis functions is implemented to produce a Galerkin radial basis function method that yields fast assembly of a sparse stiffness matrix. We provide numerical evidence for convergence rates using one and two dimensional nonlocal problems.

1. Introduction. Classical diffusion models are formulated as partial differential equations that rely on Fick's first law. However, it has been observed in cases such as diffusion through heterogeneous material that the classical model is not an adequate description of diffusion [13]. Various models have been proposed for these cases of anomalous diffusion, which include models based on fractional spatial derivatives or fractional spatial differential operators, such as the fractional Laplacian. In this paper, we consider numerical methods for solving a model for nonlocal diffusion. The nonlocal model allows for discontinuous functions and also includes the fractional Laplacian as a special case. In this section, we introduce the nonlocal operator and the problem which we propose a numerical method for solving [1, 3].

Let $\Omega \subset \mathbb{R}^n$ denote a bounded, open domain. For $u(x) : \Omega \to \mathbb{R}$, we define

$$\mathscr{L}u(x) := 2\int_{\Omega\cup\Omega_{\mathscr{I}}} \big(u(y)-u(x)\big)\gamma(x,y)\,dy \qquad \forall x \in \Omega \subseteq \mathbb{R}^n, \tag{1.1}$$

where $\gamma : \Omega\cup\Omega_{\mathscr{I}} \times \Omega\cup\Omega_{\mathscr{I}} \to \mathbb{R}$ is a nonnegative symmetric map, i.e., $\gamma(x,y) = \gamma(y,x) \geq 0$. The operator $\mathscr{L}$ is *nonlocal* because the pointwise value of $\mathscr{L}u(x)$ depends on points $y \neq x$. In contrast, the Laplacian, $\Delta u(x)$, requires only information about u at x.

The nonlocal operator (1.1) has applications in a variety of fields besides anomalous diffusion. Examples include image analyses, nonlocal heat conduction, machine learning, and peridynamic mechanics; see the discussion in the paper [3].

Our goal is to solve the steady-state nonlocal diffusion equation. Given $f : \Omega \to \mathbb{R}$, the problem is to find $u : \Omega\cup\Omega_{\mathscr{I}} \to \mathbb{R}$ such that

$$\begin{cases} \mathscr{L}u(x) = f(x) & x \in \Omega \\ u(x) = g(x) & x \in \Omega_{\mathscr{I}}. \end{cases} \tag{1.2}$$

We refer to $\Omega_{\mathscr{I}}$ as the interaction domain, γ as the kernel, and f as the source function. For comparison, the classical steady-state form of the diffusion problem is

$$\begin{cases} \nabla\cdot(\mathbf{C}\nabla u)(x) = f(x) & x \in \Omega \\ u(x) = g(x) & x \in \partial\Omega, \end{cases} \tag{1.3}$$

where $\mathbf{C}$ is the diffusion tensor. Comparing (1.2) and (1.3), we see at least two differences. The first difference is that the nonlocal problem replaces a differential operator with an integral operator. The second difference is that the boundary constraint on $\partial\Omega$ is replaced by a *volume* constraint on $\Omega_{\mathscr{I}}$. Imposing the volume constraint guarantees that the nonlocal problem is well-posed [3].

*Sandia National Laboratories, sdbond@sandia.gov
[1]Sandia National Laboratories, rblehou@sandia.gov
[2]Texas A&M University, srowe@math.tamu.edu

In this note, we propose a Galerkin radial basis function (RBF) method to numerically solve (1.2). This method obviates the need to mesh the region. We also exploit a recently developed Lagrange function quadrature scheme [5] for the necessary integrations. Consequently, our proposed method requires only information at the radial basis function nodes and also yields a straight forward assembly of a sparse stiffness matrix. A conforming discontinuous Galerkin method for a nonlocal diffusion problem was introduced in [3] where the basis functions are given by discontinuous piecewise polynomials. Assembly of this stiffness matrix results in a challenging problem in quadrature for two reasons. The first is that there are $2n$ iterated integrals, and the second is that the regions of integration involve partial element volumes. In contrast, the primary advantage of the Galerkin RBF method is that entries in the stiffness matrix only require a pointwise evaluation of the kernel and multiplication by quadrature weights—complications arising from overlapping partial element volumes are irrelevant. A disadvantage of the use of radial basis functions relative to discontinuous piecewise polynomials includes a "Gibbs phenomenon" at any discontinuities of the solution u for (1.2).

For a fixed $x \in \Omega$, the horizon of γ at x is the radius of support of $\gamma(x,y)$. It is possible for γ to have multiple horizons that depend on x. The integrability of the kernel determines the smoothing action of the inversion of $\mathscr{L}$. We consider only integrable γ with no singularities, which implies that the inversion process does not smooth the data. This should be contrasted with the case of a second order elliptic differential operator, where the solution is two orders smoother than the data. In particular, discontinuous solutions are to be expected for discontinuous source functions.

The meshfree Galerkin method we introduce uses radial basis functions (RBFs). Radial basis functions have been extensively studied for meshfree interpolation and approximation. Radial basis functions have also been applied in a variety of areas besides interpolation of scattered data on subsets of $\mathbb{R}^n$. They have seen notable success in collocation methods for elliptic, parabolic, and hyperbolic partial differential equations and a variety of other PDEs, including stochastic differential equations. RBF Galerkin methods have been investigated, but previously had difficulty with quadrature. In particular, RBF collocation methods are well suited for high dimensional, high smoothness PDEs. In the setting of certain Riemannian manifolds such as the n-sphere $\mathbb{S}^n$, RBF methods can be extended and yield interpolation methods as well as collocation and Galerkin methods for solving partial differential equations on spheres [2, 6, 7, 10, 11, 14].

In section 2, we discuss radial basis functions and interpolation. We motivate their use by considering the question of scattered data interpolation in arbitrary dimensions, followed by a discussion of positive definite and conditionally positive definite functions. We discuss a specific choice of basis, the Lagrange basis, along with a quadrature method that is used in the Galerkin radial basis function (RBF) method we introduce. In section 3, we consider a variational formulation of (1.2). Combining the variational form with radial basis functions and the quadrature technique described in section 2, we propose a meshfree Galerkin RBF method. We then present numerical experiments in both one and two dimensions and discuss the results from the experiments in section 4.

2. Radial Basis Functions. In this section, we define radial basis functions and related concepts and explain their interpolation and approximation properties.

To motivate radial basis functions, we consider the following problem: Let $\Omega \subset \mathbb{R}^n$ and let $\{x_1,...,x_N\} = X \subset \Omega$ be a collection of scattered data sites (referred to as centers or nodes), and let $\{y_1,...,y_N\}$ be given values corresponding to the centers in X. The interpolation of scattered data problem seeks an interpolant $s : \Omega \to \mathbb{R}$ such that $s(x_j) = y_j$ for $j = 1,\ldots,N$. A further modification to the question is to consider finding a subspace $V = \text{span}\{\phi_1,\ldots,\phi_N\}$ of continuous functions such that for *any* collection of scattered sites $\{x_1,\ldots,x_N\}$, a unique interpolant exists in V. In the case of $n = 1$, if we let $V = \text{span}\{1,x,\ldots,x^{N-1}\}$, the interpolation problem has a unique solution. However, by the Mairhuber-Curtis theorem, for dimension $n \geq 2$, the second question has a negative answer. Therefore, to guarantee a unique solution to the interpolation

problem of scattered data, the interpolating functions must depend on the data sites [15].

Let $\Omega \subset \mathbb{R}^n$. We say that $\Phi : \Omega \to \mathbb{R}$ is *radial* if there exists $\varphi : \mathbb{R}^+ \to \mathbb{R}$ such that $\Phi(x) = \varphi(\|x\|)$. Given a set of scattered centers $\{x_1, \ldots, x_N\} = X$ and a radial function Φ, we construct a collection of *radial basis functions* ϕ_j by defining $\phi_j(x) = \Phi(x - x_j) = \varphi(\|x - x_j\|)$. To interpolate a function $f : \Omega \to \mathbb{R}$ on the centers X, we construct an interpolant by enforcing

$$f(x_j) = \sum_{k=1}^{N} c_k \varphi(\|x_k - x_j\|) \qquad j = 1, ..., N,$$

which yields a linear system $Tc = f$ of N equations in N unknowns, with $T_{jk} = \varphi(\|x_k - x_j\|)$ and $f_j = f(x_j)$. For the interpolation problem to have a unique solution for any collection of centers, the interpolation matrix T must be invertible for any collection of centers. The set of functions which generate an invertible matrix for any collection of centers is unknown, but the restriction to the set of functions which generate positive definite matrices for arbitrary sets of centers has been studied. These functions are positive definite functions and they uniquely solve the interpolation problem for any set of centers. They have been completely characterized by Bochner's theorem and its corollaries. For example, a continuous $L^1(\mathbb{R}^n)$ function Φ is positive definite if and only if it is bounded and its Fourier transform is nonnegative and nonvanishing. One such example of a positive definite function is the Gaussian $\varphi(r) = \exp(-\varepsilon r^2)$, for $\varepsilon > 0$ [4, 15].

A popular choice of RBFs are the thin plate spline

$$\varphi(r) = r^2 \log r. \tag{2.1}$$

The approximation and interpolation of the thin plate spline has been studied extensively, but the thin plate spline is only a conditionally positive definite function. Let $\Pi_{(1)}$ denote the space of degree one polynomials with a basis $\{p_1, \ldots, p_{n+1}\}$ in $\mathbb{R}^n$. Given centers $\{x_1, ..., x_N\} = X$, define $P|_X = \text{span}\{\rho_1, \ldots, \rho_{n+1}\}$, where $(\rho_k)_j = p_k(x_j)$. We say that the thin plate spline is conditionally positive definite if and only if for any collection of scattered sites $\{x_1, \ldots, x_N\}$, the quadratic form $\xi^T T \xi$ is positive for all nonzero ξ orthogonal to $P|_X$, with $T_{j,k} = \varphi(\|x_j - x_k\|)$. To ensure that interpolation with the conditionally positive definite thin plate spline is well defined, a linear polynomial must be included in the interpolant. This has the advantage of interpolating scattered data and reproducing constants and linear polynomials. The interpolant is of the form

$$s(x) = \sum_{j=1}^{N} \xi_j \varphi(\|x - x_j\|) + \sum_{k=1}^{n+1} \zeta_k p_k(x),$$

where n is the space dimension. The interpolation linear system is

$$\begin{pmatrix} T & P \\ P^T & 0 \end{pmatrix} \begin{pmatrix} \xi \\ \zeta \end{pmatrix} = \begin{pmatrix} f \\ 0 \end{pmatrix} \tag{2.2}$$

where $T_{j,k} = \varphi(\|x_j - x_k\|)$ is symmetric, $P_{k,l} = p_l(x_k)$, and $f_j = f(x_j)$.

Using (conditionally) positive definite radial basis functions, we can uniquely interpolate arbitrarily scattered data by using translates of a fixed univariate function. The error between a function and its RBF interpolant can be shown to decrease as the density of the centers increases. To quantify the density of the centers $\{x_1, \ldots, x_N\} = X \subset \Omega$, we define the mesh norm (or fill distance)

$$h = \sup_{x \in \Omega} \min_{x_j \in X} \|x - x_j\|; \tag{2.3}$$

see Figure 2.1. The convergence rate of the interpolant to the function depends on the smoothness of both the function and the RBF, with smoother functions being approximated at faster rates. For an example of an error estimate for a thin plate spline RBF in $W_2^m(\Omega)$, the Sobolev space of order m, we have

THEOREM 2.1. *Given certain restrictions on* Ω *and for the thin plate spline in* $W_2^2(\Omega)$ *and function* $f \in W_2^\beta(\Omega)$ *for* $\frac{n}{2} < \beta \leq 2$,

$$\|f - I_X f\|_{W_2^\mu(\Omega)} \leq C h^{\beta-\mu} \|f\|_{W_2^\beta(\Omega)} \qquad 0 \leq \mu < \beta \tag{2.4}$$

where $I_X f$ *denotes the RBF interpolant of* f *on centers in* X *[12].* The restriction $2\beta > n$ guarantees that f is a continuous function, by the Sobolev embedding theorem.

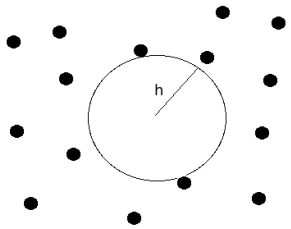
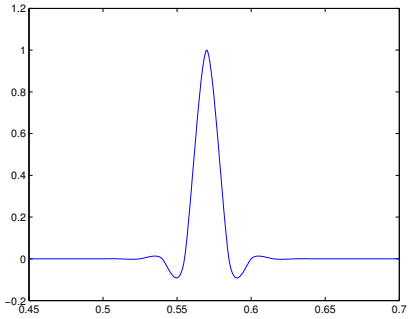
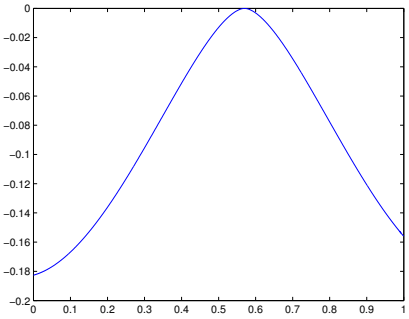


FIG. 2.1. *The geometric interpretation of the mesh norm is the largest ball in* Ω *that does not contain any centers.*



(a) 1D Lagrange Function.

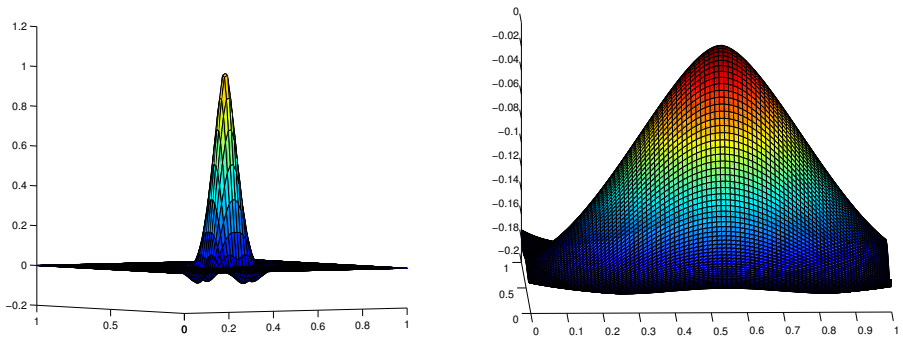


(b) 1D Thin Plate Spline

FIG. 2.2. *(a) A 1D Lagrange function constructed from* 67 *uniformly spaced points is displayed. (b) A 1D thin plate spline centered at* .57 *is displayed.*

Faster convergence rates are known when the function f is smoother than the RBF.

The thin plate spline was chosen over other RBFs due to its approximation properties and because the corresponding Lagrange basis functions enjoy an exponential decay. Approximation theorems of the type in Theorem 2.1 are known for thin plate splines, which are not known for all other RBFs. The Gaussians are known to have spectral convergence, but these results are only known to hold for analytic functions. Unlike the Gaussians and compactly supported Wendland functions, the thin plate spline does not require a "scale parameter" to adapt the RBF to the density of the data. There is no known way of a priori choosing a scale parameter, and this choice influences the conditioning and approximation of the RBF. In contrast, the thin



(a) 2D Lagrange Function. (b) 2D Thin Plate Spline

FIG. 2.3. *(a) A 2D Lagrange function constructed from 169 uniformly spaced points is displayed. (b) A thin plate spline basis function centered at* $(.54, .54)$ *is displayed.*

plate spline Lagrange functions "self-scale" with the data density. Recent results have noted the Lagrange functions of the thin plate spline decays exponentially away from its center, which is not known for other RBFs [8, 9]. There is evidence that the Lagrange functions for the thin plate splines can be replaced with *local* Lagrange functions, which are cheaper to construct than the Lagrange function. The local property has been recently proven for spheres, and current work investigates these methods for domains in $\mathbb{R}^n$ [5, 6, 8]. The local property is due to results on both the decay of the Lagrange functions away from their centers and the decay of the coefficients of the Lagrange functions. Numerical evidence suggests these results can be extended to domains in $\mathbb{R}^n$ for Lagrange functions away from the boundary. The Lagrange functions for Gaussians are not localized and the Gaussians have conditioning problems for even a few thousand points. The compactly supported Wendland functions do have approximation rates analogous to Theorem 2.1, but they require scale parameters and they are not known to have localized Lagrange functions.

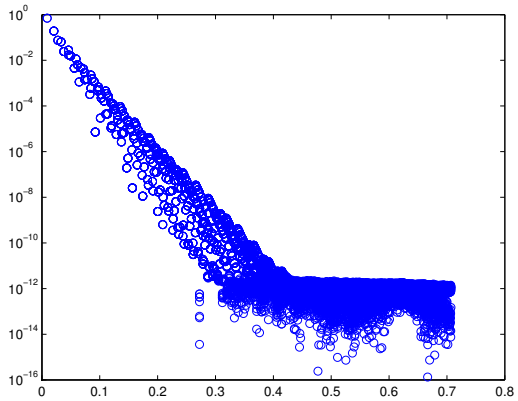


FIG. 2.4. *The exponential decay of a Lagrange function away from its center is displayed. The Lagrange function was constructed using* 1681 *centers and evaluated on* 14400 *points.*

2.1. Lagrange Basis Functions and Quadrature . In this section, we discuss an alternate basis for RBF interpolation and a quadrature method unique to the basis. Given a thin plate spline and a set of centers $\{x_j\}_{j=1}^N$, we can construct a *Lagrange* basis of RBFs, $\{\chi_j\}_{j=1}^N$ that satisfy $\chi_j(x_k) = \delta_{j,k}$ and are given by

$$\chi_j(x) = \sum_{k=1}^{N} \xi_k \varphi(\|x - x_k\|) + \sum_{l=1}^{n+1} \zeta_l p_l(x).$$

The Lagrange function χ_j is constructed by solving the linear system (2.2) with $f_k = \delta_{j,k}$. Therefore, the interpolation of a function f is given by

$$I_X f(x) = \sum_{j=1}^{N} f(x_j)\chi_j(x).$$

Figures 2.2–2.3 display a thin plate spline and Lagrange basis function and explain the interest in the Lagrange function basis. Figure 2.4 demonstrates that the Lagrange basis function is essentially of compact support due the associated exponential decay. A quadrature scheme can be constructed by using Lagrange functions. Given $F : \Omega \to \mathbb{R}$ and centers $\{x_j\}_{j=1}^N$, we define for $S \subset \Omega$

$$\int_S F(x)\,dx \approx \sum_{x_j \in S} F(x_j) w_j, \qquad w_j = \int_\Omega \chi_j(x)\,dx. \tag{2.5}$$

When $S = \Omega$, we can derive the following quadrature error estimate:

LEMMA 2.2. *Let $\Omega \subset \mathbb{R}^n$ and $X = \{x_j\}_{j=1}^N$, $\{w_j\}_{j=1}^N$ be the quadrature weights in* (2.5)*, and $F(x) \in W_2^\beta(\Omega)$ for $\frac{n}{2} < \beta \le 2$. Then, the quadrature error is bounded by*

$$\left| \int_\Omega F(x)\,dx - \sum_{j=1}^{N} F(x_j) w_j \right| \le C h^\beta \|F\|_{W_2^\beta(\Omega)}.$$

Proof. This follows by an application of Theorem 2.1, (2.5), and the Cauchy-Schwarz inequality.

$$\begin{aligned} \left| \int_\Omega F(x)\,dx - \sum_{j=1}^{N} F(x_j) w_j \right| &= \left| \int_\Omega F(x) - \sum_{j=1}^{N} F(x_j)\chi_j(x)\,dx \right| \\ &\le \sqrt{\mu(\Omega)} ||F - I_X F||_{L^2(\Omega)} \\ &\le C \sqrt{\mu(\Omega)} h^\beta ||F||_{W_2^\beta(\Omega)} \end{aligned}$$

where $\mu(\Omega)$ is the measure of Ω, which we absorb into the constant C to achieve the result. □ A scheme for directly computing the quadrature weights without computing the χ_j is described in Section 3.1.

3. Variational Formulation. In this section, we derive a variational formulation of (1.2), which then can be solved using a Galerkin method. We define the bilinear form $a(u,v)$ by

$$a(u,v) = \int_{\Omega \cup \Omega_{\mathscr{I}}} \int_{\Omega \cup \Omega_{\mathscr{I}}} \big(u(x) - u(y)\big)\big(v(x) - v(y)\big)\gamma(x,y)\,dy\,dx.$$

We define the energy functional E by

$$E(u) = \frac{1}{2}a(u,v) - \int_{\Omega} uf\,dx \quad \text{subject to } u = g \text{ over } \Omega_{\mathscr{I}}. \tag{3.1}$$

In [3], the problem of finding the minimum of the energy functional was shown to be well-posed for u in an energy constrained space $L^2_c(\Omega \cup \Omega_{\mathscr{I}})$. In contrast, we minimize the functional by the method of Lagrange multipliers because the RBF basis is not contained in the energy constrained space. The Lagrangian is defined as

$$L(u,\lambda) = E(u) + \int_{\Omega_{\mathscr{I}}} \lambda(y)\big(u(y) - g(y)\big)\,dy,$$

where $\lambda \in L^2(\Omega_{\mathscr{I}})$ is the Lagrange multiplier. Then, for every $v \in L^2(\Omega \cup \Omega_{\mathscr{I}})$ and $w \in L^2(\Omega_{\mathscr{I}})$, we minimize the Lagrangian by

$$\frac{d}{dt}L(u+tv,\lambda)\big|_{t=0} = \frac{d}{dt}E(u+tv)\big|_{t=0} + \int_{\Omega_{\mathscr{I}}} \lambda(y)g(y)\,dy = 0$$
$$\frac{d}{dt}L(u,\lambda+tw)\big|_{t=0} = \int_{\Omega_{\mathscr{I}}} w(y)\big(u(y) - g(y)\big)\,dy.$$

Computing these derivatives, the variational form of the problem is to find $u \in L^2(\Omega \cup \Omega_{\mathscr{I}})$ such that for each $v \in L^2(\Omega \cup \Omega_{\mathscr{I}})$ and each $w \in L^2(\Omega_{\mathscr{I}})$,

$$\begin{cases} a(u,v) + \int\limits_{\Omega_{\mathscr{I}}} \lambda(y)v(y)\,dy = \int\limits_{\Omega} v(x)f(x)\,dx \\ \int\limits_{\Omega_{\mathscr{I}}} w(y)u(y)\,dy = \int\limits_{\Omega_{\mathscr{I}}} w(y)g(y)\,dy. \end{cases} \tag{3.2}$$

We discretize this system by choosing finite dimensional subspaces $U_h \subset L^2(\Omega \cup \Omega_{\mathscr{I}})$ and $\Lambda_h \subset L^2(\Omega_{\mathscr{I}})$. Let $U_h = \operatorname{span}\{\phi_i\}_{i=1}^{N}$ and $\Lambda_h = \operatorname{span}\{\psi_k\}_{k=1}^{N_I}$. Then, we express the discrete solution and Lagrange multipliers in these bases as

$$u_h = \sum_{i=1}^{N} \alpha_i \phi_i \qquad \lambda_h = \sum_{k=1}^{N_I} \beta_k \psi_k.$$

To construct the linear system to solve for the coefficients of u_h and λ_h, we insert u_h and λ_h into (3.2), and we choose $v \in U_h$ and $w \in \Lambda_h$. The resulting linear system is

$$\begin{pmatrix} A & B \\ B^T & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} b \\ c \end{pmatrix}, \tag{3.3}$$

where the entries in the matrices are given by

$$A_{i,j} = \int_{\Omega \cup \Omega_{\mathscr{I}}} \int_{\Omega \cup \Omega_{\mathscr{I}}} \big(\phi_j(x) - \phi_j(y)\big)\big(\phi_i(x) - \phi_i(y)\big)\,\gamma(x,y)\,dy\,dx \tag{3.4a}$$

$$b_i = \int_{\Omega} \phi_i f\,dy, \qquad B_{i,k} = \int_{\Omega_{\mathscr{I}}} \psi_k \phi_i\,dy, \qquad c_k = \int_{\Omega_{\mathscr{I}}} \psi_k g\,dy. \tag{3.4b}$$

3.1. Galerkin Radial Basis Function Method. We propose a Galerkin RBF method for two reasons. First, the Galerkin RBF method leads to a sparse stiffness matrix by exploiting a Lagrange function quadrature rule. Otherwise, since the basis functions are globally supported, the stiffness matrix is dense. The second reason to consider this method is the ease of assembly of the stiffness matrix. In contrast with a finite element method using discontinuous piecewise polynomials, assembly of the Galerkin RBF stiffness matrix only requires pointwise evaluations of the kernel and multiplication by quadrature weights. However, we must overcome the difficulty of computing the quadrature weights, which we discuss in the following section.

We construct a Galerkin RBF method by using Lagrange function quadrature (2.5) on each of the entries in the matrices and vectors in (3.4a) and (3.4b). Choose the approximation space $U_h = \text{span}\{\chi_i\}_{i=1}^{N}$, where χ_i are the Lagrange functions of the RBF $\varphi(r) = r^2 \log r$ over some collection of centers $X \subset \Omega \cup \Omega_{\mathscr{I}}$. Let $\Lambda_h = \text{span}\{\chi_{j(k)}\}_{k=1}^{N_I}$, where the function $j : \{1,\ldots,N_I\} \to \{1,\ldots,N\}$ selects each index in $\{1,...,N\}$ such that $\chi_{j(k)}$ is centered at a point in $\Omega_{\mathscr{I}}$.

A practical quadrature method is realized by a slight modification of a recent scheme proposed by Fuselier, Hangelbroek, Narcowich, Ward, and Wright [5]. The necessary quadrature weights w_j, see (2.5), can be constructed by solving the linear system

$$\begin{pmatrix} T & P \\ P^T & 0 \end{pmatrix} \begin{pmatrix} w \\ d \end{pmatrix} = \begin{pmatrix} \nu \\ \eta \end{pmatrix} \tag{3.5}$$

where $\nu_k = \int_\Omega \varphi(\|x - x_k\|)dx$ and $\eta_l = \int_\Omega p_l(x)dx$; see the paper [5] for further details. This choice of quadrature scheme has important practical considerations; see §4.3.

In (3.4), we substitute $\phi_i(x) = \chi_i(x)$ and $\psi_k(x) = \chi_{j(k)}(x)$. We approximate the integrals by using the Lagrange function quadrature. Using this quadrature rule, the entries in $A_{i,j}$ are approximated by

$$A_{i,j} \approx 2\delta_{i,j} w_i \int_{\Omega \cup \Omega_{\mathscr{I}}} \gamma(x, x_i)\, dx - 2 w_i w_j \gamma(x_i, x_j). \tag{3.6}$$

We apply the quadrature scheme to the other values in (3.4) and compute

$$b_i \approx f(x_i) w_i \, \chi^{\Omega}(x_i), \qquad B_{i,k} \approx \delta_{i,j(k)} w_i \, \chi^{\Omega_{\mathscr{I}}}(x_i), \qquad c_k \approx g(x_k) w_k \, \chi^{\Omega_{\mathscr{I}}}(x_k),$$

where χ^S is the indicator function for a set S defined by

$$\chi^S(x) = \begin{cases} 1 & x \in S \\ 0 & x \notin S. \end{cases}$$

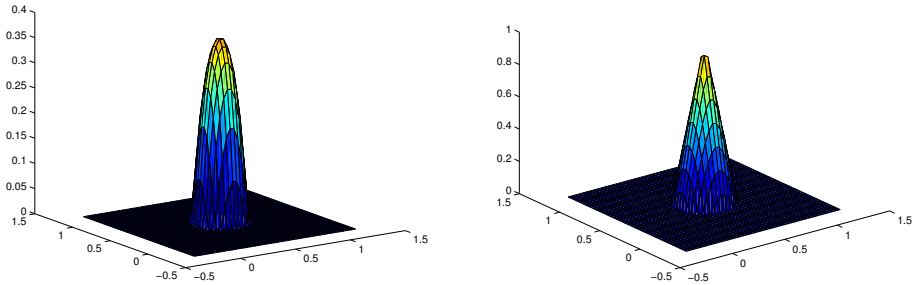
The sparsity of the Galerkin RBF matrix (3.6) depends on the mesh norm h and the horizon ε of the kernel. For centers such that $\|x_i - x_j\| \geq \varepsilon$, $A_{ij} = 0$. If the number of centers is increased, then the number of centers such that $\|x_i - x_j\| < \varepsilon$ can increase, which increases the number of nonzero entries per row. If the Lagrange function quadrature is not used, then a dense stiffness matrix is generated due to the global support of the Lagrange function.

4. Numerical Results. We present results from numerical experiments in both one and two dimensions. We present the convergence rates in terms of a loglog plot of the L^2 error versus the mesh norm h, see (2.3). Since there is currently no approximation theory for the Galerkin RBF method, we also present interpolation error rates on uniformly spaced centers as a comparison. The paper [3] establishes convergence rates for a finite element discretization using a piecewise discontinuous polynomial basis.

For all tests, we assume zero Dirichlet volume constraints (i.e., $g(x) = 0$ in (1.2)). In both one and two dimensions, we consider an infinitely smooth kernel and a continuous, but not everywhere differentiable, triangular kernel

$$\gamma(x,y) = \begin{cases} \exp(-\frac{1}{1-\varepsilon^{-2}\|x-y\|^2})\chi_{\{\|x-y\|<\varepsilon\}}(x,y) & \text{Smooth kernel} \\ (1-\frac{1}{\varepsilon}\|x-y\|)\chi_{\{\|x-y\|<\varepsilon\}}(x,y) & \text{Triangular kernel} \end{cases} \tag{4.1}$$

where $\chi_{\{\|x-y\|<\varepsilon\}}$ is the indicator function for the set $\{(x,y) : \|x-y\| < \varepsilon\}$; see Figure 4.1. We consider kernel functions of varying smoothness to observe how the smoothness of the kernel affects the convergence rate of the RBF solution. The smoothness of the kernel affects the quadrature error induced by the Lagrange function quadrature method when assembling the stiffness matrix, so we expect lower convergence rates for lower smoothness kernel functions.



(a) Mollifier kernel (b) Triangular kernel

FIG. 4.1. *(a) The mollifier kernel $\gamma(x,y)$ with $y = (.5,.5)$ and $\varepsilon = \frac{1}{4}$. (b) The triangular kernel with $y = (.5,.5)$ and $\varepsilon = \frac{1}{4}$.*

4.1. 1D Experiments. We let $\Omega = [0,1]$ and $\Omega_{\mathscr{I}} = (-\frac{1}{4},0) \cup (1,\frac{5}{4})$, and we set $\varepsilon = \frac{1}{4}$. We choose two solutions and we manufacture the corresponding source function for each. We consider continuously differentiable and discontinuous solutions u_1 and u_2, respectively

$$u_1(x) = \big(1-\cos(2\pi x)\big)\chi_{\Omega}(x) \qquad u_2(x) = \chi_{\{0\le x\le \frac{1}{2}\}}(x).$$

For both u_1 and u_2, we solve the Galerkin RBF problem for both the smooth and triangular kernels, respectively. We tested using uniformly spaced centers with mesh norm $h = .005, .0025, .00125$, and $.0005$. For a second set of tests, we used the same sets of uniformly spaced centers, but the interior points were perturbed by a random number bounded in magnitude by $\frac{h}{5}$. Figures 4.2 and 4.3 display convergence rate plots for the non-uniformly spaced centers experiment. Table 4.1 displays convergence rates for the uniformly spaced experiments and the scattered experiments. For comparison, we display the convergence rate for interpolation using the uniformly spaced centers. As can be seen in Table 4.2, the condition number of the stiffness matrix is not increasing as the mesh norm decreases. In comparison, the paper [3] establishes that the condition number of the stiffness matrix does not increase as the mesh size decreases for a finite element discretization using a piecewise discontinuous polynomial basis.

TABLE 4.1
Convergence rates for 1D experiments are displayed. The interpolation rates use the uniformly spaced centers.

	Uniformly spaced		Non-uniformly spaced		
	Smooth	Triangular	Smooth	Triangular	Interpolation
u_1	2.2	1.9	2.2	1.9	2.3
u_2	.52	.51	.53	.52	.51

TABLE 4.2
For 1D experiments, the mesh norm h, number of rows n of the stiffness matrix (3.3)*, and the reciprocal condition number for* (3.3) *for both the smooth and triangular kernel is displayed.*

		Reciprocal Condition Number	
h	n	Smooth	Triangular
5.00e-3	303	2.43e-3	4.03e-3
2.50e-3	603	2.40e-3	3.95e-3
1.25e-3	1203	2.39e-3	3.98e-3
5.00e-4	3003	2.38e-3	3.93e-3

4.2. 2D Experiments. We let $\Omega = [0,1] \times [0,1]$ and $\Omega_{\mathscr{I}} = ([-\frac{1}{4}, \frac{5}{4}] \times [-\frac{1}{4}, \frac{5}{4}]) \backslash \Omega$. We consider two solutions and we manufacture the corresponding source function for each. We consider the continuous function w_1 and the discontinuous function w_2

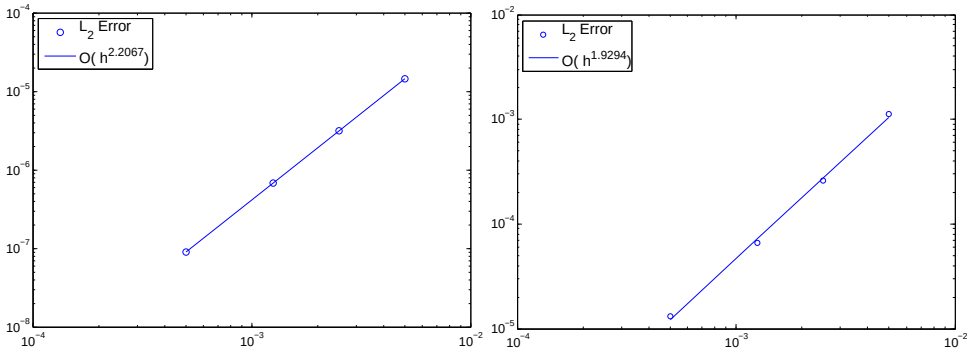
$$w_1(x,y) = \sin(2\pi x)\sin(2\pi y)\chi_{\Omega}(x,y)$$
$$w_2(x,y) = (1-\cos(2\pi x))(1-\cos(2\pi y))\chi_{\Omega_1}(x,y)$$

with $\Omega_1 = [0, \frac{1}{2}] \times [0,1]$.

For both w_1 and w_2, we solve the Galerkin RBF problem for both the smooth kernel and the triangular kernel. We tested using uniformly spaced centers with mesh norm $h = .06, .04, .02$, and $.014$. For a second set of tests, we used the same sets of uniformly spaced centers, but the interior points were perturbed by a random vector bounded in magnitude by $\frac{2h}{15}$. Figures 4.4 and 4.5 displays convergence rate plots for the non-uniformly spaced centers experiment. Table 4.3 displays convergence rates for the uniformly spaced experiments and the scattered experiments. For comparison, we display the convergence rate for interpolation using the uniformly spaced centers. The L^2 error is computed by evaluating the solution on a set of 14400 nodes. The evaluation point set X_e is constructed by taking tensor products of Gauss-Legendre nodes on $[-\frac{1}{4}, \frac{5}{4}]$. In the case of non-uniformly spaced centers, the mesh norm h is approximated by $\max_{x \in X_e} \min_{x \in X} \|x - x_e\|$. As can be seen in Table 4.4, the condition number of the stiffness matrix is not increasing as the mesh norm decreases

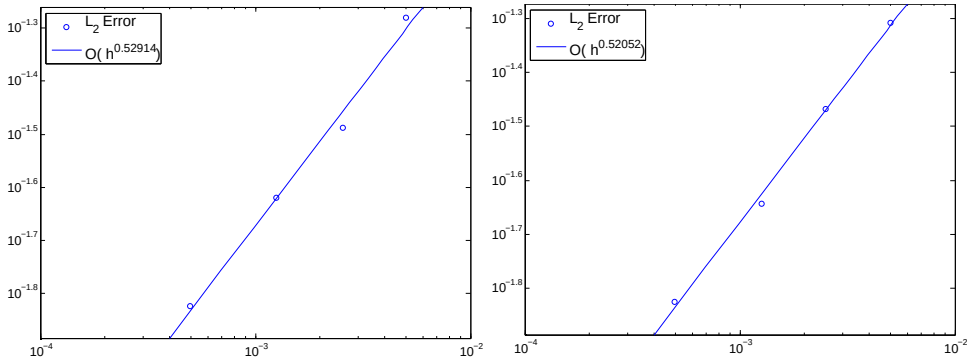
4.3. Computational Issues. In this section, we discuss some computational issues and aspects of the Galerkin RBF method. We discuss the costs of assembly, computation of L^2 errors, and construction of the source function for the experiments in section 4.

Assembly of the stiffness matrix requires the construction of the Lagrange function quadrature weights. The quadrature weights are constructed by solving equation (3.5), which is a dense, symmetric linear system of size $N \times N$ where N is the number of basis functions. Future work will investigate ways of computing the weights faster. Assembly of the sparse stiffness matrix with the quadrature weights follows by the formula



(a) $\|u_1 - u_h\|_{L^2(\Omega\cup\Omega_{\mathscr{I}})}$ for smooth kernel (b) $\|u_1 - u_h\|_{L^2(\Omega\cup\Omega_{\mathscr{I}})}$ for triangular kernel

FIG. 4.2. *1D experiments for the continuous solution u_1 with both kernel functions using non-uniformly spaced centers are displayed. The horizontal axis is the log of h and the vertical axis is the log of the L^2 error of the solution and the approximate solution u_h.*



(a) $\|u_2 - u_h\|_{L^2(\Omega\cup\Omega_{\mathscr{I}})}$ for smooth kernel (b) $\|u_2 - u_h\|_{L^2(\Omega\cup\Omega_{\mathscr{I}})}$ for triangular kernel

FIG. 4.3. *1D experiments for the discontinuous solution u_2 with both kernel functions using non-uniformly spaced centers are displayed. The horizontal axis is the log of h and the vertical axis is the log of the L^2 error of the solution and the approximate solution u_h.*

(3.6), which requires pointwise evaluations of the kernel and multiplication by the computed quadrature weight.

The L^2 error between the Galerkin solution u_h and the solution u is computed on the set $\Omega\cup\Omega_{\mathscr{I}}$ using Gauss-Legendre quadrature for 1D problems and tensor products of Gauss-Legendre nodes in 2D. For 1D experiments, 10000 Gauss-Legendre nodes are used to compute the error. In 2D, 14400 nodes in total are used, which are formed from products of 120 Gauss-Legendre nodes over $[-\frac{1}{4},\frac{5}{4}]$. For a 2D discontinuous solution, the domain is split into two sets, Ω_1 and Ω_2, on which u is continuous. On each set, products of 1D Gauss-Legendre quadrature weights are constructed and the L^2 error is computed by

$$\|u-u_h\|^2_{L^2(\Omega\cup\Omega_{\mathscr{I}})} = \|u-u_h\|^2_{L^2(\Omega_1)} + \|u-u_h\|^2_{L^2(\Omega_2)}.$$

For the experiments in §-4, we require pointwise evaluations of the source function f. We manufacture

TABLE 4.3
Convergence rates for 2D experiments are displayed. The interpolation rates use the uniformly spaced centers.

	Uniformly spaced		Non-uniformly spaced		
	Smooth	Triangular	Smooth	Triangular	Interpolation
w_1	1.49	1.49	1.5	1.5	1.5
w_2	.56	.55	.51	.55	.56

TABLE 4.4
For 2D experiments, the mesh norm h, number of rows n of the stiffness matrix (3.3)*, and the reciprocal condition number for* (3.3) *for both the smooth and triangular kernel are displayed.*

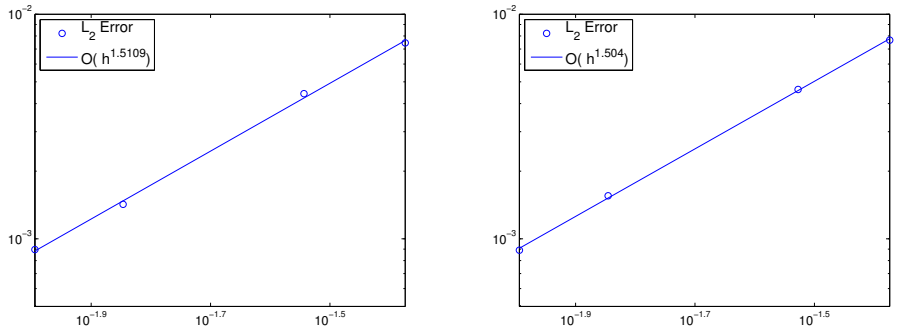
		Reciprocal condition number	
h	n	Smooth	Triangular
4.24e-2	1096	2.10e-3	5.42e-3
2.82e-2	2263	2.05e-3	4.93e-3
1.41e-2	9052	2.10e-3	5.03e-3
9.90e-3	18144	2.07e-3	5.01e-3

$f(x_i)$ by computing $\mathscr{L}u(x_i)$ by (1.2). For one dimensional experiments, this is computed by using MATLAB's integral function. The integrand is written as an anonymous function, which is then integrated from $\max(-\varepsilon, x_i-\varepsilon)$ to $\min(x_i+\varepsilon, 1+\varepsilon)$. For two dimensional experiments, the kernel is supported on a ball of radius ε. Converting the integral to polar coordinates, we use tensor product Gauss-Legendre quadrature weights over the rectangular region $[0,\varepsilon]\times[0,2\pi]$. The right hand side vector is assembled by $b_i = f(x_i)w_i$, where w_i is the Lagrange function quadrature weight. It has been observed that if f is discontinuous and x_i is placed on or near the discontinuity, the approximation $b_i = f(x_i)w_i$ did not perform well in experiments. For these nodes, we instead apply Gauss-Legendre quadrature to compute $b_i = \sum f(y_l)q_l$, where y_l are quadrature nodes and q_l are quadrature weights.

The RBF Galerkin solution u_h suffers from a "Gibbs phenomenon" at a point of discontinuity of u. In a neighborhood of the discontinuity, u_h overshoots the values of u; see Figure 4.6. As the mesh norm decreases, the width of the overshoot decreases, although the height does not.

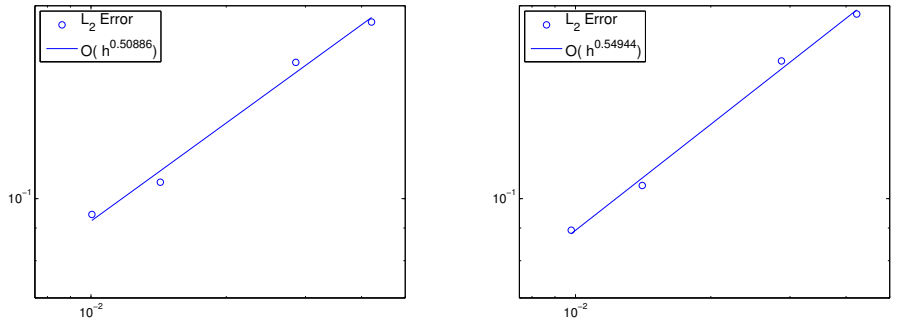
5. Conclusions and Future Work. We developed a Galerkin method for nonlocal diffusion by using Lagrange functions of radial basis functions and a recently developed quadrature method. We observed numerical evidence of L^2 convergence for both continuous and discontinuous solutions using kernel functions of varying smoothness in one and two dimensions. Future work will experiment with three dimensional problems. The construction of the quadrature weights is computationally intensive for small h in three dimensional problems. As h decreases to zero, the number of rows of the dense quadrature weight system (3.5) increases as $N = O(h^{-3})$ for three dimensional problems in contrast to $N = O(h^{-2})$ for two dimensional problems. Regardless of the dimension, the sparse stiffness matrix is assembled by pointwise evaluations of a radial kernel and multiplication by the computed quadrature weights. Further work is required to speed up the computation of the quadrature weights, establish error estimates, consider kernel functions with multiple horizons, and consider anisotropic kernels.

6. Acknowledgements. The authors would like to thank Francis Narcowich and Joseph Ward of Texas A&M University for advice and assistance on the project. Their expertise was invaluable for shaping the Galerkin RBF method. The third author would like to thank Sandia National Laboratories for the fellowship support and summer internship.



(a) $\|w_1 - w_h\|_{L^2(\Omega \cup \Omega_{\mathscr{I}})}$ for smooth kernel (b) $\|w_1 - w_h\|_{L^2(\Omega \cup \Omega_{\mathscr{I}})}$ for triangular kernel

FIG. 4.4. *2D experiments for the continuous solution w_1 with both kernel functions using non-uniformly spaced centers are displayed. The horizontal axis is the log of h and the vertical axis is the log of the L^2 error of the solution and the approximate solution w_h.*



(a) $\|w_2 - w_h\|_{L^2(\Omega \cup \Omega_{\mathscr{I}})}$ for smooth kernel (b) $\|w_2 - w_h\|_{L^2(\Omega \cup \Omega_{\mathscr{I}})}$ for triangular kernel

FIG. 4.5. *2D experiments for the discontinuous solution w_2 with both kernel functions using non-uniformly spaced centers are displayed. The horizontal axis is the log of h and the vertical axis is the log of the L^2 error of the solution and the approximate solution w_h.*

REFERENCES

[1] N. BURCH AND R. B. LEHOUCQ, *Classical, nonlocal, and fractional diffusion equations on bounded domains*, International Journal for Multiscale Computational Engineering, 9 (2011), pp. 661–674.

[2] I. CIALENCO, G. E. FASSHAUER, AND Q. YE, *Approximation of stochastic partial differential equations by a kernel-based collocation method*, International Journal of Computer Mathematics, 89 (2012), pp. 2543–2561.

[3] 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.

[4] G. E. FASSHAUER, *Meshfree Approximation Methods with MATLAB*, World Scientific, 2007.

[5] E. J. FUSELIER, T. HANGELBROEK, F. J. NARCOWICH, J. D. WARD, AND G. B. WRIGHT, *Kernel based quadrature on spheres and other homogeneous spaces*, Numerische Mathematik, (in press).

[6] ———, *Localized bases for kernel spaces on the unit sphere*, SIAM J. Numer. Anal., (in press).

[7] Q. T. L. GIA, I. H. SLOAN, AND H. WENDLAND, *Multiscale RBF collocation for solving PDEs on spheres*, International Journal of Computer Mathematics, 121 (2012), pp. 99–125.

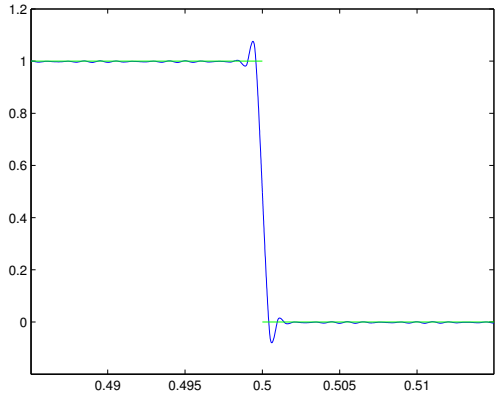


FIG. 4.6. *A step function u (green) and the Galerkin RBF approximation u_h (blue) are displayed. The overshoot and undershoot of u_h occurs to the left and right of the discontinuity at .5.*

[8] T. HANGELBROEK, *The penalized Lebesgue constant for surface spline interpolation*, Proc. Ameri. Math Soc., 140 (2012), pp. 173–187.
[9] T. HANGELBROEK, F. J. NAROWICH, AND J. D. WARD, *Kernel approximation on manifolds I: Bounding the Lebesgue constant*, SIAM Journal on Mathematical Analysis, 42 (2010), pp. 1732–1760.
[10] E. J. KANSA, *Multiquadrics- A scattered data approximation scheme with applications to computational fluid dynamics- II solutions to parabolic, hyperbolic, and elliptic partial differential equations*, Computers & Mathematics with Applications - COMPUT MATH APPL, 19 (1990), pp. 147–161.
[11] F. J. NARCOWICH, S. T. ROWE, AND J. D. WARD, *A meshless Galerkin method on spheres*. preprint.
[12] F. J. NARCOWICH, J. D. WARD, AND H. WENDLAND, *Sobolev error estimates and a Bernstein inequality for scattered data interpolation via radial basis functions*, Constructive Approximation, 24 (2006), pp. 175–186.
[13] S. P. NEUMAN AND D. M. TARTAKOVSKY, *Perspective on theories of non-Fickian transport in heterogeneous media*, Advances in Water Resources, 32 (2009), pp. 670–680.
[14] H. WENDLAND, *Meshless Galerkin methods using radial basis functions*, Mathematics of Computation- Math. Comput., 68 (1999), pp. 1521–1531.
[15] ———, *Scattered Data Approximation*, Cambridge University Press, 2005.

INCOMPRESSIBLE SMOOTHED PARTICLE HYDRODYNAMICS

KAI YANG*, NATHANIEL TRASK[1], AND MICHAEL L. PARKS[2]

Abstract. Incompressible Smoothed Particle Hydrodynamics has its advantage in solving the Navier-Stokes equations due to its stability with respect to the time step size. However, it has the corresponding demand in fast linear solvers. We have implemented IMPLICITSPH package within LAMMPS and incorporated Trilinos through a Solver Manager interface in order to solve the linear systems efficiently. A numerical example of Taylor-Green vortex is shown for validation purpose.

1. Introduction. Smoothed Particle Hydrodynamics (SPH) was first introduced in [5,11] and has been gaining recognition as a simulation tool. SPH can be used to simulate fluid motion, heat conduction, matter diffusion and elasticity, etc. [14]

In this paper, we consider the application of SPH in an incompressible Newtonian fluid. Namely, we want to solve the Navier-Stokes (NS) equations using SPH

$$\begin{cases} \dfrac{d\mathbf{u}}{dt} + \dfrac{1}{\rho}\nabla p - \dfrac{\eta}{\rho}\Delta\mathbf{u} = \mathbf{f} & \text{in } \Omega, \\ \nabla\cdot\mathbf{u} = 0 & \text{in } \Omega, \\ \mathbf{u} = 0 & \text{on } \partial\Omega. \end{cases} \tag{1.1}$$

For this type of problem, a large sound speed is usually set in order to apply compressible fluid model [12, 13]. This approach is called weakly compressible SPH (WCSPH). The error introduced by the large sound speed can be controlled at the expense of a stiffer CFL condition [2]. WCSPH is consistent only for steady-state flows or for large sound speed, and at low Reynolds number the coupled system become hyperbolic requiring appropriate non-reflecting pressure boundary conditions [9]. Alternatively, Cummins and Rudman [3] used projection method to solve incompressible SPH (ISPH) by enforcing incompressibility constraint. While this increases the computational cost and complexity, it removes the artificial stability constraint imposed in WCSPH and, therefore, allows for larger time steps. Some more recent investigations [7, 10, 17] also showed the effectiveness of ISPH.

In this proceeding paper, we first present some basic SPH discretization of differential operators and the second order projection method, which is used to decompose the NS equations into Helmholtz equations, pressure Poisson equations and a correction step. Then we introduce the SPH discretization of the NS equations and the corresponding pseudo code and algorithm. We implement the algorithms in the IMPLICITSPH module within LAMMPS, employing Trilinos to solve linear systems. In the numerical examples we test the IMPLICITSPH package with the Taylor-Green vortex problem. The delaying pattern of the numerical is shown for validation purpose.

2. Basics of SPH . The SPH interpolant [14] of a quantity $A(\cdot)$ is defined by

$$A_I(\mathbf{r}) = \int A(s)W_h(\mathbf{r}-\mathbf{s})ds,$$

where $W_h(r)$ is a smooth interpolating kernel.

This integral is approximated by summation interpolant over mass elements

*The Pennsylvania State University, yangkai1001@gmail.com
[1]Brown University, nathaniel_trask@brown.edu
[2]Sandia National Laboratories, mlparks@sandia.gov

$$A_s(\mathbf{r}) = \sum_i m_i \frac{A_i}{\rho_i} W_h(\mathbf{r} - \mathbf{r}_i).$$

Monaghan has the following discretization for derivatives [12]

$$\begin{cases} \left\langle \dfrac{\nabla P}{\rho} \right\rangle_i = \displaystyle\sum_j m_j \left(\frac{P_j}{\rho_j^2} + \frac{P_i}{\rho_i^2} \right) \nabla_i W_{ij} \\ \left\langle \dfrac{1}{\rho} \nabla \cdot (\kappa \nabla T) \right\rangle_i = \displaystyle\sum_j \frac{m_j}{\rho_i \rho_j} \frac{4\kappa_i \kappa_j}{\kappa_i + \kappa_j} (T_i - T_j) F_{ij} \end{cases} \tag{2.1}$$

where

$$\nabla_i W_{ij} := \frac{\partial W(\mathbf{r}_j - \mathbf{r}_j)}{\partial \mathbf{r}_i}$$

and F_{ij} is defined by

$$\nabla_i W_{ij} = \mathbf{r}_{ij} F_{ij}.$$

The discretization of ∇P is from the study of SPH for Euler equations. $\nabla \cdot (\kappa \nabla T)$ is from the model of heat conduction and Morris et al [15] applied this discretization to viscous term in the NS equations

$$\left\langle \frac{1}{\rho} \nabla \cdot (\mu \nabla \mathbf{u}) \right\rangle_i = \sum_j \frac{m_j (\mu_i + \mu_j) \mathbf{r}_{ij} \cdot \nabla_i W_{ij}}{\rho_a \rho_b (\mathbf{r}_{ij}^2 + 0.01 h^2)} \mathbf{u}_{ij}.$$

Hu and Adams [6] proposed particle-averaged discretization in the study of multiphase flow

$$\begin{cases} \left\langle \dfrac{1}{\rho} \nabla p \right\rangle_i = \dfrac{1}{m_i} \displaystyle\sum_j \left(\frac{p_i}{\sigma_i^2} + \frac{p_j}{\sigma_j^2} \right) \nabla_i W_{ij} \\ \left\langle \dfrac{1}{\rho} \nabla \cdot (\eta \nabla \mathbf{u}) \right\rangle_i = \dfrac{1}{m_i} \displaystyle\sum_j \frac{2\eta_i \eta_j}{\eta_i + \eta_j} \left(\frac{1}{\sigma_i^2} + \frac{1}{\sigma_j^2} \right) \frac{\mathbf{u}_{ij}}{|\mathbf{r}_{ij}|^2} (\nabla_i W_{ij} \cdot \mathbf{r}_{ij}), \end{cases} \tag{2.2}$$

where the inverse particle volume σ_i is computed by $\sigma_i = \sum_j W_{ij}$.

3. Projection Methods. We want to solve the NS equations (1.1) by the second order projection method [8], which decomposes the NS equations into the following equations

- Helmholtz equation

$$\begin{cases} \dfrac{\mathbf{u}^* - \mathbf{u}^n}{\Delta t} = -\nabla p^n + \dfrac{\nu}{2} \Delta (\mathbf{u}^* + \mathbf{u}^n), & \text{in } \Omega \\ \mathbf{u}^* = 0, & \text{on } \partial\Omega \end{cases} \tag{3.1}$$

- Pressure Poisson equation

$$\begin{cases} \Delta p^* = \dfrac{\nabla \cdot \mathbf{u}^*}{\Delta t}, & \text{in } \Omega \\ \dfrac{\partial p^*}{\partial n} = 0, & \text{on } \partial\Omega \end{cases} \tag{3.2}$$

- pressure and velocity correction.

$$\begin{cases} p^{n+1} = p^n + p^* \\ \mathbf{u}^{n+1} = \mathbf{u}^* - \Delta t \nabla p^*. \end{cases} \tag{3.3}$$

Boundary conditions

- Dirichlet boundary condition:
  Morris et al. [15] proposed a simple way to apply Dirichlet boundary condition for SPH. For simplicity we only introduce no-slip boundary condition.
  For any particle a from the domain of interest, velocity of ghost particles outside the computational domain is extrapolated using the following formula

$$\mathbf{v}_b = -\frac{d_b}{d_a}\mathbf{v}_a.$$

  d_a and d_b are the distance from particle a and particle b to boundary, respectively.
- Neumann boundary condition: skip ghost particles outside the domain
- Periodic boundary condition: have the particles interacting with its ghost

4. SPH discretization of NS equations. We apply the standard SPH discretization to calculate the smoothed interpolant at the i^{th} SPH particle

$$\hat{\mathbf{u}}(\mathbf{x}_i) = \sum_{j \in \mathscr{N}_i} \mathbf{u}_j W(||\mathbf{x}_i - \mathbf{x}_j||_2) V_j \tag{4.1}$$

Where W is a 1D kernel function that satisfies the property that $\int_{R^d} W(\mathbf{r}) d\mathbf{r} = 1$, where d is the dimension of the problem. For our applications we use the quintic spline function which has been demonstrated to provide improved stability and accuracy in simulating viscous flows.

$$W_h(r) = C \begin{cases} (3-\frac{r}{h})^5 - 6(2-\frac{r}{h})^5 + 15(1-\frac{r}{h})^5 & : \frac{r}{h} \in [0,1] \\ (3-\frac{r}{h})^5 - 6(2-\frac{r}{h})^5 & : \frac{r}{h} \in [1,2] \\ (3-\frac{r}{h})^5 & : \frac{r}{h} \in [2,3] \\ 0 & : \frac{r}{h} \notin [0,3] \end{cases} \tag{4.2}$$

Where the constant C is selected such that $\int W_h = 1$.

Using this kernel, we compute the particle volumes via

$$(\hat{V}_i)^{-1} = \sum_{j \in \mathscr{N}_i} W(||\mathbf{x}_i - \mathbf{x}_j||_2) \tag{4.3}$$

In the following, we use the discretizations in [1, 7]. For each particle i, $i = 1, \ldots, n$, we define the following terms. The pressure gradient term in the momentum equation is discretized as

$$\left(\frac{1}{\rho}\nabla p \cdot \mathbf{e}_\ell\right)_i \approx \frac{1}{m_i} \sum_{j \in \mathscr{N}_i} \left(V_i^2 p_i + V_j^2 p_j\right) \left.\frac{dW}{dr}\right|_{ij} \frac{\mathbf{r}_i - \mathbf{r}_j}{||\mathbf{r}_i - \mathbf{r}_j||_2} \cdot \mathbf{e}_\ell \qquad \ell = 1,2,3. \tag{4.4}$$

where $\mathbf{e}_1$ is a unit vector in the x direction, etc. The viscous term is discretized as

$$\left(\frac{\eta}{\rho}\nabla^2\mathbf{u}\right)_i \approx \frac{1}{m_i}\sum_{j\in\mathscr{N}_i}\frac{2\eta_i\eta_j}{\eta_i+\eta_j}\left(V_i^2+V_j^2\right)\left.\frac{dW}{dr}\right|_{ij}\frac{\mathbf{u}_i-\mathbf{u}_j}{||\mathbf{r}_i-\mathbf{r}_j||_2} \tag{4.5}$$

The SPH discretization of the momentum equation therefore gives three decoupled block systems for each component of the velocity. To discretize the Poisson operator, we have

$$\left(\nabla\cdot\frac{\nabla p}{\rho}\right)_i \approx \frac{2}{V_i}\sum_{j\in\mathscr{N}_i}\left(V_i^2+V_j^2\right)\left.\frac{dW}{dr}\right|_{ij}\frac{1}{||\mathbf{r}_i-\mathbf{r}_j||_2}\frac{p_i-p_j}{\rho_i+\rho_j} \tag{4.6}$$

and we discretize the divergence source term as

$$(\nabla\cdot\mathbf{u}^*)_i \approx \frac{1}{V_i}\sum_{j\in\mathscr{N}_i}\left(V_i^2+V_j^2\right)\left.\frac{dW}{dr}\right|_{ij}\left(\frac{\mathbf{u}_i^*+\mathbf{u}_j^*}{2}\right)\cdot\frac{\mathbf{r}_i-\mathbf{r}_j}{||\mathbf{r}_i-\mathbf{r}_j||_2}. \tag{4.7}$$

We also define for convenience the matrix operator

$$\mathbf{W} = \left[W_{ij}(\left\|\mathbf{x}_i-\mathbf{x}_j\right\|)\right], j=1,\ldots,n. \tag{4.8}$$

where W_{ij} is defined in (4.2). We also make convenient definitions for vectors and matrices, such as $\mathbf{m} = [m_1, m_2, \ldots, m_n]$ and $\mathbf{M} = \text{diag}(\mathbf{m})$. Further, we define

$$\mathbb{M} = \begin{bmatrix} \mathbf{M} & & \\ & \mathbf{M} & \\ & & \mathbf{M} \end{bmatrix}. \tag{4.9}$$

For the volumes we define $\mathbf{v} = [V_1, V_2, \ldots, V_n]$ and $\mathbf{V} = \text{diag}(\mathbf{V})$.

We can rewrite the projection scheme in terms of these SPH operators by replacing the summations in the above discretizations with the operators $\mathbf{G}$, $\mathbf{L}$, $\mathbf{P}$, and $\mathbf{D}^T$, respectively. We utilize these operator definitions because they preserve the symmetry or antisymmetry of each operation. The gradient is defined as

$$\frac{1}{\rho}\nabla p \approx \mathbb{M}^{-1}\mathbf{G}p. \tag{4.10}$$

with each component of the gradient represented as

$$\frac{1}{\rho}\nabla p\cdot\mathbf{e}_\ell \approx \mathbf{M}^{-1}\mathbf{G}_\ell p \qquad \ell = 1,2,3, \tag{4.11}$$

$$\mathbf{G} = \begin{bmatrix} \mathbf{G}_1 \\ \mathbf{G}_2 \\ \mathbf{G}_3 \end{bmatrix} \tag{4.12}$$

The Laplacian, Poisson, and divergence operators are then defined as

$$\frac{\eta}{\rho}\nabla^2\mathbf{u} \approx \mathbf{M}^{-1}\mathbf{L}\mathbf{u}, \tag{4.13}$$

$$\nabla \cdot \frac{\nabla p}{\rho} \approx 2\mathbf{V}^{-1}\mathbf{P}p, \tag{4.14}$$

and

$$\nabla \cdot \mathbf{u}^* \approx \mathbf{V}^{-1}\mathbf{D}^T\mathbf{u}^*, \tag{4.15}$$

respectively.

5. Pseudo code. From the user, gather initial position, velocity, density, viscosity and body forces, and then solve successively at each time step the two linear systems

$$\left(\mathbf{M} - \frac{\delta t}{2}\mathbf{L}\right)\mathbf{u}^* = \mathbf{M}\mathbf{u}^n + \delta t\left(-\mathbf{D}p^n + \frac{1}{2}\mathbf{L}\mathbf{u}^n + \mathbf{f}^{n+1}\right), \tag{5.1}$$

$$\mathbf{P}\delta p = \frac{1}{2\delta t}\mathbf{D}^T\mathbf{u}^*. \tag{5.2}$$

Correct the particle velocity and pressure

$$p^{n+1} = p^n + \delta p, \tag{5.3}$$

$$\mathbf{u}^{n+1} = \mathbf{u}^* - \mathbb{M}^{-1}\mathbf{G}\delta p, \tag{5.4}$$

and advect the particles forward

$$\mathbf{x}^{n+1} = \mathbf{x}^n + \frac{\delta t}{2}\left(\mathbf{u}^{n+1} + \mathbf{u}^n\right). \tag{5.5}$$

At this point corrections can be made to final particle positions (anisotropy corrections or density/div-free consistency corrections) and the particle connectivity can be recalculated.

6. LAMMPS and Trilinos & Solver Manager.

LAMMPS. LAMMPS[1] is molecular dynamics code that has various functionalities for atomic, polymeric, biological, metallic, granular, and coarse-grained systems. It provides a platform to develop new particle style, i.e. ISPH. Another importance reason that we want to implement SPH in LAMMPS is that LAMMPS is designed for parallel computing with MPI. It can help with the modeling with millions of particles.

Trilinos. Trilinos [16] is a collection of software that can solve large-scale complex multiphysics problems. The functionality we are specially interested in is it preconditioned Krylov solver. The related packages are Epetra, Belos, ML, etc.

- **Epetra**: provides basics linear algebra library and is compatible with most of other modules of Trilinos
- **Belos**: provides commonly used preconditioned iterative solvers
- **ML**: provides algebraic multigrid preconditioned based on smoothed aggregation.

[1] http://lammps.sandia.gov

Algorithm 4

1: Let there be N SPH particles. User specifies initial position ($\mathbf{x}_i^0$, $i = 1\ldots,N$), velocity ($\mathbf{u}_i^0$, $i = 1\ldots,N$), density (ρ_i, $i = 1\ldots,N$), viscosity (η), and body force ($\mathbf{f}_i$, $i = 1\ldots,N$).
2: Let the initial pressure $p_i^0 = 0$, $i = 1,\ldots,N$.
3: Compute the volumes by solving $\mathbf{W}^0\mathbf{v} = \mathbf{1}$.
4: Let the masses m_i, $i = 1,\ldots,N$ satisfy $\rho_i = \sum_{j\in\mathcal{N}_i} m_i W_h(\left\|\mathbf{x}_i - \mathbf{x}_j\right\|)$, i.e., solve $\mathbf{W}^0\mathbf{m} = \rho^0$.
5: **for** $n = 1$ to $\cdots$ **do**
6: Compute $\mathbf{V}^{-1} = \mathbf{W1}$. ▷ Compute the volumes.
7: Let $\mathbf{b}_1 = \mathbf{M}\mathbf{u}_1^n + \delta t\left(-\mathbf{G}_1 p^n + \frac{1}{2}\mathbf{L}\mathbf{u}_1^n + \mathbf{f}_1\right)$
8: Let $\mathbf{b}_2 = \mathbf{M}\mathbf{u}_2^n + \delta t\left(-\mathbf{G}_2 p^n + \frac{1}{2}\mathbf{L}\mathbf{u}_2^n + \mathbf{f}_2\right)$
9: Let $\mathbf{b}_3 = \mathbf{M}\mathbf{u}_3^n + \delta t\left(-\mathbf{G}_3 p^n + \frac{1}{2}\mathbf{L}\mathbf{u}_3^n + \mathbf{f}_3\right)$
10: Solve $\left(\mathbf{M} - \frac{\delta t}{2}\mathbf{L}\right)[\mathbf{u}_1^*\ \mathbf{u}_2^*\ \mathbf{u}_3^*] = [\mathbf{b}_1\mathbf{b}_2\mathbf{b}_3]$
11: Solve $\mathbf{P}\delta\mathbf{p} = \frac{1}{2\delta t}\mathbf{D}^T[\mathbf{u}_1^*\ \mathbf{u}_2^*\ \mathbf{u}_3^*]$.
12: $\mathbf{u}_1^{n+1} = \mathbf{u}_1^* - \delta t\mathbf{M}^{-1}\mathbf{G}_1\delta\mathbf{p}$ ▷ Correct particle velocity
13: $\mathbf{u}_2^{n+1} = \mathbf{u}_2^* - \delta t\mathbf{M}^{-1}\mathbf{G}_2\delta\mathbf{p}$ ▷ Correct particle velocity
14: $\mathbf{u}_3^{n+1} = \mathbf{u}_3^* - \delta t\mathbf{M}^{-1}\mathbf{G}_3\delta\mathbf{p}$ ▷ Correct particle velocity
15: $\mathbf{p}^{n+1} = \mathbf{p}^n + \delta\mathbf{p}$ ▷ Correct particle pressure
16: $\mathbf{x}_1^{n+1} = \mathbf{x}_1^n + \frac{\delta t}{2}\left(\mathbf{u}_1^{n+1} + \mathbf{u}_1^n\right)$ ▷ Advect particles
17: $\mathbf{x}_2^{n+1} = \mathbf{x}_2^n + \frac{\delta t}{2}\left(\mathbf{u}_2^{n+1} + \mathbf{u}_2^n\right)$ ▷ Advect particles
18: $\mathbf{x}_3^{n+1} = \mathbf{x}_3^n + \frac{\delta t}{2}\left(\mathbf{u}_3^{n+1} + \mathbf{u}_3^n\right)$ ▷ Advect particles
19: **end for**

IMPLICITSPH module and Solver Manager. The USER-SPH package [4] in LAMMPS is an explicit version of SPH implemented by Georg C. Ganzenmülcer and Martin O. Steinhauser. The discretization related to NS equations is based on the work of Monaghan [14] and Morris et al [15]

Since we are using different SPH operator and implicit scheme, we have built another package IMPLICITSPH and the discretization we used is based on the work of Hu and Adams [6, 7] .

Our main contribution includes

- **pair style**: new pair style of Hu-Adams, i.e. discretization of differential operators. Implemented in **Force** → **Pair**.
- **time integrator**: Crank-Nicolson time integrator based on projection method. Implemented in **Modify** → **Fix**.
- **solver manager**: interface between LAMMPS and Trilinos in order to let Trilinos solve the linear systems. Implemented in **Force** → **Pair**.

In the following outline, the corresponding steps in Algorithm 4 are shown.

Outline of IMPLICITSPH

1. Initial intergrate (Step 6)
   (a) Compute volume
2. Pair compute (Step 7–15)
   (a) build & solve Helmholtz equation
   (b) build & solve Poisson equation
   (c) correct pressure
   (d) correct particle velocity
3. Final integrate (Step 12–18)

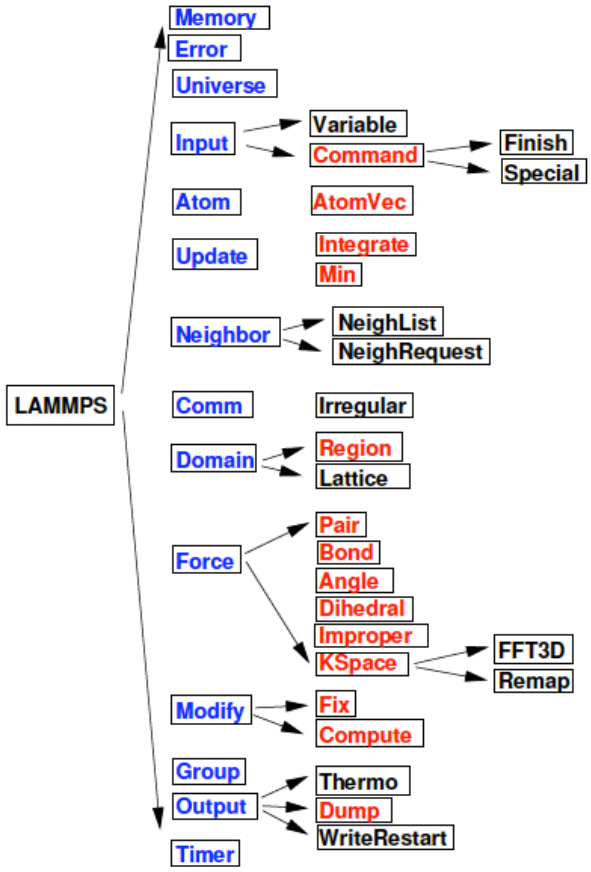


FIG. 6.1. *Class hierarchy of LAMMPS (http://lammps.sandia.gov)*

(a) update particle position

Remark: In the steps of solving Helmholtz equation and Poisson equation, we assemble sparse linear system with the solver manager and use Trilinos to obtain the solution. The solver we are using now is block-CG (Belos) preconditioned by smoothed-aggregation AMG (ML).

7. Numerical Results: Taylor-Green vortex. For the NS equations on the 2-D square domain $[0,2]\times[0,2]$ with periodic boundary condition, the analytic solution is given by

$$
\begin{aligned}
u_x &= U_0 e^{-2\nu\pi^2 t}\sin(\pi x)\cos(\pi y)\\
u_y &= -U_0 e^{-2\nu\pi^2 t}\cos(\pi x)\sin(\pi y)\\
p &= \frac{U_0^2}{4}e^{-4\nu\pi^2 t}(\cos(2\pi x)+\cos(2\pi y)+2).
\end{aligned}
$$

This is well known as Taylor-Green vortex problem. We use the corresponding initial value and start

the simulation. The velocity fields at $t_0 = 0$, $t_1 = 0.1875$, $t_2 = 0.375$ and $t_3 = 0.5625$ are provided in Figure 7.1.

From the pattern of the velocity field, we can see that the numerical solution resembles the analytic solution. And the numerical solution shows the decaying of the velocity magnitude over several time steps. Quantitative convergence tests are under investigation.

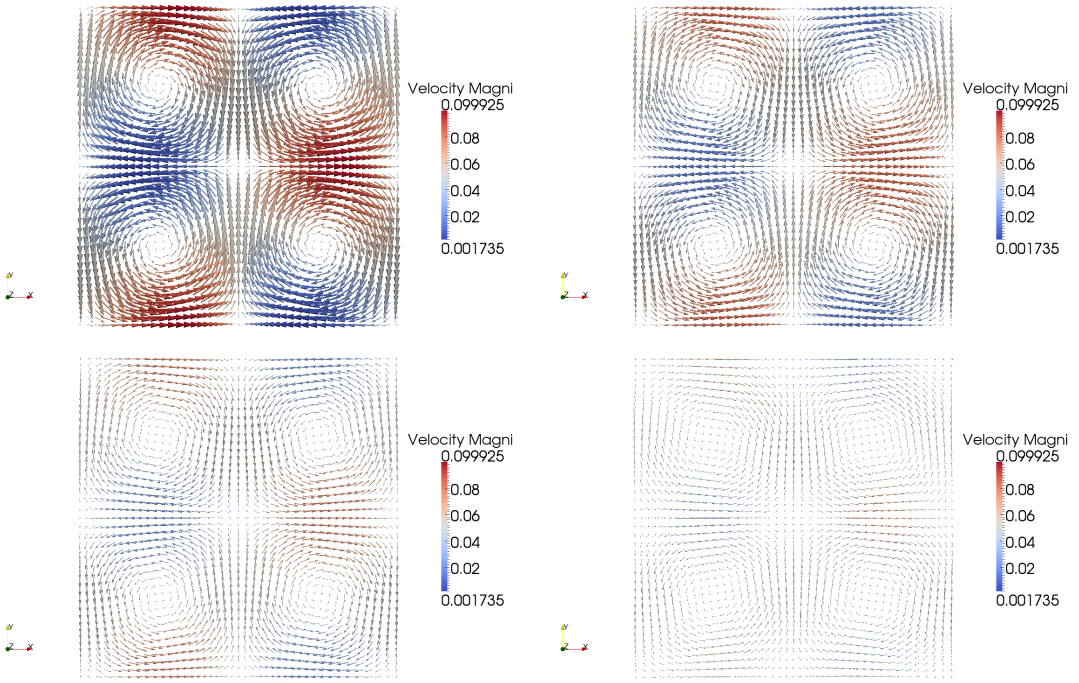


FIG. 7.1. *Velocity fields for* 0*s (top left),* 0.1875*s (top right),* 0.375*s (bottom left) and* 0.5625*s (bottom right)*

8. Conclusions. In this proceeding paper, we present the SPH discretization of the NS equations and implementation of ISPH in LAMMPS, where Solver Manager interface is built to employ the linear solvers in Trilinos. We test the code with the Taylor-Green vortex problem. The velocity field of numerical solutions shows the flow pattern and decaying of velocity magnitude over several time steps, which is consistent with analytic solutions.

Future work includes quantitative convergence tests of the IMPLICITSPH package and investigations of fast linear solvers for the Helmholtz and pressure Poisson equations.

REFERENCES

[1] S. ADAMI, X. HU, AND N. ADAMS, *A conservative SPH method for surfactant dynamics*, Journal of Computational Physics, 229 (2010), pp. 1909–1926.

[2] A. J. CHORIN, *A numerical method for solving incompressible viscous flow problems*, Journal of computational physics, 2 (1967), pp. 12–26.

[3] S. J. CUMMINS AND M. RUDMAN, *An SPH projection method*, Journal of computational physics, 152 (1999), pp. 584–607.

[4] G. C. GANZENMÜLLER, M. O. STEINHAUSER, P. VAN LIEDEKERKE, AND K. U. LEUVEN, *The implementation of smooth particle hydrodynamics in lammps.*, (2011).
[5] R. A. GINGOLD AND J. J. MONAGHAN, *Smoothed particle hydrodynamics-theory and application to non-spherical stars*, Monthly notices of the royal astronomical society, 181 (1977), pp. 375–389.
[6] X. HU AND N. ADAMS, *A multi-phase SPH method for macroscopic and mesoscopic flows*, Journal of Computational Physics, 213 (2006), pp. 844–861.
[7] ———, *An incompressible multi-phase SPH method*, Journal of Computational Physics, 227 (2007), pp. 264–278.
[8] G. E. KARNIADAKIS AND S. J. SHERWIN, *Spectral/hp element methods for CFD*, Oxford University Press, 1999.
[9] M. LASTIWKA, M. BASA, AND N. J. QUINLAN, *Permeable and non-reflecting boundary conditions in sph*, International journal for numerical methods in fluids, 61 (2009), pp. 709–724.
[10] E.-S. LEE, C. MOULINEC, R. XU, D. VIOLEAU, D. LAURENCE, AND P. STANSBY, *Comparisons of weakly compressible and truly incompressible algorithms for the sph mesh free particle method*, Journal of Computational Physics, 227 (2008), pp. 8417–8436.
[11] L. B. LUCY, *A numerical approach to the testing of the fission hypothesis*, The astronomical journal, 82 (1977), pp. 1013–1024.
[12] J. J. MONAGHAN, *Smoothed particle hydrodynamics*, Annual review of astronomy and astrophysics, 30 (1992), pp. 543–574.
[13] J. J. MONAGHAN, *Simulating free surface flows with SPH*, Journal of computational physics, 110 (1994), pp. 399–406.
[14] J. J. MONAGHAN, *Smoothed particle hydrodynamics*, Reports on progress in physics, 68 (2005), p. 1703.
[15] J. P. MORRIS, P. J. FOX, AND Y. ZHU, *Modeling low Reynolds number incompressible flows using SPH*, Journal of computational physics, 136 (1997), pp. 214–226.
[16] M. SALA, M. A. HEROUX, AND D. M. DAY, *Trilinos Tutorial*, Tech. Rep. SAND2004-2189, Sandia National Laboratories, 2004.
[17] R. XU, P. STANSBY, AND D. LAURENCE, *Accuracy and stability in incompressible sph (isph) based on the projection method and a new approach*, Journal of Computational Physics, 228 (2009), pp. 6703–6725.

Combinatorial Algorithms and Visualization

Articles in this section focus on development of combinatorial algorithms, their implementation in multicore or manycore architectures, and visualization techniques. Their applications vary from usage in a single graphic processing unit to a massively parallel supoercomputer.

Deveci et al. demonstrate a task mapping algorithm that partitions the geometric coordinates of nodes of a supercomputer and the coordinates of the application in order to effectively do task mapping. They demonstrate their result on a mini-applications on up to 6K processors. *Miller and Moreland* explore the use of key-value reduction on disconneected geometric elements, implement it for different visualization operations and demonstrate it is efficient in modern architectures. *Tong and Moreland* introduce a clipping technique for parallel triangle clipping on the GPU. They apply the technique on unstructured data and demonstrate up to 10x speedup over traditional methods. *Ye and Moreland* compare different search structures in graphic processing units with respect to build time, query time and memory usage. They show that the selection of the search structures is problem dependent. *Slota and Rajamanickam* present a multithreaded algorithm to find the strongly connected components of a graph. They introduce a new method called multistep which results in over 60x speedup over serial Tarjan's algorithm. They present their results on various graphs from social network analysis and compare their algorithm to state-of-the-art algorithms.

S. Rajamanickam
M.L. Parks
S.S. Collis

July 22, 2014

TOPOLOGY AWARE TASK MAPPING USING GEOMETRIC PARTITIONING

MEHMET DEVECI*, SIVASANKARAN RAJAMANICKAM[1], KAREN D. DEVINE[1], VITUS J. LEUNG[1], KEVIN T. PEDRETTI[1], AND ÜMIT V. ÇATALYÜREK*

Abstract. As supercomputers are reaching toward petascale, the number of nodes in supercomputers is increasing, along with the number of processors in each node. This growth causes more sparse network allocations when working with thousands of processors. As the sparsity of the allocated nodes may harm the scalability of the application, network-topology-aware task mapping becomes important to reduce communication costs and congestion for parallel applications. In this paper, we try to reduce communication overhead with topology mapping that uses a geometric partitioning algorithm. We show an application whose scalability can be improved with a proper task mapping, and by using our speculative method we reduce the communication time up to 40%.

1. Introduction. The number of nodes in supercomputers is increasing rapidly. For example, Hopper, a Cray XE6 machine at the National Energy Research Scientific Center (NERSC), has 6528 nodes with a 3D torus network topology; each node consists of 24 cores. Moreover, as supercomputers have increasingly large diameters and many users, processor allocations (the sets of processors assigned by a job scheduler to parallel jobs) become more sparse. These factors cause messages to travel long routes in the network, which makes maintaining communication scalability in large scale machines difficult. In addition, if the underlying tasks of a parallel job are not mapped properly to the processors, messages might travel very long distances, or communication links might be congested by heavy traffic.

For a good mapping of the tasks to the processors, one needs to consider the underlying tasks with their communication pattern and the physical processor topologies. This is called topology mapping. Usually, the tasks' communication patterns and processor topologies are represented as graphs.

There have been several methods that reduce communication overhead with better placement of tasks on processors. For example, Bokhari [6] reduces the problem of task mapping to graph isomorphism and proposes a heuristic to reduce the communication latency on array processors. Lee and Aggarwal [11] proposes an automated two-phase mapping strategy, which is extended by Bollinger and Midkiff [7]. Moreover, a task mapping method that uses recursive bisection of the graphs is included in SCOTCH package [13]. Different topology-aware mapping methods on BlueGene/L are studied by Yu et. al. [15]. Bhatelé et. al [5] proposes an algorithm for load balancing and topology-aware mapping of tasks for CHARM++. Hoefler and Snir [10] prove that the task mapping problem is NP-Complete. They study several graph-based mapping methods (e.g., greedy, recursive bisection) and propose a method based on Reverse Cuthill McKee (RCM) ordering. Aktulga et. al [1] reduce an eigenvalue solver application's communication using topology-aware task mapping methods.

Rather than address the expensive graph-based mapping problem directly, we propose in this paper a topology-aware mapping algorithm that uses an inexpensive geometric partitioner. Communication between tasks and network topology are approximated by the relative positions of tasks and processors on the coordinate axes. The algorithm uses a geometric partitioning algorithm to reorder the tasks and processors, and performs the mapping operation using the result of this reordering.

The rest of the paper is as follows. We give some background information in section 2. Then the proposed topology-aware mapping method is explained in section 3. We present experiments and results in section 4, and conclusions in section 5.

2. Background. We propose two metrics to represent communication in a network. However, because communication is a real time process, it is affected by many outside factors. Therefore, these theoretical

*The Ohio State University, {mdeveci,umit}@bmi.osu.edu
[1]Sandia National Laboratories, {srajama,kddevin,vjleung,ktpedre}@sandia.gov

metrics can only approximate actual communication time. The first metric is called *dilation*. It is related to the average length of paths taken by messages. The other metric is called *maximum congestion*, which represents the maximum number of messages across communication links. In this paper, we assume static routing of messages. Also, we assume that each message is transferred with a single path (the messages are not split and sent through multiple paths), and all the links have the same capacity. The messages are routed in the x direction first, then in the y and z directions, accordingly.

Let $G_t(V_t, E_t)$ be the graph representing the task communication, where V_t is the set of tasks, and E_t is the set of edges that represents the communication pattern of the tasks. If $t_1, t_2 \in V_t$, then edge $(t_1, t_2) \in E_t$ if and only if tasks t_1 and t_2 require communication between them. In the same way, let $G_n(V_n, E_n)$ be the graph representing the nodes together with the communication links between them. V_n is the set of nodes, and E_n is the set edges that represents the physical communication links between the nodes. If $n_1, n_2 \in V_n$, then edge $(n_1, n_2) \in E_n$ if and only if nodes n_1 and n_2 have a connecting link between them. Let Γ be a function for the assignment of the tasks to the nodes. That is, $n_1 = \Gamma(t_1)$ if t_1 is assigned to a processor in node n_1. Using these assumptions, we define the dilation as follows:

$$dilation(t_1, t_2) = SPL(\Gamma(t_1), \Gamma(t_2), G_n), \tag{2.1}$$

where *SPL* is a function that returns the shortest path length between two nodes. Then, the total dilation can be defined as following:

$$Dilation(\Gamma) = \sum_{(t_1,t_2)\in E_t} dilation(t_1, t_2) \tag{2.2}$$

Dilation is related to the average number of edges traversed by each message, or the *average hop count*:

$$AverageHopCount(\Gamma) = Dilation(\Gamma)/|E_t| \tag{2.3}$$

In this paper, we measure the average hop count, and use the terms "hop count" and "average hop count" interchangeably.

The other metric, congestion, is defined as follows:

$$Congestion(e) = \sum_{(t_1,t_2)\in E_t} inSP(e, \Gamma(t_1), \Gamma(t_2), G_n) \tag{2.4}$$

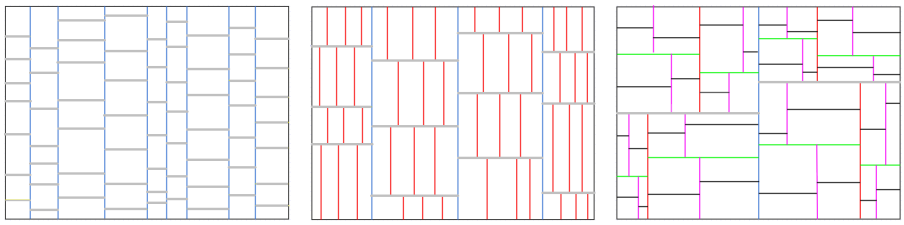
where *inSP* returns 1 if and only if link e is in the shortest path between $\Gamma(t_1)$ and $\Gamma(t_2)$. Otherwise, it returns 0. Therefore, the congestion of a link becomes the number of messages that go through it. Then the maximum congestion becomes

$$MaxCongestion(\Gamma) = \max_{e\in E_n} Congestion(e) \tag{2.5}$$

Maximum congestion represents the maximum number of messages that go through a link. We refer to maximum congestion as "congestion" in the rest of this paper.

3. Methods. Our proposed topology-aware mapping algorithm represents the topology of a parallel computer with the coordinates of processors (where proximity of coordinates approximates bandwidth between processors), rather than with a topology graph (in which bandwidths between processors are explicitly given as edge weights). The algorithm uses a geometric partitioning algorithm. Thus, it requires both coordinates for tasks (e.g., task centers) and coordinates for processors. The partitioning algorithm is used to reorder the tasks and processors; the resulting ordering is used to determine a mapping.

3.1. Multi-Jagged (MJ) Algorithm for Geometric Partitioning. Our task mapping algorithm uses a geometric partitioner, the Multi-dimensional Jagged algorithm (MJ) [14] of the Zoltan2 Toolkit [8], to partition the task and machine coordinates. MJ partitions a given set of coordinates into a desired number of parts (P) in a given number of steps which is called *recursion depth* (RD). During each recursion, one-dimensional partitioning is done along a dimension, and this dimension is alternated at each recursion. Therefore, MJ is a generalization of Recursive Coordinate Bisection (RCB) [4] in which MJ can do multisections instead of bisections. Figure 3.1 shows possible 64-way partitioning of MJ with $RD = 2, 3$ and 6.



(a) $RD = 2$ (b) $RD = 3$ (c) $RD = 6$

FIG. 3.1. *The partitioning of the dataset into 64 parts using MJ with different recursion depths RD. Cutlines that are in the same level of recursion share the same color.*

Given a dataset to partition into P parts in RD steps, MJ partitions the data into $p_i = \lceil P_i^{\frac{1}{RD-i}} \rceil$ parts during the ith step, where $P_0 = P$. For each part obtained during the ith level, the future number of parts P_{i+1} is calculated, and the partitioning algorithm is repeated on each part. Simply, $P_{i+1} = \lceil P_i/p_i \rceil$ for part j if $j < mod(P_i, P_j)$; otherwise $P_{i+1} = \lfloor P_i/p_i \rfloor$. For example, in figure 3.1(b), during the first step, data is partitioned into $P_0 = 4$ parts. Then, for all four obtained parts, the future number of parts is $P_{i+1} = 16$, and the partitioning is repeated on these parts.

3.2. Using MJ for Task Mapping. Although MJ is proposed as a parallel (MPI+OpenMP) algorithm, it is used as a sequential algorithm in this context. Since current supercomputers have $O(100K)$ processors, partitioning in parallel such a small dataset is communication bounded; therefore, little or no speedup can be obtained by parallelizing this process.

The proposed mapping algorithm is defined as follows: Given $tdim$-dimensional coordinates of tasks (tc), and $pdim$-dimensional coordinates of processors (pc), together with the number of tasks (tn) and processors (pn), the algorithm returns a mapping from processors to tasks ($p2t$) (and/or tasks to processors $t2p$). Algorithm 5 gives the description of the task mapping algorithm.

Although MJ is a partitioning algorithm, its main purpose in this algorithm is to consistently number the processors and tasks. MJ partitions the tasks and processors. assigning a part number to each processor and task. The processors and tasks that are assigned the same part number are then mapped to each other in GETMAPPINGARRAYS; the results of the mapping are stored in $p2t$ and $t2p$. Since tasks and processors are partitioned separately, the algorithm ensures that part number assignments are consistent between each MJ call. In order to be consistent, first the minimum coordinate dimension is chosen between the tasks and processors. For example, if $pdim = 3$ while $tdim = 2$, one of the processors' coordinates is ignored to ensure that the geometric partitioner follows the same order in both partitioning operations. Moreover, the algorithm can follow different paths depending on the number of tasks and processors. There are three possibilities:

1. $tn = pn$**:** When the numbers of processors and tasks are equal, there is a one-to-one mapping between processors and tasks. If task t is assigned to processor p, $t = p2t[p]$, and $p = t2p[t]$.

Algorithm 5 Task Mapping Algorithm using MJ

Require: $tc, tdim, tn, pc, pdim, pn$
  $minDim \leftarrow \min(tdim,\ pdim)$
  $usedNumProcs \leftarrow numParts \leftarrow \min(tn,\ pn)$
  if $pn > tn$ **then**
    $procPerm \leftarrow$GETCLOSESTSUBSET$(pc,\ pdim,\ pn,\ tn)$
  else
    $procPerm \leftarrow range(0,\ pn)$
  end if
  $taskPerm \leftarrow range(0,\ tn)$
  $taskParts \leftarrow$MJ$(tc,\ minDim,\ tn,\ taskPerm,\ numParts)$
  $procParts \leftarrow$MJ$(pc,\ minDim,\ usedNumProcs,\ procPerm,\ numParts)$
  $(p2t, t2p) \leftarrow$GETMAPPINGARRAYS$(taskParts,\ procParts,\ taskPerm,\ procPerm, tn,\ pn)$

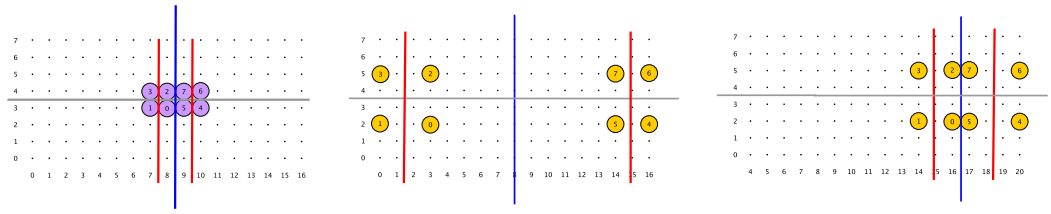
2. $tn > pn$**:** If there are more tasks than processors, a processor is assigned multiple tasks. Both processors and tasks are partitioned into pn parts, with multiple tasks per part. The mapping results will be $t \in p2t[p]$ and $p = t2p[t]$.
3. $tn < pn$**:** When there are more processors than tasks, the algorithm does not split a task among multiple processors. Instead, during a preprocessing step, it chooses a subset of processors that has size tn. Then mapping is performed within this subset as if $tn = pn$. Processors not in the subset will be idle, as they are not assigned any tasks.

3.3. Choosing a subset of processors. The algorithm assumes atomicity of tasks. That is, it does not allow multiple processors to share a task. When there are more processors than tasks, the algorithm chooses a subset of the processors during a preprocessing step.

The quality of the mapping can depend on this subset selection. Since there are $\binom{pn}{tn}$ subsets, an exhaustive search of all subsets has a cost that increases exponentially with pn. Instead, our mapping algorithm adapts a greedy heuristic that tries to find a subset containing processors that are close to each other, so that fewer hops are required for communication among the processors. Our greedy heuristic is an adapted version of K-means clustering [9]. In our modified algorithm, each cluster is limited to have only a specified number of points, tn in our case. Moreover, the clusters are allowed to overlap; that is, a coordinate can be assigned to more than one cluster. Each cluster maintains a maximum heap with size tn, and only the points with the tn smallest distances to the cluster center are kept in the cluster. When a point is processed, its distance to all of the clusters is calculated. If the distance of the point is smaller than the maximum of the heap, the point is inserted the cluster, and the point with maximum distance is discarded. This process is repeated with a limit on the maximum number of iterations (10), or until the center of the clusters does not move. At the end of the clustering algorithm, the distances of each coordinate are summed and the cluster having the closest points is chosen as the subset.

The complexity of an iteration of the original K-means algorithm is $O(pn \times k)$ for k clusters. The modified version has a complexity of $O(pn \times k \times \log(tn))$. In order to relax the complexity, we set $k = 2^{minDim} + 1$. Therefore, there are nine clusters for three-dimensional datasets. Eight of the clusters are initialized with the eight corners of the dataset, while one is initialized to the center.

3.4. Improving the quality of the mapping. As the mapping algorithm uses the partitioning results of MJ, the quality of the mapping depends highly on the partitioning results of MJ. There are various operations that can effect the partitioning and the mapping quality which will be described in this section.



(a) Task coordinates (b) Processor coordinates (c) Shifted processor coordinates

FIG. 3.2. *An example showing the benefit of shifting the processor coordinates in torus networks. The processors and tasks having the same part number are mapped to each other. Figure 3.2(c) shows the partitioning of processors after shifting around the wrap-around link. The mapping at figure 3.2(b) has an average hop count of 3.66 and 3 on x and y directions, respectively. The mapping obtained after shifting has an average hop count of 2 and 3 in x and y.*

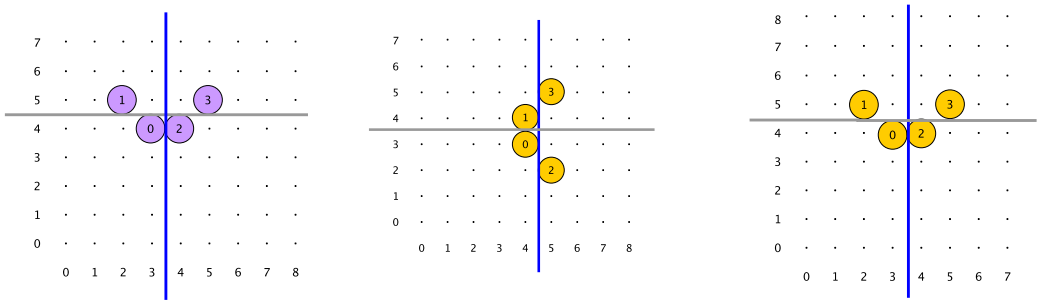
3.4.1. Shifting the processor coordinates. Many networks in current supercomputers have 3D torus interconnection between the nodes. For example, the 6528 nodes of NERSC's Cray XE6, Hopper, are laid out in a 3D torus with dimensions 17x8x24 (with two nodes in each coordinate). The x coordinates of the nodes range from 0 to 16, and there are wrap-around links between nodes 0 and 16 in the x dimension. Being a geometric partitioner, MJ ignores this connectivity information. Figure 3.2 shows an example of mapping eight tasks onto eight processors that have a 2D torus topology with dimensions 17x8 with wrap-around links. MJ is used to partition (reorder) the tasks and processors, as indicated by the blue, grey and red lines at recursion levels 0, 1, and 2, respectively. Processors and tasks that are assigned the same part number are mapped to each other. Assume that each task in 3.2(a) communicates only with its neighbor tasks. According to this communication pattern, while the mapping as a result of figure 3.2(b) obtains average hop counts of 3.66 and 3 in the x and y dimensions, respectively, the mapping in figure 3.2(c) achieves average hop counts of 2 and 3. Therefore, mapping quality can be improved by shifting around wrap-around links.

MJ adopts a greedy heuristic for detecting the shift position. The shifting of the coordinates is performed for each dimension independently. The algorithm finds the biggest gap between any two consecutive processors along a dimension and treats this gap as if it were the link at wrap-around. For example, in figure 3.2(b) the biggest gap along the x dimension is found between the processors on coordinates 3 and 14. Therefore, the heuristic shifts the coordinates of all processors that have $x \leq 3$. If there is a tie on the gap distance, the tie is broken by using the number of processors along these dimensions. In this case, the one with the larger number of processors on the gap coordinates is chosen as the shift position.

Even though the list of processors is not sorted for any dimensions, this operation can still be performed in O(pn) time by using counting sort algorithm. This is because the coordinates of the processors are integers, and the minimum and the maximum coordinates are known and are expected to be reasonably small numbers.

3.4.2. Rotating the processor and task coordinates. The quality of the mapping also depends on the order of the dimensions according to which the partitioning is performed. For example, Figure 3.3 shows how the quality of the mapping can change by choosing a different order of dimensions in partitioning.

It is difficult to predict which dimension ordering for partitioning will provide the best mapping quality. One could choose a permutation of the dimensions based on the aspect ratios of the processor and task coordinates. The permutation that makes the aspect ratios along dimensions closest can be chosen as the best permutation. However, as our experiments will show, this greedy method fails to find the best permutation in most of the mappings. To overcome this issue, we use a speculative method. Since the proposed mapping algorithm is sequential, each of the processors calculates the same mapping independently. A reduceAll



(a) Task coordinates (b) Processor coordinates (c) Rotated processor coordinates

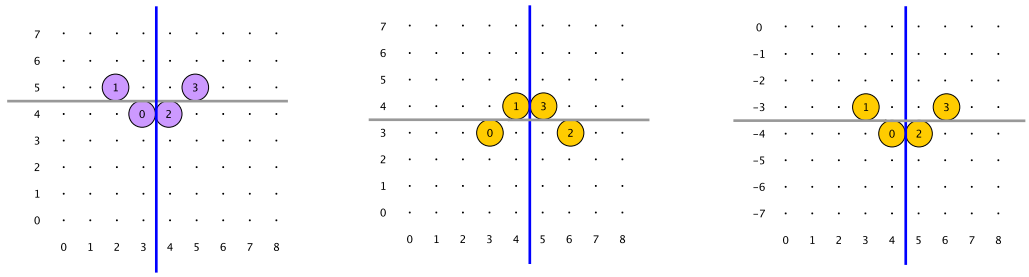
FIG. 3.3. *An example showing the benefit of rotating the processor orientation. Figure 3.3(c) shows the partitioning of processors after rotation; the partitioning is performed on the y dimension first, and then on the x dimension. Assuming that communication is required between only tasks 0 and 2, the mapping at figure 3.3(b) has an average hop count of 1 and 1 in the x and y directions, respectively. The mapping obtained after axis rotation has an average hop count of 1 and 0 in x and y.*

operation is performed at the beginning of task mapping operation in order to obtain all processor and task coordinates. Then each of the processors performs the sequential mapping operation and obtains the exact same mapping. However, as there are pn processors, it is possible to calculate different mappings on each processor. Then, if a task communication graph that contains the communication pattern of the tasks is provided, the quality of the mapping (in terms of hop count) can be evaluated. The algorithm can compare these different mappings, and choose the one that has the best quality. This can be one with one extra reduceAll and broadcast operation. If the dimensions of the tasks and the processors are $tdim$ and $pdim$, we can have $(tdim)! \times (pdim)! = rp$ different rotations. For a 3D torus with 3D task coordinates, there are $3! \times 3! = 36$ different rotations. The processors are grouped into sets that have size 36, in which each processor calculates a mapping using a different rotation. Using the task communication graph, each processor calculates the quality of its own mapping. Then within each group, the processor having the best quality mapping is determined, and this processor broadcasts its result to the other processors in the group. When the number of processors is not divisible by rp, the remainder processors are distributed to other groups to allow as many rotations as possible to be calculated within each group.

3.4.3. Reflection of processor or task coordinates on a coordinate axis. As with the rotation of the coordinates, reflection of the processor coordinates along a dimension may increase the quality of the mapping. Figure 3.4 shows an example how the reflection operation may improve the mappings.

Again, it is difficult to predict whether a reflection operation improves or degrades the quality of the mapping. However, there is a limited number of possibilities for this operation. A task having $tdim$ dimensions can be reflected in 2^{tdim} different ways; therefore, there are only eight different ways of reflecting 3D task coordinates. In the mapping algorithm, the operation is performed on either processor or task coordinates, whichever has the higher dimension. Therefore, the total number of different operations becomes 2^{maxDim}, where $maxDim = max(tdim, pdim)$. The processors can be grouped where each group has 2^{maxDim} processors that each calculate a different mapping. Combined with the rotation operation, the total number of different solutions becomes $2^{maxDim} \times tdim! \times pdim!$, which is 288 for the usual case of a 3D torus with three-dimensional tasks.

4. Experiments. The proposed mapping techniques are implemented in the Zoltan2 library [8]. Our experiments are run in MiniGhost [3], a finite-difference proxy application that implements a finite difference stencil across a three-dimensional uniform grid. Each task communicates with two neighbors along each

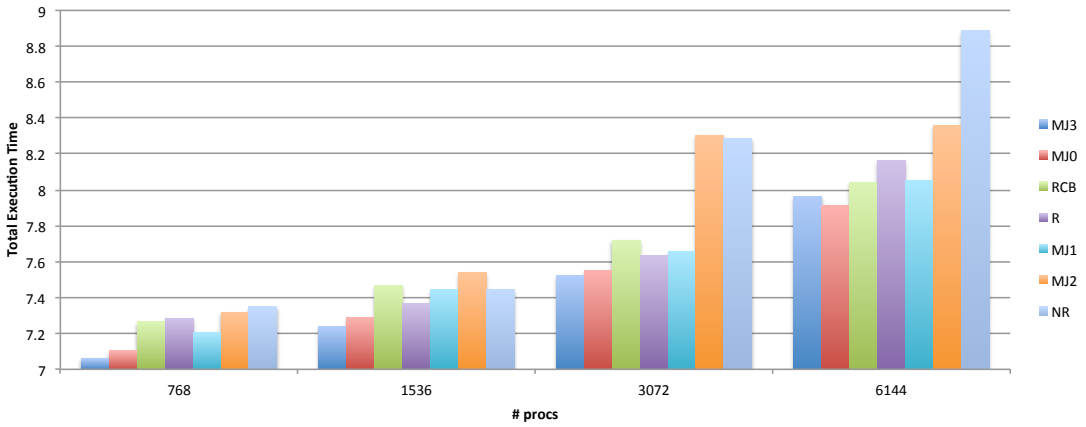


(a) Task coordinates (b) Processor coordinates (c) Reflected processor coordinates

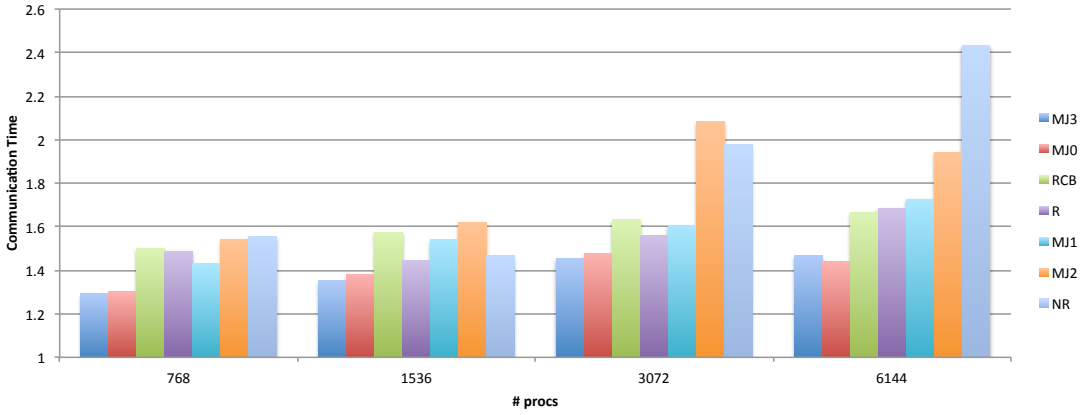
FIG. 3.4. *An example showing the benefit of reflecting the processor coordinates. Figure 3.4(c) shows the partitioning of processors after the coordinates are subjected to a reflection operation on Y axis (scale with* -1*). Assuming that communication is required between only tasks 0 and 2, the mapping at figure 3.4(b) has an average hop count of 3 and 0 in the x and y directions, respectively. The mapping obtained after reflection operation has an average hop count of 1 and 0 in x and y.*

dimension; tasks along a geometry boundary communicate only with neighbors interior to the boundary (i.e., boundary conditions are non-periodic). Each task is assigned a subgrid of the 3D grid based on its task number. The numbers of tasks in each dimension pn_x, pn_y, pn_z (with $pn_x \times pn_y \times pn_z = pn$) are specified by the user. Subgrids of the 3D grid are assigned to tasks by sweeping first in the x-direction, then the y-direction, then the z direction. Thus, task i shares subgrid boundaries (and, thus, requires communication) with tasks $i+1$ and $i-1$ to its east and west, respectively; with tasks $i+pn_x$ and $i-pn_x$ to its north and south; and with tasks $i+(pn_x)(pn_y)$ and $i-(pn_x)(pn_y)$ to its front and back. In the default MiniGhost configuration, task i is performed by rank i. As shown in [2], the run time of MiniGhost may not scale well in weak scaling tests. We claim that scalability can be improved with topology-aware mapping of the tasks onto the processors so that tasks that share boundaries are placed "near" each other in the processor allocation. Therefore, we run experiments with MiniGhost and compare the effects of different mapping strategies on the communication and total execution time. These mapping methods include

1. *NR*: The default MiniGhost mapping of tasks to ranks: task i is performed by rank i.
2. *R*: A MiniGhost option that reorders tasks into 16-task blocks, with 2x2x4 tasks per block. Ideally, a block is assigned to cores within the same node, so that frequently communicating tasks are within the same node. However, this mapping does not account for inter-node communication patterns, since it does not use information about the position of nodes in the network. (This 16-task block is not optimal for Hopper, which has 24 cores per node; results in [12] show performance of *R* relative to other methods on a 16-core/node Cray. Future modifications to MiniGhost will accommodate 24-task blocks.)
3. *RCB*: Recursive Coordinate Bisection with a single rotation determined using the aspect ratios of the task and processor coordinates [12].
4. *MJ*0: MJ with recursion depth $RD = \lceil logP \rceil$ (i.e., performing bisections at each level) and 36 different solutions calculated according to 36 different rotations; the solution with the lowest hop count metric is chosen.
5. *MJ*1: MJ with an average recursion depth $RD = \frac{\lceil logP \rceil + 3}{2}$, and 36 different rotations as in *MJ*0.
6. *MJ*2: MJ with the minimum recursion depth $RD = minDim = 3$, and 36 different rotations as in *MJ*0.



(a) Total Execution Time



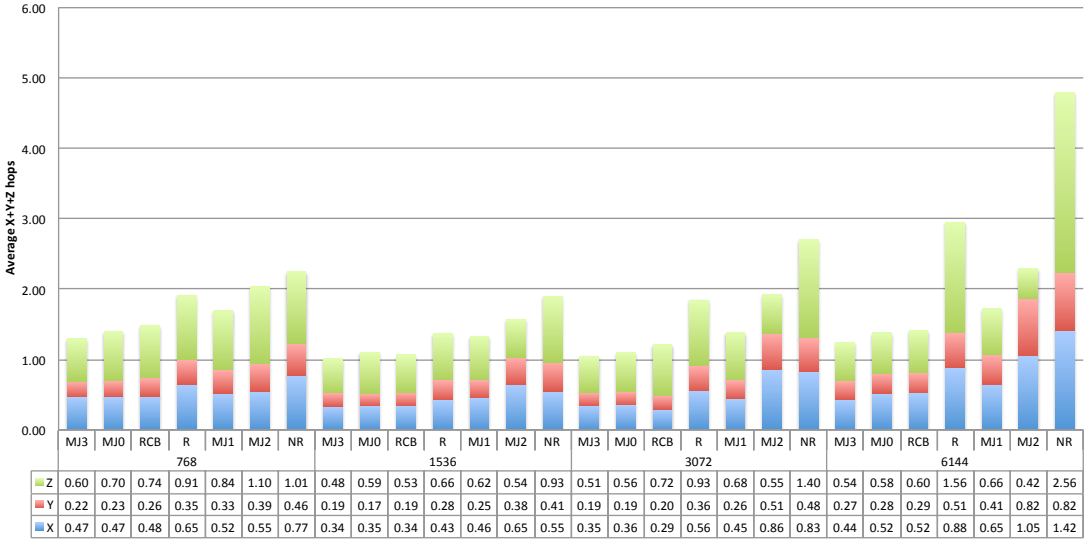
(b) Communication Time

FIG. 4.1. *Weak scaling results of MiniGhost.*

7. *MJ*3**:** MJ with recursion depth $RD = \lceil logP \rceil$, and 36 different rotations as in *MJ*0. Additionally, as a preprocessing operation, coordinates are shifted to accommodate torus networks.

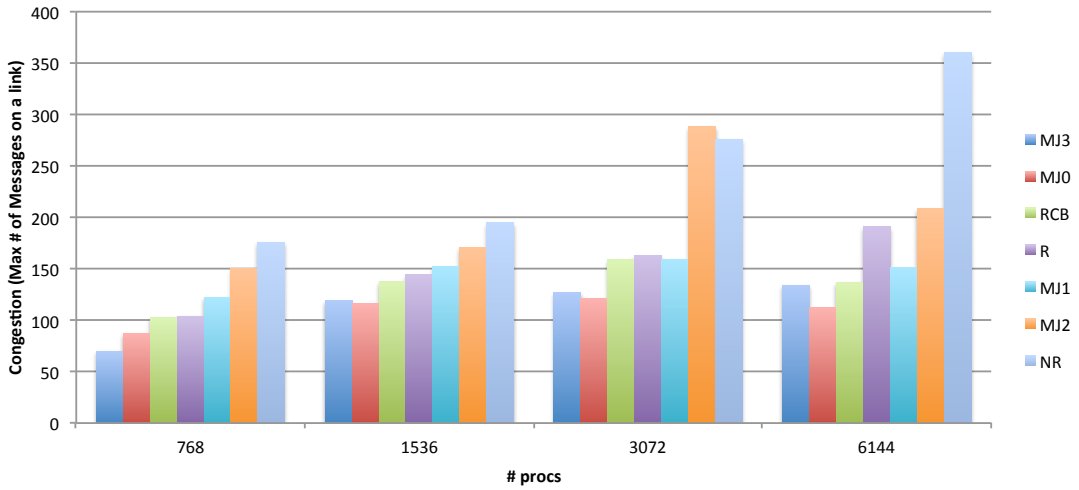
The reflection operation is not evaluated in this experiment, as MiniGhost tasks have a perfect uniform-grid structure. Therefore, reflection would not change the quality of the solutions. Also, as MiniGhost expects $tn = pn$, only this case of the algorithm is evaluated.

Figure 4.1 shows weak scaling results (total execution time and communication time) of MiniGhost with the above task reordering methods. Figure 4.2 shows the calculated quality metrics (maximum congestion and average $x+y+z$ hops). All variants are run within the same processor allocation, and the results are averaged for three different processor allocations. Only the mapping of the tasks to the processors differs among the variants; each processor's load and communication requirements stay the same. As seen in Figure 4.1, the total execution time of MiniGhost can be reduced with the use of topology-aware mappings.



	MJ3	MJ0	RCB	R	MJ1	MJ2	NR	MJ3	MJ0	RCB	R	MJ1	MJ2	NR	MJ3	MJ0	RCB	R	MJ1	MJ2	NR	MJ3	MJ0	RCB	R	MJ1	MJ2	NR
	768							1536							3072							6144						
Z	0.60	0.70	0.74	0.91	0.84	1.10	1.01	0.48	0.59	0.53	0.66	0.62	0.54	0.93	0.51	0.56	0.72	0.93	0.68	0.55	1.40	0.54	0.58	0.60	1.56	0.66	0.42	2.56
Y	0.22	0.23	0.26	0.35	0.33	0.39	0.46	0.19	0.17	0.19	0.28	0.25	0.38	0.41	0.19	0.19	0.20	0.36	0.26	0.51	0.48	0.27	0.28	0.29	0.51	0.41	0.82	0.82
X	0.47	0.47	0.48	0.65	0.52	0.55	0.77	0.34	0.35	0.34	0.43	0.46	0.65	0.55	0.35	0.36	0.29	0.56	0.45	0.86	0.83	0.44	0.52	0.52	0.88	0.65	1.05	1.42

(a) Average Hop Count along x, y and z dimensions



(b) Maximum Congestion

FIG. 4.2. *Average Hop Count and Maximum Congestion Metric results.*

For example, while *R* reduces the total execution time (communication time) from 8.89 (2.43) seconds to 8.16 (1.68) seconds, greater reduction of the total execution time and the communication time is obtained with the use of topology-aware mapping methods.

Among the MJ variants, *MJ*0 and *MJ*3 obtain the best performance. As the recursion depth *RD* of MJ increases, the quality of the mappings increases. For example, the average hop counts for *MJ*2 on 6144 processors are 1.05, 0.82, and 0.42 along x, y, and z dimensions, respectively. These numbers are 0.44,

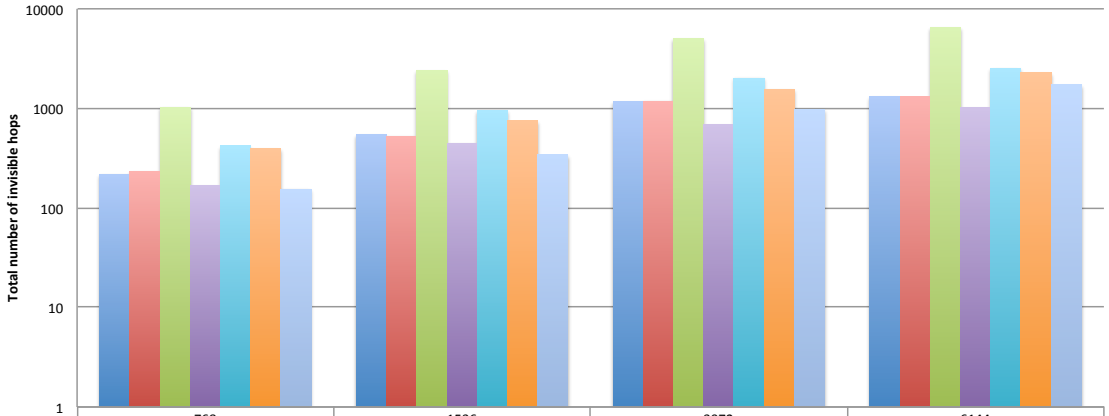
0.27, and 0.54 for *MJ*3. Also, as seen in Figure 4.2(b), *MJ*2's maximum congestion (averaged over three allocations) is 208.67, while it is 134.00 for MJ_3. As a result, the communication time of *MJ*2 is 1.94 seconds, while it is 1.46 seconds for *MJ*0. This result is reasonable, since MJ uses more information for partitioning when *RD* increases. For example, with 6144 processors, *MJ*2 (with $RD = 3$) partitions both the task and processor coordinates into 19 parts along the first dimension. This partitioning is done using information only about the x dimension of the tasks and processors; the algorithm ignores information about the y and z dimensions. Therefore, the partitioning algorithm is intensively greedy, harming the mapping quality. On the other hand, when MJ is used with $RD = \lceil logP \rceil$, MJ partitions the coordinates into two parts at each recursion level, and the information used in each dimension's partitioning increases cumulatively as the recursion depth increases. Therefore, MJ obtains the best results when it is used with bisection. Also, the proposed shifting method that is applied with *MJ*3 reduces the runtime of MiniGhost, as it reduces the hop count metric. In most instances, it reduces the communication time relative to *MJ*0 by 2%, while it reduces the average hop count $5-10\%$. The reduction in communication time is not as significant as the reduction in hop count, as the proposed shifting method increases the congestion relative to *MJ*0. Moreover, *MJ*3 is outperformed by *MJ*0 on 6144 processors. Although *MJ*3 obtains better hop counts in this instance, its maximum congestion is higher than that of *MJ*0.

RCB is outperformed by *MJ*0 and *MJ*3, as RCB uses only a single rotation based on coordinate aspect ratios of the tasks and processors. This result shows that a greedy choice of dimension ordering may not find the best possible answer that can be obtained with RCB. Overall, *MJ*0 reduced the average hop count relative to RCB by $2-9\%$ by performing speculative trials. Moreover, the method reduces the maximum congestion of RCB by $15-25\%$ in the experiments. As a result, *MJ*0 reduces the communication time relative to *RCB* by $10-14\%$

Figure 4.2 shows the quality metrics calculated for the experiments. As seen from the figures, average hop counts follow the same trend as communication times with some exceptions. Differences in the congestion metric explain cases where average hop count is not proportional to communication time. For example, on 3072 processors, although *MJ*2 obtains better average hop count than *NR*, its communication time is worse, as it has higher congestion. In most cases, a mapping algorithm with lower congestion and average hop count obtains lower communication time. However, there are some exceptions. For example on 768, 1536 and 3072 processors, *RCB* obtains better hop counts and maximum congestion than *R*, although the communication time of *R* is slightly better. Since two nodes share a single coordinate in Hopper, messages that are exchanged between cores that share the same coordinate but reside on different nodes do not affect the hop count metric. To explain the performance in these cases, we calculated the number of messages that are exchanged between processors that are in different nodes but share the same coordinate. We call this the number of "invisible hops." As seen in Figure 4.3, the RCB implementation does not distinguish between cores with the same coordinates. Therefore, it assigns many communicating tasks to processors on different nodes. For example, the *RCB* mapping requires an extra 1024 internode message exchanges on 768 processors, while *R* requires only 168. Since the cores have the same coordinates, this difference is not captured in the hop count metric. However, it still requires internode communication, which harms the performance with RCB mappings. These results show the importance of optimizing intra-Gemini communications as well as using the additional metric.

Overall, *MJ*0 and *MJ*3 reduced the communication time relative to the default MiniGhost mapping *NR* by $8-40\%$. The reductions are $7-13\%$ and $11-14\%$ with respect to *R* and *RCB*, respectively.

5. Conclusion. We have proposed a new topology-aware task mapping method that uses geometric partitioner MJ to reorder the task and processor coordinates. We show results with task mapping in the MiniGhost proxy application. MJ obtained better mappings as its recursion depth increased, since the information used in each recursion increases cumulatively. We have also proposed techniques to improve the



	768	1536	3072	6144
MJ3	217.33	542.67	1168	1322.67
MJ0	232	522.67	1184.67	1308
RCB	1024	2400.67	5055.33	6544.67
R	168	448	685.33	1010.67
MJ1	418.67	948	2017.33	2506.67
MJ2	394.67	753.33	1546.67	2275.33
NR	154.67	341.33	984	1725.33

FIG. 4.3. *The number of "invisible hops," i.e., the number of internode messages that are sent between processors sharing the same coordinate. Since these processors share one Gemini link, these messages do not affect the congestion or hop count metrics.*

quality of the mapping. For example, by speculatively trying all permutations of dimensions, MJ reduces the communication time $11-14\%$ with respect to RCB mapping. Moreover, the proposed processor coordinate shift technique reduces the communication time by another 2%. On 6144 processors, the overall reduction of communication time by $MJ0$ was 40% compared to the baseline in which no reordering of tasks is performed. As future work, we will evaluate the mapping algorithms' effectiveness on unstructured applications with any number of processors and tasks. We will also study the impact of dynamic system conditions, such as congestion and network contention, on mapping strategies.

6. Acknowledgment. We thank Erik Boman for helpful discussions.

REFERENCES

[1] H. M. Aktulga, C. Yang, E. G. Ng, P. Maris, and J. P. Vary. Topology-aware mappings for large-scale eigenvalue problems. In *Euro-Par 2012 Parallel Processing*, pages 830–842. Springer, 2012.

[2] R. Barrett, P. Crozier, D. Doerfler, S. Hammond, M. Heroux, H. Thornquist, T. Trucano, and C. Vaughan. Summary of work for ASC L2 milestone 4465: Characterize the role of the mini-application in predicting key performance characteristics of real applications. Technical Report SAND2012-4667, Sandia National Laboratories, 2012.

[3] R. F. Barrett, C. T. Vaughan, and M. A. Heroux. Minighost: a miniapp for exploring boundary exchange strategies using stencil computations in scientific parallel computing. Technical Report SAND2012-10431, Sandia National Laboratories, Albuquerque, NM, 2012.

[4] M. Berger and S. Bokhari. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Trans. Computers*, C36(5):570–580, 1987.

[5] A. Bhatelé, L. V. Kalé, and S. Kumar. Dynamic topology aware load balancing algorithms for molecular dynamics applications. In *Proceedings of the 23rd international conference on Supercomputing*, pages 110–116. ACM, 2009.

[6] S. H. Bokhari. On the mapping problem. *Computers, IEEE Transactions on*, 100(3):207–214, 1981.

[7] S. W. Bollinger and S. F. Midkiff. Heuristic technique for processor and link assignment in multicomputers. *Computers, IEEE Transactions on*, 40(3):325–333, 1991.

[8] E. G. Boman, K. D. Devine, V. J. Leung, S. Rajamanickam, L. A. Riesen, M. Deveci, and U. Catalyurek. Zoltan2: Next-generation combinatorial toolkit. Technical report, Sandia National Laboratories, 2012.

[9] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[10] T. Hoefler and M. Snir. Generic topology mapping strategies for large-scale parallel architectures. In *Proceedings of the international conference on Supercomputing*, pages 75–84. ACM, 2011.

[11] S.-Y. Lee and J. Aggarwal. A mapping strategy for parallel processing. *Computers, IEEE Transactions on*, 100(4):433–442, 1987.

[12] V. Leung, D. Bunde, J. Ebbers, N. Price, M. Swank, S. Feer, and Z. Rhodes. Task mapping for non-contiguous allocations. Technical Report SAND2013-0962, Sandia National Laboratories, Albuquerque, NM, February 2013.

[13] F. Pellegrini and J. Roman. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *High-Performance Computing and Networking*, pages 493–498. Springer, 1996.

[14] S. Rajamanickam, K. D. Devine, M. Deveci, and U. V. Catalyurek. Multi-jagged: A scalable multi-section based spatial partitioning algorithm. Technical report, Sandia National Laboratories, 2012.

[15] H. Yu, I. Chung, J. Moreira, et al. Topology mapping for blue gene/l supercomputer. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 116. ACM, 2006.

APPLICATIONS OF KEY-VALUE REDUCTION IN VISUALIZATION

ROBERT MILLER* AND KENNETH MORELAND[1]

Abstract. Graphics and visualization pipelines often make use of highly parallelized algorithms which transform an input mesh into an output mesh. One example is Marching Cubes, which transforms a voxel grid into a triangle mesh approximation of an isosurface. These techniques often discard the topological connectivity of the output mesh, and instead produce a 'soup' of disconnected geometric elements. We explore the use of an algorithm called Key-Value reduction for a variety of computations on these output elements.

1. Introduction. Many existing parallel graphics techniques are designed to generate geometry from some form of input mesh. Such algorithms tend to have the attribute that computation of any given output feature depends solely on a small set of input features. This provides a natural partitioning of the input which can then be used to develop parallel algorithms. As an example, the Marching Cubes [9] algorithm can be parallelized per voxel, so that only the data values at the voxel corners are necessary to generate the geometry within the voxel (if any).

The problem with this naive parallelization is that it generates a "soup" of output primitives, with no information about connectivity. The aforementioned Marching Cubes algorithm simply produces a set of disconnected triangles. This is not a concern so long as all further computation on the output depends exclusively on the local attributes of individual primitives and storage cost is not a primary concern, such as is the case with rendering.

Rendering has often been the ultimate goal for the generated output of parallel graphics algorithms and the size of the output is not a concern, so these techniques have been sufficient. When additional processing of the output is desired that requires information about the connectivity between multiple primitives, the situation becomes more difficult. Consider the calculation of the curvature of the isosurface generated by Marching Cubes. Given only a triangle soup, there is no simple method to determine the neighbors of any given triangle, which is a required step to compute the curvature. A more complex example would be the case of a visualization pipeline, where different types of topological connectivity information may be necessary depending on the structure of the pipeline.

This is not a new problem, and techniques exist to determine topological information about a primitive soup [10]. Generally this approach starts by finding and coalescing duplicate vertices, which may require a bounded-radius nearest-neighbor search to resolve vertices that are identical save for floating point error. Next, the algorithm finds all primitives that share two or more vertices, and links them together as neighbors. Finally, some "duplicate" vertices may not actually represent desirable connections, such as is the case where two cones meet at their apices, so these vertices may need to be split in a final pass. This process is computationally complex, and is specific to the topological connections between triangles.

We present a technique which applies equally well to other types of topological connections, such as determining the neighboring facets of tetrahedra. Our technique makes use of the fact that in visualization geometry is rarely generated in a vacuum; we usually have some information about how the geometry is generated. To list a couple of examples demonstrated in Figure 1.1: Marching Cubes generates its output vertices on the edges of a structured grid, and tetrahedralizations can be generated on an existing spatial grid or mesh.

2. Related Work. We use Marching Cubes as an example of parallelized geometry generation [9]. There are myriad existing isosurfacing implementations on the GPU, dating from Rottger's implementation

*University of California, bobmiller@ucdavis.edu
[1]Sandia National Laboratories, kmorel@sandia.gov

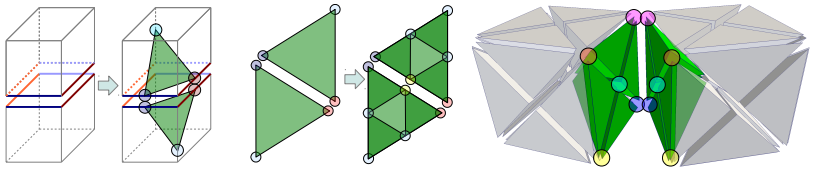


FIG. 1.1.
Operations that generate geometry usually rely on a known input topology for that purpose. Marching Cubes (left) generates all output vertices on the edges of a known voxel grid. By marking these output vertices with the ID of their generating edges, connectivity information can be retained. The same process can be applied to cell subdivision (center). Tetrahedralizations (right) are more complex, but still supportable by keying on the input face as well as the input edge. In each figure, output topological connections known from input topology are displayed by using identical colors to highlight connected vertices.

[11]. Dyken provides a fairly detailed overview of the advancements in GPU implementations of Marching Cubes and Marching Tetrahedra [6].

Kipfer and Westermann [8] use the observation that polygon vertices from Marching Tetrahedra all lie on tetrahedra edges to uniquely specify each polygon vertex. Our implementation of isosurface generation uses a similar observation, but our technique is more general than Kipfer and Westermann's and does not require the auxiliary edge structure of their technique.

Stream compaction is an important element for efficient geometry generation on the GPU. Horn provides an early method for stream compaction on the GPU [7], which was improved upon by Sengupta [12], who also provides a CUDA implementation. These approaches rely on the data parallelism technique and the application of prefix sums as described by Blelloch [3, 4]. Dyken further optimizes GPU compaction techniques by making use of a data structure called a Histogram Pyramid [6]. For our examples, we use geometry generation methods that make use of the prefix-sum method of compaction, but other compaction techniques such as histogram pyramids could be substituted for the prefix sum technique for increased performance in any case where our method is applied.

Bell, a primary developer of the Thrust library, presents a vertex welding technique as an example application of Thrust [2]. Bell also notes that GPU sorting techniques can be instrumental in the development of high-performance GPU algorithms. The technique uses a lexicographic sort directly on the vertices generated by Marching Cubes, then collapses duplicates to get the final welded surface. We find this sorting approach well suited for topology construction in finely-threaded architectures and seek to improve and generalize the technique into a new technique called Key-Value reduction.

The algorithm, hereafter referred to as Vertex-Weld, works as described in Algorithm 1. Vertices are first sorted by some representation of their coordinates. Bell suggests a lexicographic sort where first the x, then y, then z coordinates are compared. After the sort, vertices at identical locations are adjacent in the array. These coincident points are marked and compacted. Note that the lexicographic sorting method is very sensitive to floating-point error because small differences in either of the x or y coordinates may lead to nonadjacent placement of otherwise identical vertices in the sorted array.

Vertex-Weld is relevant to topology reconstruction because it provides unique identities for distinct topological features, such as vertices. Prior to the operation there may exist many instances of the same topological feature. This makes it difficult to determine whether two facets connect to the same vertex, for example. After the Vertex-Weld process is complete, it is simple to determine all facets that contain a particular vertex, which can allow computation of incidence and adjacency lists. Vertex-Weld can be

Algorithm 6 *

1: **procedure** $Vetrex-Weld((vertices))$
   ▷ *vertices*: Array of vertex data (i.e. coordinates).
   ▷ Sort vertices to make identical vertices adjacent.
2: $sorted-vertices \leftarrow Lexicographic-Sort(vertices)$
   ▷ Copy first element of each group of duplicates.
3: $welded-vertices \leftarrow Unique(sorted-vertices)$
   ▷ For each item in *vertices* find the corresponding
   ▷ index in $welded-vertices$.
4: $cell-connections \leftarrow Vectorized-Find(welded-vertices, vertices)$
5: **return** $(welded-vertices, cell-connections)$
6: **end procedure**

Alg. 1: VERTEX-WELD, as demonstrated in the Thrust library examples, takes geometric soup as input and produces a welded topology.

efficiently implemented using well-studied parallel algorithms. For example, the functions Lexicographic-Sort, Unique, and Vectorized-Find are performed using the parallel Thrust functions `sort`, `unique`, and `lower_bound`, respectively.

Our approach has many similarities, but there are several cases where Vertex-Weld was not designed to accommodate topological reconstruction. In particular, Vertex-Weld provides no way to make use of the provenance of the disconnected geometry. Additionally, the Vertex-Weld algorithm as presented is unstable in the presence of floating point errors because these can cause the Lexicographic-Sort to place "identical" points in non-contiguous regions of memory, thus invalidating the algorithm. A more stable solution which still uses bitwise comparison can be constructed by sorting on each dimension and removing duplicates each time, but this is inefficient. Instead, we resolve both of these concerns by allowing a sort on input topological features, and a generalized reduction/merge operation. These modifications give us the technique we are calling Key/Value reduction.

We pattern our algorithms after some of the features of the MapReduce framework [5]. Like MapReduce, our algorithms first map keys to values then collect keys and reduce the values. MapReduce is a system that works extremely well in distributed computing, but is not designed to work efficiently on a single node. Key/Value reduction is intended for use in highly threaded environments such as nVidia's CUDA or Intel's Xeon Phi. Although we believe our algorithms could be implemented in a MapReduce framework (an exercise we leave to the reader), it would require additional collection operations to resolve topological connections. Other researchers [13, 14] have implemented visualization algorithms directly in MapReduce, but they serve purposes other than those we address.

3. Methodology.

3.1. Using Input Mesh Topology. The essential idea behind Key/Value reduction is to use components of the input topology as *keys* in a MapReduce-like framework as demonstrated in Figure 3.1. In this figure we have labeled the major steps of our algorithm with the analogous component in MapReduce to facilitate the understanding for those already familiar with this system.

The first step is a *map* operation that generates key-value pairs where the key is some component of the input topology and the value is a generated component of the output topology. In the case of Marching Cubes, the values are the vertices generated for the new surface mesh, and each vertex is keyed by an identifier for the edge used to interpolate the vertex. The map operation may also generate elements that are

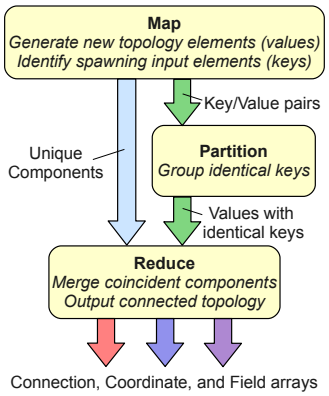


FIG. 3.1. *Basic flow of an algorithm using input mesh topology to determine characteristics of an output mesh.*

known to be unique and therefore do not need keys. For example, when subdividing cells, some vertices come directly from the input vertices whereas others are interpolated and must be connected. The map operation is designed such that key-value pairs can be generated independently, which allows us to safely compute them concurrently on a very large pool of threads.

The second step is a *partition* operation that reorganizes the key-value pairs to group duplicate keys. Like most MapReduce implementations, we find a parallel sort to be an efficient way to shuffle the data. However, we can also take advantage of domain decompositions when available to shorten the partitioning time.

The third step is a *reduce* operation that merges groups of coincident components identified by the partition and generates the connected structures. In the Marching Cubes example, this reduction means averaging the field values on merged vertices and updating the triangle connection indices. Because all dependent operations are collected into partitions, the reduce operations can also be safely computed concurrently.

Although our system is conceptually similar to MapReduce and we believe it possible to implement in a MapReduce framework, we choose to implement our algorithms using the more imperative parallel operations provided by Thrust. This makes it easier to take advantage of known properties such as domain decompositions or implicitly unique keys. It also simplifies storing topologies in efficient indexed array structures rather than collections of key-value entities.

3.2. General Merging Algorithm. Figure 3.2 provides a simple example of applying our technique using input mesh topology, described in Section 3.1, to find connections among generated vertices. First, a mapping operation generates the connections for a set of cells. Vertices are duplicated to allow independent operation on multiple threads. For the purposes of this paper we assume the map operation is a previously known algorithm such as Marching Cubes with the trivial extension that cell connection lists contain pairs of input index (key) and output vertex (value) rather than just the output vertices.

Next, key-value pairs are partitioned by sorting the pairs based on keys. From the sorted list of keys we can efficiently extract a list of unique keys, which serves to identify the connected vertices to be created. This list of sorted unique keys can also used to look up where each original unsorted key resides in the final list of merged vertices, which is how we generate the $cell-connections$ array defining the cell topology.

The final step is to merge values with identical keys in the reduction phase. This reduction operation provides an opportunity to combine neighborhood information such as averaging normals across surface polygons. To facilitate averaging we also generate a *counts* array marking the number of cells incident

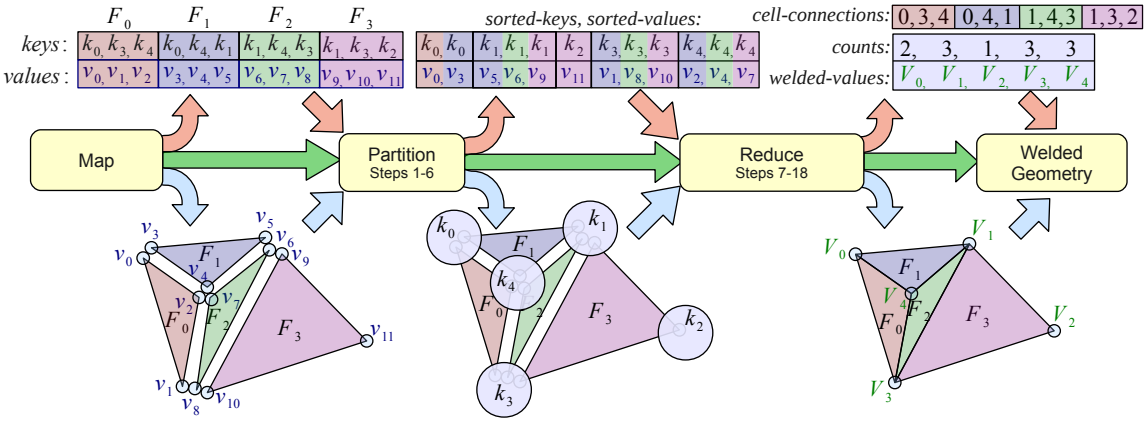


FIG. 3.2. *An example of using input mesh topology to find output vertex connections. A map operation, such as Marching Cubes, defines some collection of cells as shown at left. A partition phase groups the keys and a reduce phase combines vertices and establishes a connections array to the new indices. We collect the mechanisms of this process in an algorithm we call Key-Value-Reduce.*

to each vertex. Although this array is not specifically necessary to describe the final topology, it can be leveraged to find vertex incidence lists using the Vertex-Incidence-List method described in Algorithm 3.

We provide a generalized method that captures these partition and reduce phases named Key-Value-Reduce that groups various types of geometric elements and then merge each group into single outputs. The workings of Key-Value-Reduce are described in Algorithm 2.

The Key-Value-Reduce algorithm takes as its input data the result of a map operation. Also passed to Key-Value-Reduce are a *merge* operation, which combines two values, and a *transform* operation, which completes a reduction from a fully merged set. Together the merge and transform operations allow reductions to take place iteratively, which can be important for distributed or streaming implementations.

Our Key-Value-Reduce algorithm is implemented wholly using the parallel operations provided by the Thrust library. Steps 1 through 6 use functions that are analogous to those directly provided by thrust. Steps 7 through 18 in Key-Value-Reduce are described at a high level for clarity. For increased data parallelism, we actually implement this portion of the algorithm using the Thrust library's `inclusive_scan_by_key` and `zip_iterator`. An example of how this algorithm works is provided in Figure 3.2.

We note that the Key-Value-Reduce algorithm combines features of both MapReduce and Vertex-Weld, making it a curious amalgamation between the two. The remainder of this section describes applications of Key-Value-Reduce as a typical vertex weld (albeit potentially faster than Vertex-Weld with a more robust coincident point comparison). Subsequent sections apply the Key-Value-Reduce algorithm in more unique ways to identify different topological connections.

3.3. Histograms. Histograms are probably the simplest example of Key-Value reduction. In this case, generation of the keys simply requires that for each input element, we assign the index of the bin it should fall into as the key. The reduction operator is a no-op, because for the histogram the size of each key group is the output value.

3.4. Cell-To-Point. It is relatively common to have information about cells and need to interpolate to find this information at the cell vertices. This can be easily accomplished by parallelizing on each cell. We

Algorithm 7 *

procedure $Key-Value-Reduce((keys, values, compare, merge, transform))$
▷ keys: An array of keys uniquely identifying elements.
▷ values: An array of mergeable elements for each key.
▷ compare: A key comparator $compare(k_1, k_2) \rightarrow \text{const}\, bool$.
▷ merge: A merging operator $merge(m_1, m_2) \rightarrow m_3$.
▷ transform: A transformation $transform(m_1, size) \rightarrow out$.
▷ Sort values using keys to make groups.
$(sorted-keys, sorted-values) \leftarrow Key-Sort(keys, values, compare)$
▷ Determine the new indices for each element.
$unique-keys \leftarrow Unique(sorted-keys)$
$cell-connections \leftarrow Vectorized-Find(unique-keys, keys)$
▷ Get a reverse map from output to sorted arrays.
$reverse-map \leftarrow Vectorized-Find(sorted-keys, unique-keys)$
▷ Get the size of each group (one per unique key).
$welded-size \leftarrow Length(reverse-map)$
$counts \leftarrow \{\}$
    for all $i \leftarrow 0$ to $welded-size-2$ **do** in parallel
        $counts[i] \leftarrow reverse-map[i+1] - reverse-map[i]$
    end for
$counts[welded-size-1] \leftarrow Length(sorted-keys) - reverse-map[welded-size-1]$
▷ Merge each group into a single element.
$welded-values \leftarrow \{\}$
    for all $weld-index \leftarrow 0$ to $welded-size-1$ **do**in parallel
        $sort-index-start \leftarrow reverse-map[weld-index]$
        $count \leftarrow counts[weld-index]$
        $v \leftarrow sorted-values[sort-index-start]$
        for all $group-index \leftarrow 1$ to $count-1$ **do**
            $sort-index \leftarrow sort-index-start + group-index$
            $v \leftarrow merge(sorted-values[sort-index])$
        end for
        $welded-values[weld-index] \leftarrow transform(v, count)$
    end for
return $(welded-values, cell-connections, counts)$
end procedure

Alg. 2: The Key-Value-Reduce procedure welds vertices and allows multiple local samples of vertex attributes to be merged to form a better sampling without non-local operation during geometry generation.

use the unique indices of its vertices as the keys, and the value that we wish to interpolate as a value. On uniform grids, a simple averaging can function as the reduce step. For unstructured grids, generally we use an inverse-distance weighted average function for the reduction operator.

3.5. Point-Curvature. Finding the curvature around a point may be estimated by parallelizing on edges. During key-value generation, point IDs for the keys, and the curvature along the edge for the value (calculable simply from the change in normal on the edge). The reduction step can then be a simple average

Algorithm 8 *

1: **procedure** $Vertex-Incidence-Lists((cell-ids, cell-connections, counts))$
▷ *cell-ids*: An array of cell identifiers for each cell vertex.
▷ usually sequential repeated indices $\{0,0,0,1,1,1,...\}$
▷ *cell-connections*: Result from Key-Value-Reduce.
▷ *counts*: Result from Key-Value-Reduce, cell counts per vertex.
▷ Use a key-sort with cell connections as the keys to
▷ group cell identifiers by vertices.
2: $(-, links) \leftarrow Key-Sort(cell-connections, cell-ids)$
▷ The size of each incidence list is stored in count.
3: $links-counts \leftarrow counts$
▷ Use an exclusive scan to find the offset to the
▷ incidence list for each vertex.
4: $links-offsets \leftarrow Exclusive-Scan(links-counts)$
5: **return** $(links, links-counts, links-offsets)$
6: **end procedure**

Alg. 3: Vertex-Incidence-Lists takes the output of the Key-Value-Reduce algorithm and quickly generates an incidence list, represented as an array of *cell-ids* and an array of pointers into *cell-ids* called *links-offsets* that represents the start of each cell's incidence list. Adjacency lists across edges may be produced by removing incidences that do not occur exactly twice in a cell, and adjacency lists across faces may be generated similarly.

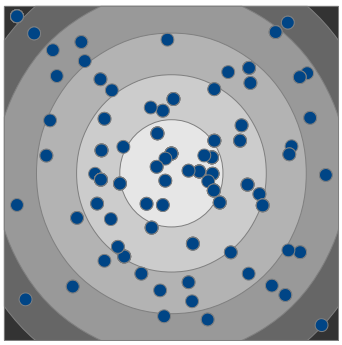


FIG. 3.3. *During the key-generation phase, each of the points assigns itself a bin ID based on the distance from a point of interest. No real reduction is needed, but the number of values in each bin is required for the histogram.*

to get the local curvature at each point.

3.6. Marching Cubes. In the case of Marching Cubes (or any variant like Marching Tetrahedra), we know a good deal about the input topology on which the output is generated. Specifically, the output vertices from Marching Cubes are generated only on the edges of input voxels.

For a structured voxel grid, we assign each unique edge an implicit integer index, then store the corresponding edge index for each vertex generated by Marching Cubes. Because the edge indices are implicit, we need neither to create nor to store an edge list. For unstructured grids, no such implicit edge identifiers necessarily exist. However, each edge is uniquely identified by its two end vertices. We build unique local

edge indices by concatenating these two end vertices in a canonical order (we put the smallest vertex first). This unstructured approach works fine for voxel grids, but requires more bits in the indices than necessary and can thus slow down the key sort.

The results of this keyed Marching Cubes map are completed using the Key-Value-Reduce process described in Section 3.2, which results in a manifold surface. We also offer to blend other field information at the vertices. This is particularly helpful for creating interpolated surface normals from flat triangle normals or input gradients, which are not continuous across cell boundaries.

3.7. Mesh Coarsening. One of the more curious attributes of our topology construction is that there is flexibility in how we record the history of topological elements, and changing the association between input and output elements can have interesting and useful outcomes. In this section we describe how to alter the topological provenance of the Marching Cubes algorithm to provide an effective but free first-level coarsening of the resulting surface.

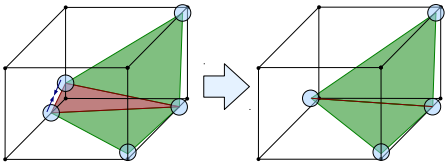


FIG. 3.4. *When several Marching Cubes output vertices (blue) fall close to the same input vertex (black), small or skinny triangles may be produced (red). Merging the nearby vertices collapses the skinny triangles into degenerate triangles, which are then removed.*

The basic idea for our coarsening is described in Figure 3.4. Marching Cubes generates small or skinny triangles when two adjacent output vertices are generated near the same input mesh vertex. These skinny triangles are a problematic artifact of Marching Cubes. They lead to poor interpolation of surface normals, shading, and other fields as demonstrated in Figure 3.5. The juxtaposition of skinny and fat triangles can also cause irregularities in their orientations, which can yield a staircase-like appearance as evident in Figure 3.6.

Although our technique reduces the mesh size, philosophically speaking we are more interested in improving the quality of the mesh than in reducing the number of triangles, which is the primary focus of previous work. Specifically, it is our goal to eliminate poor quality triangles in the mesh while performing minimal modifications to the others, without requiring an additional processing pass. We define triangle quality Q as

$$Q = \frac{4a\sqrt{3}}{h_1^2 + h_2^2 + h_3^2}, \tag{3.1}$$

where h_x are side lengths and a is the triangle area. This quality metric operates only on triangle shape, and is agnostic to triangle size.

This measure of triangle quality considers only aspect ratio, and is such that triangles where $Q > 0.6$ are considered to be of acceptable quality, and $Q = 1$ when the triangle is equilateral [1]. This is equivalent to the pdetriq function in MATLAB. We use this metric to empirically demonstrate the effectiveness of our coarsening.

Figure 3.7 demonstrates the effect of our coarsening to the isosurface of the MRI head rendered in Figure 3.6. In general the triangles that are removed by our coarsening (green) are either small or poor quality as compared with other triangles in the distribution. Triangles that are not removed by the coarsening (orange) are altered into the output triangles (blue) when point coordinates are averaged. Although not all

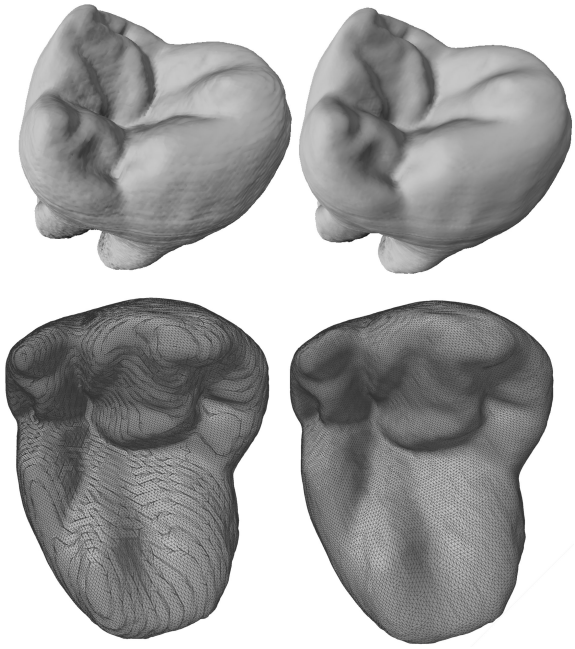


FIG. 3.5. *Marching Cubes output without (left) and with (right) coarsening on the tooth dataset. The top images highlight the reduction of normal artifacts via our coarsening. The bottom images show improvement in mesh quality when coarsening is enabled.*

FIG. 3.6. *At left artifacts are visible due to sampling error caused by the local operation of our Marching Cubes implementation. At center surface normals are averaged on merged vertices, and smooth shading attempts to make the bumps less noticeable. At right we remove these artifacts using our coarsening without incurring additional performance costs.*

passed triangles are improved by this coordinate averaging, there is a lower bound on produced triangle quality. After coarsening, most of the triangles are relatively large and high quality. The cumulative triangle distributions show that our coarsening reduces the size of the output mesh to approximately half of its original size. The median quality of the removed triangles is approximately $Q = 0.5$ whereas the median quality of passed and output triangles is approximately $Q = 0.85$.

A problem with the coarsening technique is that in some cases, such as the case shown in Figure 3.8, non-manifold surface elements may be generated by collapsing vertices from different parts of the surface. In this example, two vertices from different faces are merged and a 3D volume is collapsed into a single plane. If such surface elements are undesirable, they may be eliminated in one pass by removing all triangles containing an edge curvature of 180°.

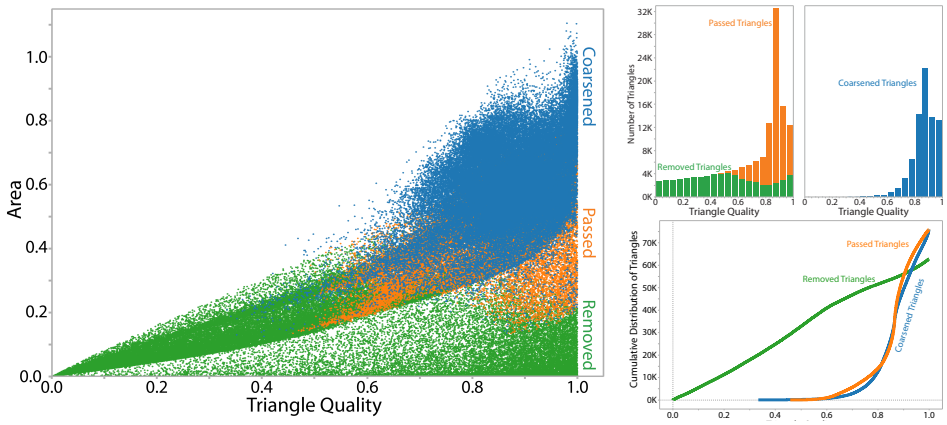


FIG. 3.7. *Left: Scatterplot of triangle area and triangle quality for the MRI head dataset as calculated in Eq. 3.1. Top Right: Histograms of the quality of passed, removed, and coarsened triangles. Bottom Right: Cumulative distributions of triangles before and after coarsening.*

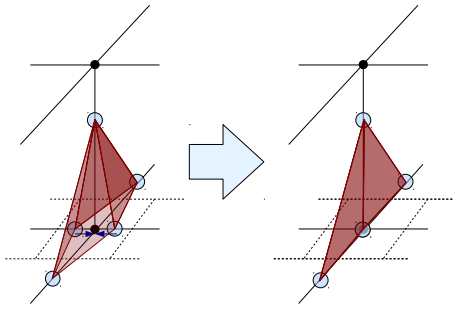


FIG. 3.8. *An example of mesh coarsening causing a non-manifold, overlapping surface.*

4. Results and Discussion. All tests were run on a Macintosh Pro with 2x2.26 GHz Quad Core Intel Xeon processors, 6GB 1066 MHz DDR3 ECC memory, and an nVidia GeForce GT 120 512MB. Due to space constraints, we are omitting the results for Tetrahedralization and Subdivision, but these perform similarly to the Marching Cubes case on similar grid size, which is described below. In each of these cases, performance is largely determined by the size of the final output grid.

4.1. Histogram. Performance results using the Key-Value reduce operation to create a histogram on 10,000,000 points and 5000 bins is shown in Figure 4.1. This point and bin count were chosen due to contiguous memory limitations of the graphics card.

4.2. Overhead of Key-Value-Reduce. Figure 4.2 shows an example of taking a traditional vertex welding approach on the output of Marching Cubes and replacing it with Key-Value-Reduce. Within the variance of our tests, it does not appear to provide a significant loss of performance. The input grid is a 512x512x512 grid, with approximately 25% of the cells creating output geometry. The size of the input grid is limited by available memory on the graphics card.

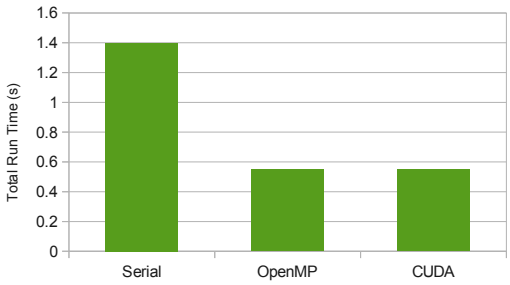


FIG. 4.1. *Time to run a histogram of 10,000,000 points with Key-Value-Reduce.*

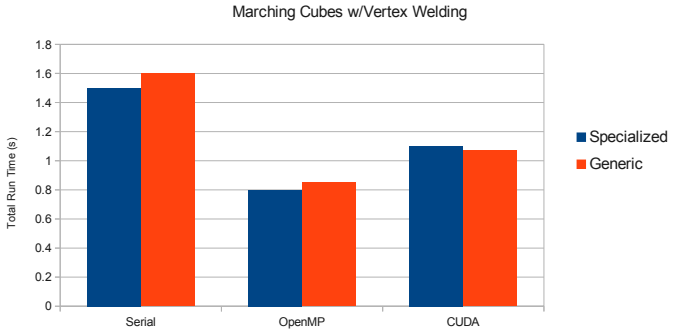


FIG. 4.2. *Vertex welding timing information on output from Marching Cubes on a 512x512x512 grid. Changing from the specialized Vertex-Weld method to our Key-Value-Reduce method does not incur significant overhead.*

4.3. Coarsening. Figure 4.3 shows that when using Key-Value-Reduce as a vertex welding operation in conjunction with a coarsening operation, we can actually reduce execution time while improving triangle quality as shown in Figure 3.5.

5. Conclusion. We implement the Key-Value reduction pattern for a variety of distinct visualization operations, and demonstrate that it is efficient on both CUDA and OpenMP. The Key-Value pattern does not fit all visualization operations, but where it fits, it allows for simple and efficient implementation in finely-threaded environments.

6. Acknowledgements. This work was supported in part by the DOE Office of Science, Advanced Scientific Computing Research, under award number 10-014707, program manager Lucy Nowell.

This work was supported in part by the Director, Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy under Contract No. 12-015215, through the Scientific Discovery through Advanced Computing (SciDAC) Institute of Scalable Data Management, Analysis and Visualization.

Sandia National Laboratories is a multi-program laboratory operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear

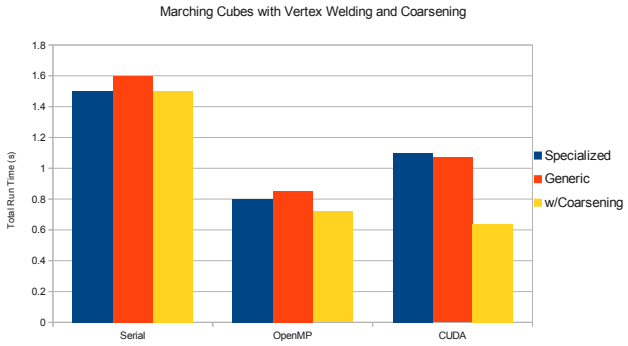


FIG. 4.3. *Comparison of previous techniques for Vertex-Welding with the coarsening approach on the output of Marching Cubes on a 512x512x512 grid. CUDA appears to be able to take advantage of the smaller number of functor calls due to the presence of more identical keys.*

Security Administration.

REFERENCES

[1] R. E. BANK AND M. HOLST, *A New Paradigm for Parallel Adaptive Meshing Algorithms*, SIAM Review, (2003).
[2] N. BELL, *High-Productivity CUDA Development with the Thrust Template Library*, 2010.
[3] G. BLELLOCH, *Prefix sums and their applications*, Synthesis of Parallel Algorithms, (1990).
[4] G. E. BLELLOCH, *Vector Models for Data-Parallel Computing*, Computing Control Engineering Journal, (1991).
[5] J. DEAN AND S. GHEMAWAT, *MapReduce: Simplified data processing on large clusters*, Communications of the ACM, (2008).
[6] C. DYKEN, G. ZIEGLER, C. THEOBALT, AND H.-P. SEIDEL, *High-speed Marching Cubes using HistoPyramids*, Computer Graphics Forum, (2008).
[7] D. HORN, *Stream Reduction Operations for GPGPU Applications*, in GPU Gems 2, 2005.
[8] P. KIPFER AND R. WESTERMANN, *GPU construction and transparent rendering of iso-surfaces*, in Proceedings Vision, Modeling and Visualization, 2005.
[9] W. E. LORENSEN AND H. E. CLINE, *Marching cubes: A high resolution 3D surface construction algorithm*, SIGGRAPH, (1987).
[10] J. PARK, B. CHOI, AND Y. CHUNG, *Efficient topology construction from triangle soup*, in Geometric Modeling and Processing, Proceedings, 2004.
[11] S. RÖTTGER, M. KRAUS, AND T. ERTL, *Hardware-accelerated volume and isosurface rendering based on cell-projection*, in Proceedings of the conference on Visualization'00, 2000.
[12] S. SENGUPTA, M. HARRIS, Y. ZHANG, AND J. D. OWENS, *Scan Primitives for GPU Computing*, Computing, (2007).
[13] J. A. STUART, C.-K. CHEN, K.-L. MA, AND J. D. OWENS, *Multi-GPU volume rendering using MapReduce*, in 1st International Workshop on MapReduce and its Applications, June 2010.
[14] H. T. VO, J. BRONSON, B. SUMMA, J. L. COMBA, J. FREIRE, B. HOWE, V. PASCUCCI, AND C. T. SILVA, *Parallel visualization on large clusters using MapReduce*, in Proceedings of the IEEE Symposium on Large-Scale Data Analysis and Visualization, 2011.

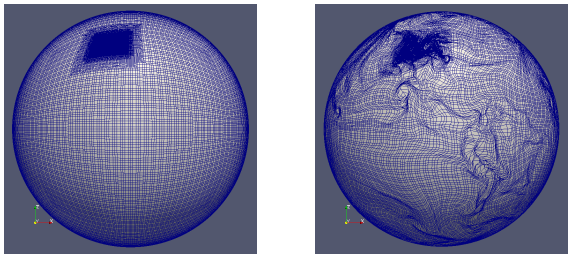
PARALLEL TRIANGLE CLIPPING ON GPU

XIN TONG* AND KENNETH MORELAND[1]

Abstract. Polygon clipping, the process of finding the intersection of two arbitrary polygons, is a well studied problem. However, general algorithms have many cases and conditions. Such conditional execution is problematic when using a SIMD-like GPU to concurrently clip many polygons with few vertices. We present a clipping technique specialized for triangles and quadrilaterals that runs effectively when replicated in a thread warp. We apply our technique on spherical quadrilateral unstructured grid data, and achieve more than 10 times speedup over traditional techniques.

1. Introduction. Triangle clipping is a simple case of 2D polygon clipping, a fundamental operation in computer graphics [4]. Basically, it clips one polygon by another polygon, and generates a polygon that is the intersection of the two input polygons. We can use the triangle clipping to solve the incremental remapping problem.

Incremental remapping [3] [8] is a strategy to solve the standard transport or continuity equation, which projects cell volumes along Lagrangian trajectories and then remap the resulting density distribution onto the original mesh. There are two basic approaches to solve the incremental remapping: computing the cell-face fluxes, and computing the cell area average of transported concentration. For regular grids, cell-face flux is easier to compute with a computationally economic approach. But for unstructured grids, the flux area is no longer easy to compute so we can instead compute the cell area average of transported concentration. Computing the cell area average requires computing the intersection of two grids, one original grid of the data(Figure 1.1(a)) and one warped grid (Figure 1.1(b)) generated by warping the original grid along the Lagrangian trajectories. The polygon clipping process should output a clipped grid whose cells are the clipped polygons generated by clipping the polygonal cells from the warped grid by the polygonal cell from he original grid.



(a) Original Grid (clipping polygons)

(b) Warped Grid (subject polygons)

FIG. 1.1. *Comparison between two sets of grid*

Polygon clipping between two large grids is a computationally intensive work, because the number of polygons to clip grows with grid size. The grid size becomes larger with the increasing of the computation power. Thus, doing the polygon clipping fast is necessary for large size data.

Climate scientists perform simulation on different types of grids. In this paper, we use the grid from the Community Atmosphere Model version 5 (CAM5) [2] as our input data. CAM5 is based on a cubed-

*The Ohio State University, tong@cse.ohio-state.edu
[1]Sandia National Laboratories, kmorel@sandia.gov

sphere grid, which allows quasi-uniform grids for the sphere, as in Figure 1.1(a). A cubed-sphere grid is a coordinate system derived from central projections [13], which splits the earth into regions of quadrilaterals.

A parallel computation on a GPU can achieve a fast parallel clipping based on the independency property of the computation. The traditional clipping algorithm is a serial algorithm based on CPU architecture, which is not efficient for GPU computation. In this paper, we propose a parallel algorithm to perform triangle grid clipping with a GPU program. This algorithm can trivially be run concurrently for multiple triangle pairs, but it contains many cases and branches that make it inefficient on a GPU. So we use a lookup table to store the branches and their corresponding instruction sets to minimize the number of instructions in the GPU kernel program. We extend the algorithm to solve quadrilateral grid clipping by splitting each quadrilateral into two triangles. The cubed-sphere grid from the CAM5 model is used in this paper as the quadrilateral grid. A search structure is designed for the cubed-sphere grid to accelerate the process of searching triangle pairs. Since incremental remapping only needs the integration over the clipped polygon, so our approach can be applied to incremental remapping on any kinds of 2D grids that can be triangulated to triangle grid.

This paper proceeds as follows. First, We summarize the previous polygon clipping algorithm in Section 2. Then, we discuss our parallel triangle clipping algorithm in Section 3, and the triangle search structure in Section 4. At last, we evaluate the results and performance of our method in Section 5, and conclude our work in Section 6.

2. Related Works. Polygon clipping against a fixed plane is a well studied area in computer graphics. The clipping technique clips polygons against the screen boundary and remove the part of polygon that lie outside of the screen so that they would not be rendered. Sutherland and Hodgman [14] proposed a reentrant polygon clipping algorithm (Sutherland-Hodgman algorithm) that clips polygon against convex windows. Liang and Barsky [7] provided a clipping method that can better handle the difficult case that the clipping polygon surrounds a corner of the clip window than the previous methods. Maillot [1] simplifies the Sutherland-Hodgman algorithm by only considering the polygon being triangle or quadrilateral. It achieves better memory usage than the Sutherland-Hodgman algorithm. Maillot's another paper [10] extends the Sutherland-Cohen line clipping algorithm [11] to clip polygon against a rectangular window. The focus of the paper is taking care of the turning point when a polygon is clipped by two planes at the corner of the screen. McGuire also provided an efficient implementation of Sutherland-Hodgman algorithm to clip triangle and quadrilateral against a fixed plane. This algorithm is implemented in GPU shader so that it is efficient on a GPU parallel architecture. These clipping algorithms above clip polygon against fixed planes. Our triangle-triangle clipping problem is to clip polygon against anther polygon. It is similar to the problem of clipping against a fixed plane because both problems involve solving intersections and generating an ordered sequence of vertices as the final output. But the polygon-polygon clipping problem has more cases to consider and is more complex.

A few approaches are available to clip an arbitrary polygon by another arbitrary polygon. Weiler algorithm [17] and Vatti's algorithm [15] can clip between two arbitrary polygons. But both algorithms are complicated to understand and to implement. Greiner and Hormann [4] proposed a simpler approach to clip between two arbitrary polygons. The idea is to build a linked list of the vertices and intersection points for each of the two polygons. Then the algorithm traverses the two linked lists to find the vertices of the clipped polygon. This method removes the degenerate cases by perturbing the vertices. Other works [6] [9] extend Greiner's method to handle degenerate cases. The existing polygon-polygon clipping algorithms run on a single CPU and achieves a reasonable performance. However, these algorithms are designed to clip arbitrary polygon that has arbitrary number of vertices, which is not good for a GPU implementation. The relatively simple Greiner's algorithm requires the linked list data structure, which prevents the algorithm map on the GPU memory well. Our method simplifies the problem by only clipping against two triangles that have fixed number of vertices and does not require using linked list, which is easy to map onto GPU memory and

achieve a high performance with a parallel GPU implementation.

3. Triangle Clipping Algorithm. The classic polygon-polygon clipping algorithm is about arbitrary polygon based on a linked list data structure [4]. However, the size of linked list varies largely, making it hard to map on GPU memory. Clipping on a big data set of polygons on a CPU is very time consuming, so we want to modify the data structure and algorithm to make a clipping technique for the GPU. Our algorithm restricts the input polygon to be triangle to simplify the data structure and reduces the number of cases to consider when clipping triangles.

3.1. Algorithm Overview. The polygon-polygon clipping problem clips a *subject polygon*, against a *clipping polygon*, and produce a *clipped polygon*. The vertices of the clipped polygon are either from the vertices of the two input polygons, or their intersection points. The vertex of the clipped polygon from the input polygons must be a *inside vertex*, which is the vertex of a polygon that is inside the shape of the other polygon. The general approach for polygon clipping is first find out all the inside vertices and intersection points of the two triangles, and then sort those points based on the geometry to assemble the clipped polygon.

Comparing to clipping between two arbitrary polygons, there are limited number of clipping cases between two triangles. Each case shares the same set of standard operations to assemble the clipped polygon. Note that in the triangle-triangle clipping, the subject polygon and the clipping polygon can be interchanged without changing the result (the clipped polygon remains the same). So we do not differentiate the symmetric cases. We differentiate the 6 cases by the number of inside vertices as listed below.

- None of the two triangles has any inside vertex
- One of the two triangles has one inside vertex
- One of the two triangles has two inside vertices
- One of the two triangles has three inside vertices
- One triangle has one inside vertex, the other has two inside vertices
- Both triangles have one inside vertex

After the 11 cases are determined, the next step is to use 14 instruction statements to generate the array for clipped polygons. We first initiate an array of the space of 6 vertices, because we know there are at most 6 vertices in the clipped polygon. Then we fill this array by adding vertices one by one, and keeping the count of added vertices so far. How to sort those points is the tricky part in designing a clipping algorithm, which will be described below.

3.2. Clipping Cases. In the implementation we differentiate the symmetric cases, and enumerated 11 cases as shown in Figure 3.3. Each case shows a set of the possible intersection configurations. The the green triangle is the subject triangle, and the red triangle is the clipping triangle. Case 1 shows the case that none of the two triangle has any inside vertex. For case 1, there is no inside vertex for either triangle. The clipped polygon is composed of only the intersection points of the two triangles. The number of vertices in the clipped polygon can be 3, 4, 5 or 6, as shown in Figure 3.3. We consider them as the same case because the clipping operations for generating the clipped polygon are the same. Case 2 and case 3 are symmetric cases that one of the two triangles has one inside vertex. Case 4 and case 5 are symmetric cases that one of the two triangles has two inside vertices. Case 6 and case 7 are symmetric cases that one of the two triangles has three inside vertices. Case 8 and case 9 are symmetric cases that one triangle has one inside vertex, the other has two inside vertices. Case 10 and case 11 are for the situation that both triangles have one inside vertex. The last two cases are not symmetric. The first configuration of case 10 has two intersection points. The difference between the second configuration of the case 10 and the case 11 is their directionality of the vertices of the two triangles, clockwise or counterclockwise. The difference between case 10 and case 11 is that in case 10 the vertices of two triangles have the same directionality, i.e. both are clockwise or both are counterclockwise; while in case 11 the directionalities are different. The case 10 and the case

11 can be differentiated by testing whether the edge s_{23} intersects with the edge c_{12}. Note that we don't restrict the directionality of the input triangles to be the same. In our application of incremental mapping, the directionality of the subject triangle is unpredictable because the directionality can change after warping.

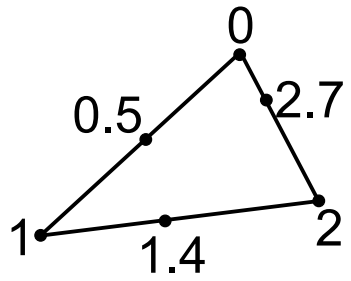


FIG. 3.1. *Relative position used to sort the intersection points and the vertices on a triangle. A point on the triangle boundary has a relative position represented as a floating point number. The relative positions of 6 points are presented as examples marked on the triangle.*

After differentiate the 11 cases with the number of inside vertices from the two triangles, we detect all the intersection points. Then we mark the intersection points with a value that gives the *relative position* on one of the triangle. The relative position is a value in $[0,3)$. The three vertices have a relative position of 0, 1 or 2. The relative position of a point on an edge of the triangle is a floating point value between the relative positions of the two incident vertices of the edge based on its relative distance to the two vertices. We know that the order of vertices and intersection points in one triangle should remain the same in the clipped triangle. If we keep this value to order the vertices in the clipped polygon, we can only choose one of the triangle to order the vertices in the clipped polygon. Here we choose the triangle has the larger number of inside vertices. The inside vertice are swapped to the front of the vertex array, so their index is smaller than the outside vertices. Because there are at most 6 vertices in the clipped polygon, we declare an array of 6 vertices. To assemble the clipped polygon of the case 1-7 that at most one of the triangle has inside vertices, we first put all the intersection points into the array with the order of their relative position in the triangle that has the inside vertices, and then put all the inside vertices into the result array according to its relative position. To assemble the clipped polygon of the case 8 and case 9, we first put in the only two intersection points, and the inside vertex from the triangle who has only one inside vertex between the two intersection points. Then we append the two inside vertices from the triangle that has two inside vertices. For the case 10, we first add all the intersection points from the clipping triangle,

We enumerate 11 cases for the intersection of two triangles, as shown in Figure 3.3. 3 tests are performed to distinguish the 11 cases. They are (T1) the number of inside vertices inside of the subject triangle, (T2) the number of inside vertices of the clipping triangle, and (T3) an intersection test between two edges from the two triangles, as shown in Table 3.1.

3.3. Tests on Spherical Coordinates. Inside Test and Intersection Test on spherical coordinates.

The clipping is performs with a few steps. The first step is figuring out which case the two triangles' relation belongs to. To do this, we need a test to decide whether a point is inside a triangle on the sphere surface. Note that the triangle is a spherical triangle, i.e. the edge between two points is on the great circle of the sphere and also the shortest path between these two points on the sphere surface. Thus, edges of the triangle are actually arcs on the surface of the sphere. In Figure 3.2, A, B and C are three points on the sphere and form a triangle. O is the sphere center. The three planes OAB, OBC, and OCA bound the spherical triangle. We define the normals to these three planes as N_{OAB}, N_{OBC}, N_{OCA}, respectively. These normals have

the property that they either all point into the triangle or away from the triangle. G is inside the triangle if and only if the three dot products $OG \cdot N_{OAB}$, $OG \cdot N_{OBC}$, and $OG \cdot N_{OCA}$ are all positive or all negative.

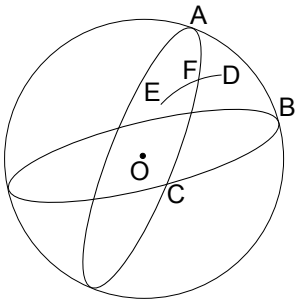


FIG. 3.2. *Inside test and intersection test for spherical triangle. The right figure is a magnified view of the triangle ABC in the left figure*

If both triangles have one inside vertex, we then perform an intersection test to distinguish between cases 10 and 11. The subject triangle has the vertices s_1, s_2, s_3, and the clipping triangle has the vertices c_1, c_2, c_3. As shown in cases 10 and 11 of Figure 3.3, the intersection is performed between edges s_{23} and c_{12}. We use plane intersections in Cartesian space to find intersections of arcs on the sphere. In Figure 3.2, we need to know whether arc AC and arc GE would intersect at some point, say point F. We know that the line OF is contained in both the plane OAC and OEG. So we have $OF \cdot (OA \times OC) = 0$, and $OF \cdot (OG \times OE) = 0$. Besides, F is on the intersection line of these two planes 1 unit (the sphere's radius) away from O, which is easily computed. F is just the intersection point of the two great circles, which may lay outside of the arc AC and BG. Then we test whether F is between A and C and between G and E. For example, if F is between A and C, the three cross products, $OA \times OF$, $OF \times OC$ and $OC \times OA$ should have same sign (same direction), i.e. $(OA \times OF) \cdot (OF \times OC) > 0$ and $(OF \times OC) \cdot (OC \times OA) > 0$.

3.4. Branchless . As we know the branches in a GPU program kill the computation performance, because a thread will run a branch as long as there is one thread in the same warp needs to go through this branch. To minimize the amount of branching on the GPU, all the cases share the same instructions for clipping and use a mask to determine which instructions get executed. This allows different cases to share instructions among the threads in a warp. For the triangle clipping algorithm, we summarize 14 C++ instruction statements. Some of the statements are used by many of the cases, but we write this statement only once in the GPU kernel method and execute it only for those cases that need it. Note that these 14 statements are in order. When running one statement for a certain case, the mask will have an effect on determining whether to run this statement or not. The value of the mask is checked from Table 3.1 for each case and each statement.

In Table 3.1,The first two statements(I1 and I2) are for adding all the intersection points to the clipped triangle array. Each time a intersection point is detected, its relative position on the triangle is recorded. There are at most two intersection points on one edge, so as long as we find a new intersection point whose relative location is smaller than the previous intersection point on the same edge, we put the new point before the previous intersection point, instead of appending to the clipped triangle array. Note that for the same intersection point, its position values are different in the two triangles, which is why we differentiate the intersection instructions by the statements I1 and I2. Instructions I3 to I14 are for assigning vertex values and changing the valid size of the vertex array of each clipped polygon.

Two triangles, subject triangle and clipping triangle, intersect in different orientations with a different number of intersection points. To define the clipping process, the clipping triangle clips the subject triangle and generates the clipped polygon, whose number of vertices varies from 3 to 6. As demonstrated in Figure 3.3, the vertices of the clipped polygon include all the intersection points and any *inside vertex*, which is a triangle's vertex that is inside the other triangle. However, in our simple case, we can exhaust all the cases of triangle intersection and build a table for the methods of finding the clipped polygon for all different cases. So the GPU program just needs to check the table to figure out how to build the vertex array of the clipped polygon after figuring out the case of the intersection.

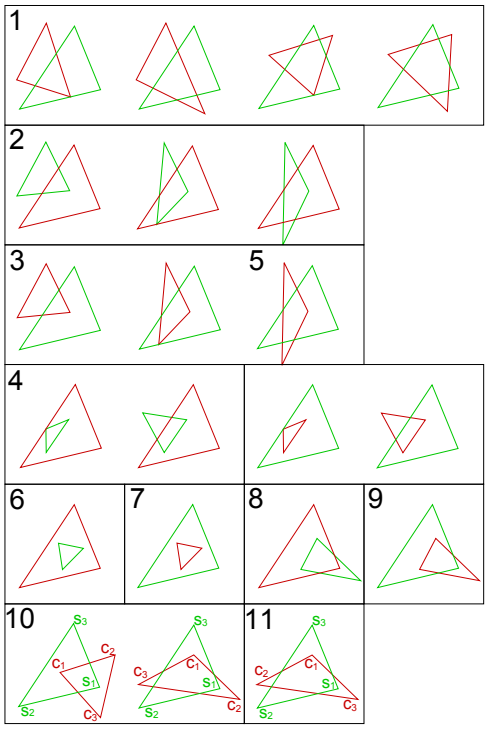


FIG. 3.3. *The 11 cases of clipping. The green triangle is the subject triangle, and the red triangle is the clipping triangle.*

3.5. Parallel Results Reduction. The results are stored in two arrays. One array stores the coordinates of the polygon vertices, where each polygon has the space to put 6 vertices. The other array stores the number of vertices each polygon has. Even when a polygon's number of vertices is less than 6, we still reserve 6 vertices' space for it in the vertex array. Then, we need to squeeze the array and remove the blank vertices in order to store it in a VTK unstructured grid file. First we do a prefix sum for the array of vertex counts and get the offset array for the position of writing the new tight vertex coordinates array. Then we fill the new array with the value of the polygon vertex coordinates in parallel by referring the offset array to know the writing position of the new array in parallel. After we get the array of vertex coordinates and the array of number of vertices for each clipped polygon, we write them into the VTK unstructured grid file as our final result.

Index	Descriptions	1	2	3	4	5	6	7	8	9	10	11
T1	number of inside vertices in the subject triangle	0	1	0	2	0	3	0	2	1	1	1
T2	number of inside vertices in the clipping triangle	0	0	1	0	2	0	3	1	2	1	1
T3	intersect s_{23} and c_{12}	-	-	-	-	-	-	-	-	-	0	1
I1	Add intersection points from the subject triangle	1	1	0	1	0	0	0	1	0	0	0
I2	Add intersection points from the clipping triangle	0	0	1	0	1	0	0	0	1	1	1
I3	d[cnt] = d[cnt - 1]	0	0	0	0	0	0	0	1	1	0	0
I4	d[cnt - 1] = s[0]	0	0	0	0	0	0	0	0	1	0	1
I5	d[cnt - 1] = c[0]	0	0	0	0	0	0	0	1	0	0	0
I6	cnt++	0	0	0	0	0	0	0	1	1	0	0
I7	d[cnt++] = s[0]	0	1	0	1	0	1	0	1	0	0	0
I8	d[cnt++] = c[0]	0	0	1	0	1	0	1	0	1	1	1
I9	d[cnt++] = s[1]	0	0	0	1	0	1	0	1	0	0	0
I10	d[cnt++] = c[1]	0	0	0	0	1	0	1	0	1	0	0
I11	d[cnt++] = s[2]	0	0	0	0	0	1	0	0	0	0	0
I12	d[cnt++] = c[2]	0	0	0	0	0	0	1	0	0	0	0
I13	d[cnt++] = d[0]	0	0	0	0	0	0	0	0	0	1	0
I14	d[0] = s[0]	0	0	0	0	0	0	0	0	0	1	0

TABLE 3.1

Tests and instructions for 11 cases. Each column is one case. Each row is a test(T1-T3) or a instruction(I1-I14). For the instructions, 1 means executing the instruction, while 0 means not executing. s[] and c[] are arrays holding 3 vertices of triangle S and C, respectively. Array d[] hold the vertices of clipped polygon. cnt is the count of valid vertices in array d[].

4. Triangle Search Structure. The input data are two quadrilateral grids. Since our clipping is based on triangles, we break each quadrilateral into two triangles by adding an edge between the shorter diagonal of the quadrilateral, which makes sure a concave quadrilateral is triangulated properly. Then, we have two set of triangles S and C, which contain the number of n_s and n_c triangles, respectively. Theoretically, we need to clip on $n_s \times n_c$ pairs of triangles. However, we do not have to do the clipping for two triangles that are far away from each other. We design a search structure to help remove the far away triangle pairs with lower cost and generate the clipping pairs candidates.

The search structure is built on two stages. The first stage divides the space into uniform partitions [5] and finds all pairs of triangles contained in the same partition. The second stage makes a bounding box intersection test for the triangle pairs to remove the pairs whose bounding box does not intersect. Because of the spherical space, we use a cubed-sphere space to make partitions and build the bounding boxes of triangles to achieve higher accuracy and balanced GPU workload.

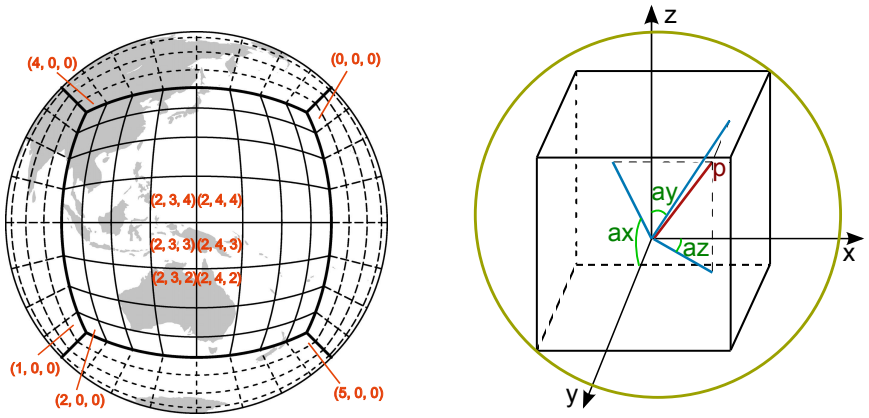
4.1. Build Search Structure on Cubed-Sphere Grid. Our data is on the surface of the Globe. Partitioning with 3D Cartesian coordinates is not useful because the there is no triangle inside the globe and the triangles are not distributed uniformly in Cartesian space. On the other hand, spherical coordinates (latitude and longitude) are not efficient either because the latitude-longitude regions at and near the poles make for ineffective regions in a search structure.

The space we use in our search structure is cubed-sphere grid [12] [13] [16], which is the same as the given data. This grid divides the sphere into 6 identical regions by the central (gnomonic) projection of the 12 edges of an inscribed cube in the sphere, as shown in Figure 4.1(a). Each of the regions on the sphere is divided by the central projection of a regular grid on a face of the inscribed cube. We index the grid by a 3D tuple (f, ix, iy), where f is the face integer in the range $[0,5]$ and ix and iy are the 2D grid index on each face. In this grid, each partition has similar area. As long as the triangles are relatively evenly distributed on the sphere surface, this grid gives us a balanced workload. A further optimization would be building a hierarchical search structure, e.g. quad-tree [18], to optimize the case that triangles are not evenly distributed, which is a future work. This cubed grid partition conservatively removes non-intersecting triangles because if the partitions of two triangles are not overlapped, the two triangles cannot intersect.

Our search structure tells which partition each of the triangles belongs to or which triangles are con-

θ_x	θ_y	θ_z	face	local coordinates
-	$[\frac{\pi}{4}, \frac{3\pi}{4})$	$[\frac{7\pi}{4}, 2\pi) \cup [0, \frac{\pi}{4})$	0	$(\theta_y - \frac{\pi}{4}, (\theta_z + \frac{\pi}{4}) \% \frac{\pi}{2})$
-	$[\frac{5\pi}{4}, 7\pi/4)$	$[\frac{3\pi}{4}, \frac{5\pi}{4})$	1	$(\theta_y - \frac{5\pi}{4}, \theta_z - \frac{3\pi}{4})$
$[\frac{7\pi}{4}, 2\pi) \cup [0, \frac{\pi}{4})$	-	$[\frac{\pi}{4}, \frac{3\pi}{4})$	2	$(\theta_z - \frac{\pi}{4}, (\theta_x + \frac{\pi}{4}) \% \frac{\pi}{2})$
$[\frac{3\pi}{4}, \frac{5\pi}{4})$	-	$[\frac{5\pi}{4}, \frac{7\pi}{4})$	3	$(\theta_z - \frac{5\pi}{4}, \theta_x - \frac{3\pi}{4})$
$[\frac{\pi}{4}, \frac{3\pi}{4})$	$[\frac{7\pi}{4}, 2\pi) \cup [0, \frac{\pi}{4})$	-	4	$(\theta_x - \frac{\pi}{4}, (\theta_y + \frac{\pi}{4}) \% \frac{\pi}{2})$
$[\frac{5\pi}{4}, \frac{7\pi}{4})$	$[\frac{3\pi}{4}, \frac{5\pi}{4})$	-	5	$(\theta_x - \frac{5\pi}{4}, \theta_y - \frac{3\pi}{4})$

TABLE 4.1
Conversion from 3D angles to 2D local coordinates



(a) Cubed-sphere partition index [16] (b) 3 angles around the axis for point p

FIG. 4.1. *Coordinate system on cubed-sphere grid*

tained by the each of the partitions. One triangle can overlap on multiple partitions, and one partition can contain multiple triangles, so this is a many-to-many relationship between triangles and partitions. This relationship can be represented by a $R \times 2$ array, where each row shows a pair of numbers (t, p) showing the overlapping of triangle index t and partition index p, as shown in Table 4.1(a) for the triangle set S in the example given in Figure 4.2. This 2D array is the search structure we need to build for this set of grid. We need the search structure for each of the two grids.

To build the search structure, we first need to figure out the 3D partition index (f, ix, iy) for each triangle. Each triangle vertex, (lon, lat), is contained in a unique partition (f, ix, iy). To compute the (f, ix, iy) from (lon, lat), we need to convert (lon, lat) into a 3D coordinates $(\theta_x, \theta_y, \theta_z)$, where $\theta_x, \theta_y, \theta_z$ are the angle around the x, y, z axis, respectively, as shown in Figure 4.1(b). We can see that θ_z is defined the same way as longitude. From the three values $\theta_x, \theta_y, \theta_z$ we can get the face f and the local 2D coordinates (ix, iy) by checking the Table 4.1. The local 2D coordinate is calculated with two values from $\theta_x, \theta_y, \theta_z$, also shown in Table 4.1. We can just use one integer to index the partition, which is $f \times N_x \times N_y + N_x \times iy + ix$, where $N_x \times N_y$ is the size of the grid on each face.

After finding out the partition for the triangle vertex, it is not hard to find out the overlapped partition for the triangle. When we have the partition for the 3 vertices $p_1(f_1, x_1, y_1)$, $p_2(f_2, x_2, y_2)$ and $p_3(f_3, x_3, y_3)$

(a)

S	Partition
S_1	p_1
S_3	p_1
S_1	p_2
S_2	p_2
S_3	p_2
S_1	p_3
S_3	p_3
S_2	p_4
S_3	p_4
S_4	p_4

(b)

C	Partition
C_1	p_1
C_2	p_1
C_2	p_2
C_1	p_3
C_2	p_3
C_2	p_4

(c)

S	C
S_1	C_1
S_1	C_2
S_3	C_1
S_3	C_2
S_1	C_2
S_2	C_2
S_3	C_2
S_1	C_1
S_1	C_2
S_3	C_1
S_3	C_2
S_2	C_2
S_3	C_2
S_4	C_2

(d)

S	C
S_1	C_1
S_1	C_2
S_2	C_2
S_3	C_1
S_3	C_2
S_4	C_2

(e)

S	C
S_1	C_1
S_1	C_2
S_2	C_2
S_3	C_1
S_3	C_2

TABLE 4.2

Searching steps for the example given in Figure 4.2. (a):Search structure of triangle set S. (b)Search structure of triangle set C. (c):Triangle pairs after comparing the search structures of the two sets. (d):Triangle pairs after removing duplicated pairs (e):Triangle pair after checking bounding box

of a triangle, we first check their faces. For most of the cases, the 3 faces are the same, i.e. $f_1 = f_2 = f_3$. So the overlapped partitions of the triangle are the partitions within the axis aligned bounding box of the partitions, i.e. $[f_1, \min(x_1,x_2,x_3) \ldots \max(x_1,x_2,x_3), \min(y_1,y_2,y_3) \ldots \max(y_1,y_2,y_3)]$. For the case that the 3 faces are not all the same, the triangle is located at the boundary of the faces. So we need to include the partitions on each of the different faces. For each face, we treat all three vertices as in this face and compute its local coordinates on this face by referring to Table 4.1, even for the vertex that is not on this face. In this way, the triangle would not miss any intersections with the triangles of different faces. Then, we sort the tuples based on the partition index and generate the search structure for one set of triangles, which is a 2D array showing the relationship between each triangle and its overlapped partition, as in Table 4.1(a).

4.2. Search Triangle Pairs. With the search structure of both triangle sets, we can find the triangle clipping pair candidates (S_i, C_i), where S_i is the triangle index of subject triangles, and C_i is the triangle index of clipping triangles. We do this by comparing these two search structures and pair the triangles that have the same partition index.

To compare the two search structures, we compute two arrays for each search structure. One array O is to store the starting offset of the each of the partition on the search structure array, another array Q is to store the number of the triangles for each of the partitions [18]. The two arrays have entries for all the partitions, even for the partitions that do not have any overlapped triangle, where the number of triangles is set to be 0. In this way, the four arrays O_s, Q_s, and O_c, Q_c, which are the offset arrays and count arrays for the two triangle sets, have the same size($6 \times N_x \times N_y$), which will be easy to compare by GPU parallel programs.

Then, we need to build an array P that shows all the pairs of triangles in the same partition, (S_i, C_i), shown in Table 4.1(c). To know the number of rows for each partition in array P, we can simply do an element-wise multiplication between the two array Q_s and Q_c, and get the array Q_p. To know the starting offset for each partition on array P , we perform a prefix sum on Q_p, and get the array O_p. Then we initiate the array P with the length of $(O_p[last] + Q_p[last])$, which is also the sum of the array Q_p. In the end, we fill

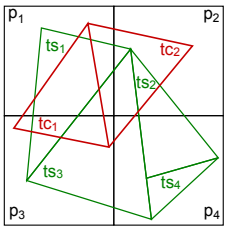


FIG. 4.2. *Example of triangles in 4 partitions.*

the array Q_p in parallel by referring the 7 arrays: O_s, Q_s, O_c, Q_c, O_p and the two search structure arrays A_s and A_c. Generally, we read the triangle index from A_s and A_c, and O_s and O_c tells the location to read, and Q_s and Q_c tells the number of items to read. Then O_p tells the location on P to write triangle pair indices.

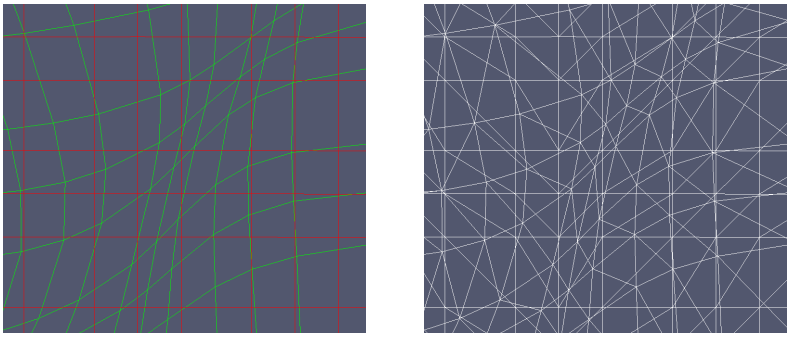
There are some duplicated triangle pairs in the array P, because some triangles may have more than one partitions in common. One triangle may pair with the same triangle in different partitions, including different faces or same face but different 2D grids. In the example, we see duplicated pairs in Table 4.1(c). After removing duplicated pair by performing parallel sort and unique operation, we get the unique pairs in Table 4.1(d).

The size of the triangle pair array P, can be further reduced by removing the pairs that the two triangles' bounding boxes do not overlap. We randomly pick one face from in the faces of the 6 vertices of the two triangles, and calculate the 2D local coordinates on this one face for the 6 vertices. Then we check whether the two triangles 2D bounding boxes have any overlapping. If they are not overlapped, we know the two triangles would not overlap, so we remove the triangle pair from P. In the example of Figure 4.2, we remove one pair (S_4, C_2), because their bounding box does not overlap. In the end, there are only 5 pairs of triangles remains to be the clipping candidates, as shown in Table 4.1(e). Note that it does not matter if the order of the last two operations, removing duplicated pairs and removing overlapping pairs, is reversed.

5. Results and Performance. The generated clipped polygon result is a very dense polygon grid. To verify the correctness of the result, we give a close view of the result in Figure 5.1(b). To compare with the result, we also show the input grids in Figure 5.1(a), where original grid is red and the warped grid is green. Comparing Figure 5.1(a) and Figure 5.1(b), we can see that the edges in the result grid overlap with all the edges of the two input grids, because the edges of clipped polygon come from the edges of subject polygon and clipping polygon. The edges in the result grid that do not overlap with the input grids are the diagonal lines that we added to the input grids in order to divide each quadrilateral into two triangles.

Performance was measured on a machine with Intel Core i7 2600 CPU, 16GB system memory, and nVidia GeForce GTX 560 GPU with 336 cores and 2GB memory. The tests were run with the Linux operating system, Ubuntu 12.04 LTS.

Figure 5.2 compares the performance of our GPU parallel program with a CPU serial program. We tested the program with 8 cubed-sphere grids containing different number of triangles (after dividing quadrilateral), from 134,730 to 1,084,428 triangles. The Y axis represents the total running time of the clipping program except the time of writing results to a file. One reason not including the time of file writing is that the file writing takes much longer time than the computing, especially for a parallel program. Another reason is that in the application of incremental remap, only the integration value over the clipped grid is needed, so there is no need to store the clipped grid itself. The serial program performs a bounding box intersection followed by a serial triangle clipping for all pairs of triangles. From Figure 5.2, we see that both programs



(a) Input grids. red: original grid, green: warped grid

(b) Result grid

FIG. 5.1. *The close view of the inputs and result*

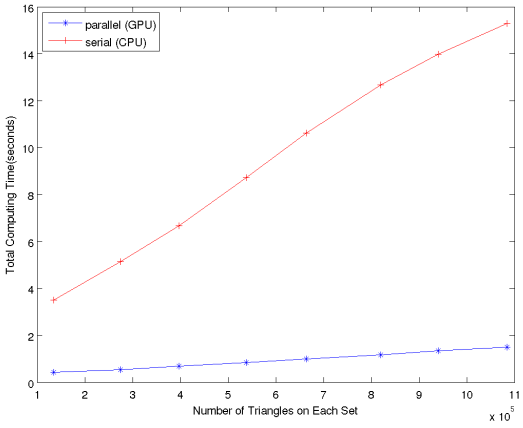


FIG. 5.2. *Total running time of the parallel program and serial program (file writing time is not included)*

scale linearly with the data size. But the parallel program achieves around 10 times speedup comparing to the serial program.

In order to test the effects of our optimization of avoiding branching in the clipping CUDA kernel by sharing instructions among different cases, we measured the running times of the clipping kernel with and without optimization, as shown in Figure 5.3. We see that both running times scales linearly with the data size, but the one with optimization saved around 42% of the time. Thus, our optimization shows its effectiveness.

In order to see the time use of of different steps in the GPU parallel program, we give a stacked bar char in Figure 5.4. Note that the total height of the stacked bar is the same time measurement as the blue curve in Figure 5.2. From Figure 5.4, we see that the searching step takes the largest amount of time. The searching step includes the processes of building the search structure, generating pairs from the search structure, reducing the pairs with bounding box intersection, and removing duplicated pairs. The searching step helps to reduce a great number of triangle pairs that needs to be clipped, which allows the following

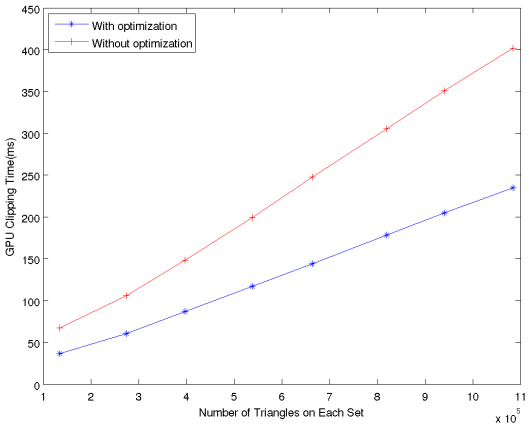


FIG. 5.3. *Running time of the clipping kernel*

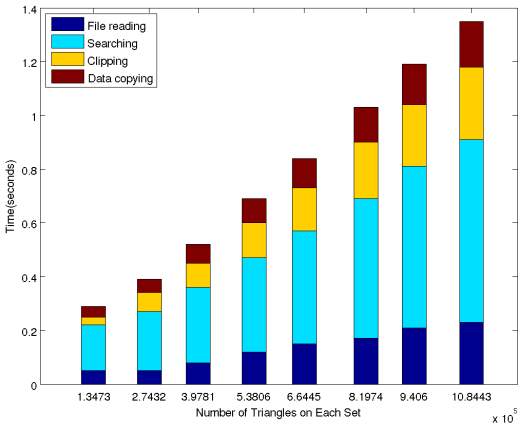


FIG. 5.4. *Time uses in different steps for the parallel program. Dark blue(File reading): time to read grid files; light blue(Searching): time to generate triangle pairs for clipping; yellow(Clipping): time to clip triangles; red(Data copying): time to copy data from host memory to device memory*

clipping step to do less work and use less memory space. Thus, the time spent in searching is very necessary even though it is not part of the clipping algorithm. We can see that the percentage of clipping time over the total running time increases with the data size. So comparing with the searching, the clipping becomes more and more dominating in the entire computing process. The time to copy data from host memory (system memory) to device memory (GPU memory) is small and is included in the searching time. From the figure, the data copying from device memory to host memory takes a relatively long time, because the number of polygons in the output data is much larger than in the input data.

Table 5.1 gives the number of input triangles, number of triangle pairs(a)(b)(c) in different stages of searching, and number of clipped polygons in the result. In a brute-force searching approach, the number

Triangles	Pairs (a)	Pairs (b)	Pairs(c)	Polygons in results
67K	8083K	4058K	1407K	841K
137K	11345K	5181K	2287K	1488K
198K	15408K	6761K	3307K	2162K
269K	21361K	8330K	4451K	2914K
332K	26648K	9728K	5483K	3583K
409K	34618K	11410K	6783K	4417K
470K	40504K	12720K	7786K	5064K
542K	47318K	14249K	8957K	5849K

TABLE 5.1

Number of triangle pairs reduces in the searching stage. pairs(a): number of triangle pairs using our search structure; pairs(b): number of triangle pairs after bounding box intersection test; pairs(c): number of triangle pairs after removing duplicated pairs

of pairs of triangles to do intersection is the square of the number of triangles in one triangle set. From this table, we see that after applying our search structure, the number of pairs (second column) is much smaller than the square of number of triangles (first column). Then, after using bounding box test(b) and removing the duplicated pairs(c), the pair number reduces again, which is only around 0.001% of the number of the original pairs. A pair of triangles generates at most one polygon in the result. From the table, the number of polygons in the result(last column) is over 60% of the number of pairs after the searching step, pair(c). This means most of the triangle pairs generated from the searching stage can intersect with each other. Thus, the searching step or the search structure really helps to reduce the computation and memory demands in the clipping step.

6. Conclusions. In this paper, we present a parallel triangle clipping algorithm on GPU in order to provide a strategy for solving the incremental remapping problem on unstructured grid. This algorithm exhausts the 11 cases of intersection between two triangles and provides a set of operations of generating clipped polygon for each case. The clipping algorithm is further optimized by using instruction masks to allow sharing instructions among different cases in order to minimize the amount of branching on the GPU. We also design a parallel triangle search structure for cubed-sphere grid in order to find triangle pairs before the clipping process. The search structure partitions the space evenly based on the concept of the cubed-sphere grid, which achieves a workload balance of the parallel searching process.

An implementation of our parallel program achieves 10 times speedup over a serial clipping program. The optimization of minimizing branching in the GPU program is also verified to be effective, which saves 42% of the time in the clipping kernel. In one clipping program run, the step of searching triangle pairs takes more time than the clipping step, but it significantly reduces the workload and memory demand for the clipping work.

The concept of our parallel clipping algorithm is also applicable for 2D Cartesian coordinates, not only for spherical coordinates. Our approach can solve incremental remapping on any kinds of 2D grid by first triangulating the grid to triangle grid. This work can be extended to a triangle clipping on a distributed memory system, which allows running incremental remapping over large scale simulation. Besides, the searching performance may be improved with a hierarchical search structure.

7. Acknowledgement. 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 Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

REFERENCES

[1] J. ARVO, ed., *Graphics Gems II*, Academic Press Professional, Inc., San Diego, CA, USA, 1991.

[2] J. M. DENNIS, J. EDWARDS, K. J. EVANS, O. GUBA, P. H. LAURITZEN, A. A. MIRIN, A. ST.-CYR, M. A. TAYLOR, AND P. H. WORLEY, *Cam-se: A scalable spectral element dynamical core for the community atmosphere model.*, IJHPCA, 26 (2012), pp. 74–89.

[3] J. K. DUKOWICZ AND J. R. BAUMGARDNER, *Incremental Remapping as a Transport/Advection Algorithm*, J. Comput. Phys., 160 (2000), pp. 318–335.

[4] G. GREINER AND K. HORMANN, *Efficient clipping of arbitrary polygons*, ACM Trans. Graph., 17 (1998), pp. 71–83.

[5] J. KALOJANOV AND P. SLUSALLEK, *A parallel algorithm for construction of uniform grids*, in Proceedings of the Conference on High Performance Graphics 2009, HPG '09, New York, NY, USA, 2009, ACM, pp. 23–28.

[6] D. H. KIM AND M.-J. KIM, *An extension of polygon clipping to resolve degenerate cases*, Computer-Aided Design & Applications, 3 (2006), p. 447.

[7] Y.-D. LIANG AND B. A. BARSKY, *An analysis and algorithm for polygon clipping*, Commun. ACM, 26 (1983), pp. 868–877.

[8] W. H. LIPSCOMB AND T. D. RINGLER, *An incremental remapping transport scheme on a spherical geodesic grid*, Mon. Wea. Rev., 133 (2005), pp. 2335–2350.

[9] Y. K. LIU, X. Q. WANG, S. Z. BAO, M. GOMBOI, AND B. ALIK, *An algorithm for polygon clipping, and for determining polygon intersections and unions*, Computers & Geosciences, 33 (2007), pp. 589 – 598.

[10] P.-G. MAILLOT, *A new, fast method for 2d polygon clipping: Analysis and software implementation*, ACM Trans. Graph., 11 (1992), pp. 276–290.

[11] W. M. NEWMAN AND R. F. SPROULL, eds., *Principles of Interactive Computer Graphics (2Nd Ed.)*, McGraw-Hill, Inc., New York, NY, USA, 1979.

[12] C. RONCHI, *The cubed sphere: A new method for the solution of partial differential equations in spherical geometry*, Journal of Computational Physics, 124 (1996), pp. 93–114.

[13] R. SADOURNY, *Conservative finite-difference approximations of the primitive equations on quasi-uniform spherical grids*, Monthly Weather Review, 100 (1972), pp. 136–144.

[14] I. E. SUTHERLAND AND G. W. HODGMAN, *Reentrant polygon clipping*, Commun. ACM, 17 (1974), pp. 32–42.

[15] B. R. VATTI, *A generic solution to polygon clipping*, Commun. ACM, 35 (1992), pp. 56–63.

[16] W. WASHINGTON, L. BUJA, AND A. CRAIG, *The computational future for climate and earth system models: On the path to petaflop and beyond*, 367 (2009), pp. 833–846.

[17] K. WEILER AND P. ATHERTON, *Hidden surface removal using polygon area sorting*, SIGGRAPH Comput. Graph., 11 (1977), pp. 214–222.

[18] K. ZHOU, M. GONG, X. HUANG, AND B. GUO, *Data-parallel octrees for surface reconstruction*, Visualization and Computer Graphics, IEEE Transactions on, 17 (2011), pp. 669–681.

COMPARING GPU PARALLEL SEARCH STRUCTURES USING DAX

YUCONG YE* AND KENNETH MORELAND[1]

Abstract. Particle advection is a valuable technique for many scientific simulations. It is the basis of generating streamlines, pathlines, etc. While interpolation of particle properties in the uniform grid is trivial, interpolation in unstructured grid requires first locating the containing cell before interpoloting the cell. As a result, search structures are often used to accelerate this process. Even though search structures in CPU is well established, the introduction of GPGPU allows a parallel approach to search structures. In this paper, we implemented 3 search structures using DAX(Data Analysis at Extreme) [8,9] and benchmarked them according to build time, query time and memory usage.

1. Introduction. While working with unstructured grid, one common operation is locating cells, which means finding the cell within the grid that contains a certain point. Once we are able to locate the cell, we are able to perform many useful tasks, such as cell property interpolation. As a result, we use this specific query task as a benchmark to compare the different search structures.

Search structure is a well established topic in the CPU realm. There are many types of search structures, including uniform grid, quadtree, octree, kd-tree, etc. They provide their own unique advantages and disadvantages. Often hierarchical search structures are preferred in serial programming.

Since parallel architectures are getting more and more popular, people have been investigating into parallel search structures. Most of the papers are constructing the search structures in a shared memory multi CPU environment. Some were written for distributed environment. Only a handful of papers take advantage of the extremely parallel architecture of GPUs. Within this handful of GPU search structure papers, almost all of them are developed for ray tracing. We sampled 3 different implementations of GPU search structures and implemented them using DAX. Modifications are applied to tailor them better for cell locating.

For the benchmark, we are not using unstructured grid as the input. Instead, we are using random spheres as the cells, so in the rest of this paper we will use primitives to describe the cells in a volume, and spheres are primitives.

Parallel search structures have their own characteristics. Therefore they excel in different situations. The strength of uniform grid is simplicity. It is easy to understand and straight forward to implement. Since the construction process only takes 1 phase, the build time for a uniform grid is very short. On the other hand, a uniform grid can be extremely imbalance due to skewed primitive distribution. As a result, the query time of uniform grid is slower in general.

For kd-tree, it provides a mostly balanced search structure. It we are to build a kd-tree on top of a bunch of points, we will get a completly balanced kd-tree because at the end all the leaf buckets in a kd-tree contain the same amount of points. However, since the inputs are primitives, we need to duplicate the primitive IDs as they overlap onto neighbor leaf buckets. As a result, the kd-tree for primitives are not entirely balanced. Since its complicating to build a kd-tree, the build time is expected to be very long. On the other hand, since a kd-tree is mostly balanced, each thread will perform the same amount of work while querying for points. At the same time, there is the overhead of traversing the kd-tree in each thread. It ends up the query time is longer than the other 2 search structures. Since there is no empty bucket in kd-tree, we expect the memory usage of a kd-tree to be very low.

For 2-level grid, it seems to be the middle ground of uniform grid and kd-tree. It provides a limited

*University of California, Davis Computer Science, ye@ucdavis.edu
[1]Sandia National Laboratories, kmorel@sandia.gov

hierarchy yet maintains the simplicity. It is expected to provide relatively low build time, relatively low query time, and relatively low memory usage.

First of all, we are going to describe the specific task that we perform with the 3 search structures. Secondly we will describe the details of each search structure. After all, we will present the benchmark results and compare with our speculations.

Our main contribution is to benchmark different GPU search structures in order to better understand their advantages and disadvantages. We hope that by providing the benchmark results and discussions, we will have a better understanding of when to use which search structure.

2. Related Works. The development for better search structures has never stopped, especially for kd-tree because it is the go to search structure for ray tracing. Within kd-tree, SAH(Surface Area Heuristic) kd-tree has been the most popular among ray tracing applications. In [1, 10], Shevtsov and Choi both described how to construct SAH KD-tree in parallel. While their algorithms are parallel, they both focus on shared memory multicore CPUs, though Choi does claim that one of their algorithms should be applicable to implement in GPU. Nonetheless, both papers do not provide what we need because we need an kd-tree implementation that works in the GPU.

There are not only multicore parallel kd-tree algorithms, Zhou provided a SAH kd-tree construction algorithm in GPU in [12]. Their approach to build SAH kd-tree is that they separate the construction process into two stages, large node stage and small node stage. While in large node stage, Zhou proposed to use spatial median split because SAH approximation is usually incorrect in large nodes. Also, the algorithm parallelizes on triangles in the large node stage, which exploits maximum parallelization. While in the small node stage, Zhou used a SAH cost function to determine splitting planes and the algorithm is parallelized on nodes instead of triangles. Although Zhous approach provides good performance for ray tracing applications, it does not fit cell locating. The purpose of using SAH(Surface Area Heuristic) is to split a node so that the surface areas of the two children, weighted by the number of primitives, are equal, which is very beneficial for ray tracing. However, for cell locating, we want to split a node so that its children contains the equal amount of primitives. As a result, the leaves of the kd-tree will have a more balanced number of primitives, therefore when locating primitives in GPU, each thread will need to go through a smaller set of primitives.

Since we have found no paper that introduces a suitable algorithm for us, we turn to existing libraries. PISTON is a portable cross-platform data-parallel visualization and analysis library developed in Los Alamos National Security, LLC. They have an implementation of kd-tree using thrust, which is a library for CUDA. The kd-tree is built by providing a point cloud and using a median splitting plane. Since a point cannot overlap multiple buckets, every split produces two children that contains equal amount of primitives. More specifically, the algorithm produces a kd-tree with log(n) levels where each leaf contains zero or one point. We used the median splitting function with some modification to better suit our need.

Although SAH kd-tree algorithm is good for ray tracing, one downside would be the build time. Since evaluating the SAH cost function is an expensive operation and there is no way to build the levels of a kd-tree in parallel, the build time of an SAH kd-tree is usually the bottleneck. Kalojanov proposed to use a parallel uniform grid for ray tracing [6]. The algorithm first uses the number of primitives and the volume size to estimate the optimal grid resolution, and then parallelize on primitives to insert each primitive into its corresponding bucket. It provides a faster build time and Kalojanov claimed that up to a certain amount of primitives, the uniform grid outperforms the kd-tree.

Obviously, the uniform grid does not handle non uniform density dataset nicely. Therefore, Kalojanov later on proposed 2-level grid for ray tracing [5]. The idea is to construct a coarser uniform grid as the top level grid, and then for each bucket of the top level grid, evaluate the number of primitives inside and construct a finer grid within each bucket. Kalojanov describes how to build the leaf level grids in parallel by elevating sort.

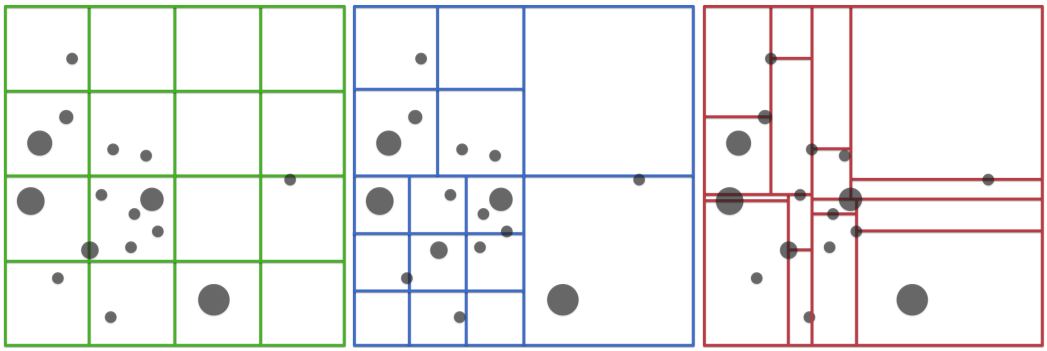


FIG. 3.1. *The 3 search structures on top of the same set of input primitives (circles/spheres in this case).* ***Left/Green****: A 4×4 uniform grid. There are many empty buckets.* ***Mid/Blue****: 2-level grid first uses a coarse top level grid (2×2), and then construct finer leaf level grids (2×2 in top left bucket and 3×3 in bottom left bucket) depend on the density of each top level buckets.* ***Right/Red****: Kd-tree uses medians of the circle/sphere centroids as splitting planes, which the number of primitives in each bucket is more balanced.*

Finally there is a GPU octree construction from Zhou in [11]. It is not building the octree for ray tracing but surface reconstruction. However, the purpose of this octree algorithm is not for locating cells but finding nearest neighbors. As a result, it cannot be used in our benchmark.

3. Methods. After sampling all the papers that implement a GPU parallel search structure, we decide on implementing uniform grid, 2-level grid, and kd-tree. We chose them because uniform grid and kd-tree stands on the two extremes of the scale where each of them have clear advantages and disadvantages. The 2-level grid tries to take the advantages from both ends. We implement the 3 search structures in DAX [8,9] so it can both run in serial and CUDA. Below we will describe in detail the implementation of these 3 search structures. For each search structure we include a simple 2D diagram to help demonstrate the internal structures in Figure 3.1.

3.1. Uniform Grid. The construction of uniform grid (Figure 3.1) is relatively simple. We model the implementation in Kalojanov's paper [6] and translate it into DAX. First, we compute the grid resolution based on the number of primitives and the volume. We use the same equation as Kalojanov used which is given in [2,4]:

$$R_x = d_x \sqrt[3]{\frac{\lambda N}{V}}, \quad R_y = d_y \sqrt[3]{\frac{\lambda N}{V}}, \quad R_z = d_z \sqrt[3]{\frac{\lambda N}{V}}, \tag{3.1}$$

where $\vec{d}$ is the size of the diagonal and V is the volume of the scene's bounding box. N is the number of primitives, and λ is a user-defined constant called grid density. Like Kalojanov, we set the grid density to 5. For more information about choosing the optimal grid density, please refer to [3]. Since we can implicitly represent a uniform grid by using 3 vectors, origin, spacing and extent, we do not need explicit specification for each grid point, which reserves memory.

Second, we need to map all the primitives into its corresponding buckets. We use an array to store the bucket IDs each primitive correspond to. Since there is no way to dynamically allocate memory in GPU, when a primitive overlaps multiple buckets, we cannot increase the array size within the GPU. As a result, we need to calculate and allocate the required memory beforehand. Therefore, two phases are required here:

Parallelize on each primitive, find out how many buckets each primitive overlaps, and then perform a prefix sum operation to obtain the total number of duplicate primitives. Then the total number of duplicate

primitives is used to allocate memory for the bucket IDs array, and the intermediate sums are used as the starting index of each primitive in the bucket IDs array for the next phase. Then the same operation is run again which instead of outputting the count of overlapping primitives, we output the bucket IDs of each primitive into the bucket IDs array indexed by the intermediate sums from the first phase. The way to determine what buckets a primitive overlaps is to use the AABB(Axis-Aligned Bounding Box). While more accurate overlap tests can be used, they are usually more expensive and as a result hurts the overall performance.

Third, after finding out all the bucket IDs, there are two arrays in hand, one is the bucket IDs, the other one is the primitive IDs that duplicates for how many buckets each primitive overlaps. Sort by key is performed using bucket IDs as the key, so the primitive IDs will be ordered so that all primitives in the same bucket are consecutive in the sorted array.

Finally, we create an index array that maps bucket ID to the sorted primitive IDs. More specifically, when we have a bucket ID, we can obtain the primitive count of the bucket and the starting index in the primitive IDs array. With both the primitive count and the starting index, we can obtain all the primitives in the given bucket.

After constructing the uniform grid, we also need a way to locate the cells. Since its a uniform grid, we can compute the containing bucket by using the given point location, and because we have the containing bucket ID, we can easily obtain the primitives in this bucket. Then we can search through this array of primitives to test whether the given point is inside any of the primitives.

3.2. 2-Level Grid. For 2-level grid, we are modeling an implementation given by Kalojanov et al [5]. It uses the uniform grid as the basic implementation. The basic concept of 2-level grid is to first build a coarse top level uniform grid of the volume, and then for each bucket in the top level grid, build another leaf level uniform grid, as shown in Figure 3.1. Therefore, depending on the distribution of the input data, denser region of the volume will have a finer leaf level grid. We will discuss in more detail the configurations of the top level grid and the leaf level grids.

The construction of the top level grid is the same as the uniform grid. However, here we choose a small grid density because we want the top level grid to only provide a base framework and allow the leaf level grids to refine the details. The grid density we choose is the same as [5], which is $\frac{1}{16}$.

For the construction of leaf level grids, we do not want to iterate through the top level buckets and perform uniform grid construction in serial. As presented in [5], the construction of leaf level grids can be performed in parallel. The detailed steps are: After building the top level uniform grid, we obtain the primitive count for each top level bucket. Using the uniform grid resolution equation above, we compute the leaf level grid resolutions parallelize on top level buckets. The leaf level grid density is 1.2, which is the same in [5] too. Here we use an array to store the resolutions of all the leaf level grids, whose length is the same as the number of top level buckets. Also, since we know the resolutions of the leaf level grids, we can compute the bounds and locations of the leaf level buckets and the leaf level bucket counts. Since we have the bounds and locations of all the leaf level buckets, we can map all the primitives to the leaf level buckets the same way as we map primitives into uniform grid. The only difference is that the computation of the leaf level bucket location and bounds requires both the top level grid information and the according leaf level grid information. Finally, we reduce the leaf bucket arrays to start index and primitive count which are used to query the sorted primitives array. The sorted primitives array ensures that the primitives are sorted by the leaf level bucket IDs so that primitives in the same leaf level bucket are consecutive in the sorted primitives array.

Locating cells in 2-level grid is similar to the uniform grid too. First of all, given a point, we locate the top level bucket the point is in. Secondly, leaf level grid resolution of the located bucket can be obtained, and that allows us to locate the point in the leaf level grid. Finally, since we have the leaf level grid, we have the

array of primitives that are in this bucket, so that we can iterate through this array and test for each primitive whether the point is in the primitive.

3.3. Kd-Tree. Kd-tree is different from uniform grid or 2-level grid because its build process is directly related to the input data. The structure of the tree is defined by the input data because we need to choose a splitting plane for each node. There are many ways to choose a splitting plane. Splitting at space median is the simplest case, which is splitting a node to two equal volumes. Splitting at points median computes the median of all the points in the node, and splits there. Therefore, the split results in two children with equal amount of points. Surface area heuristic (SAH) is another more advanced splitting function usually used with ray tracing. In our case, we want the kd-tree to be more balanced according to the number of primitives in the leaves so that when we do parallel primitive locating the work would be more balanced among the threads. As a result, the points median splitting method provides the most benefit to us. However, the input in our kd-tree is primitives instead of points. As a result, we use the centroids of the primitives as the points to construct the kd-tree, as shown in Figure 3.1. Note that it might not be able to build a completely balanced kd-tree, but in our tests, it is balanced enough.

We used the implementation of kd-tree from PISTON [7]. Their implementation uses points as input and split by medians. Also, their kd-tree produces a $\log(n)$ level tree and each leaf bucket will have 0 or 1 primitive. In order to modify it for our case, we used a 2-stage approach to construct the kd-tree.

In the first stage, we compute primitive centroids and use them as input to the PISTON kd-tree. As the output, we get the depth of the tree and the splitting values of all the nodes in an array. Since the PISTON kd-tree is balanced, which means the depth for all the paths are the same, we can index the node in the tree mathematically. For example, given a node with index n, the index of its first child would be $2n+1$.

In the second stage, we map all primitives into the leaves of the kd-tree. First of all, we compute the AABBs of all primitives. Secondly, we count how many leaves each primitive overlaps. And finally, we query the leaf IDs of each primitive. In order to count overlapping leaves and query leaf IDs, each thread bins the corners of the AABB into 1 leaf by querying the splitting values array. As a result, each thread has a loop with number of iterations equal to the depth of the tree.

To query the kd-tree, we perform the same operations as mapping primitives into leaves, but with points. More specifically, each thread bins the point location into a leaf by using the splitting values. And again we need to iterate through the array of primitives in the bucket and determine whether the point is in any primitive.

4. Benchmark Configurations. The major variables to compare are the build time, query time, and memory usage. We want to compare with different input data sizes, different primitive sizes, and different query locations. For the input primitives, we want a dataset that is non-uniformly distributed because uniform grid would be good at handling uniformly distributed datasets. As a result, we randomly distribute the points using polar coordinates. More specifically, we generate random r, θ and ϕ, and then use the following equations to translate into cartesian coordinates:

$$\begin{aligned} x &= r\sin(\theta)\cos(\phi), \\ y &= r\sin(\theta)\sin(\phi), \\ z &= r\cos(\theta). \end{aligned} \tag{4.1}$$

By generating the points this way, the center of the volume will have higher density, as shown in Figure 4.1. Also a fixed or varying radius is assigned to each point, so that they become spheres. For varying radius spheres, we make radius larger when the sphere is farther from the center of the volume because the density is lower when its farther away, as shown in Figure 4.1.

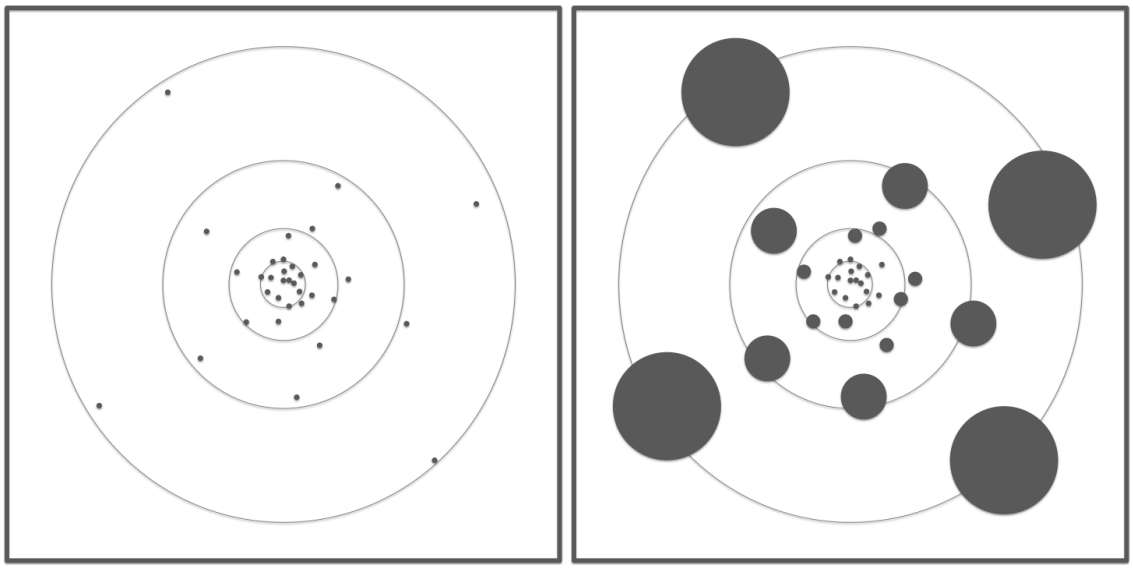


FIG. 4.1. *Fixed radius spheres (Left) and varying radius spheres (Right) as input to search structures. The spheres are more dense in the center of the volume. With varying radii, the radii are linear to the distance from the center of the volume. That is, the spheres are larger when they are farthur away from the center of the volume.*

For querying points, we have two schemes. One scheme is to use the centroids of the original spheres, which guarantees 100% hit rate. The other scheme is to use uniformly random points, which means low hit rate.

As a result, these are the different configurations for the benchmark: different number of spheres (from 100 to 1,000,000), fixed sphere radii vs. varying sphere radii, and querying original sphere centers vs. querying uniformly random points. From these configurations, we believe we will obtain a better understanding of the three search structures.

5. Results. In this section we are going to first describe the testing machine and then we will present the results and discussions. The testing machine has the following configuration:

- Processor: 2 x 2.26 GHz Quad-Core Intel Xeon
- Memory: 6 GB 1066 MHz DDR3 ECC
- Graphics: NVIDIA GeForce GT 120 512 MB

Because of the limited GPU resources, we did not run tests with more than 1 million input spheres.

In the benchmark, we compare the build time, query time, and memory usage of the 3 search structures, which are uniform grid, 2-level grid, and kd-tree, with different set of configurations.

5.1. Build Time. For static scenes, build time would not be an important factor. However for dynamic scenes, basically the search structure is needed to be rebuilt each time the scene changes. As a result, build time is very important for dynamic scenes. From our observation, the build time is mainly constrained by 2 variables. One is the number of input spheres, which is linear to the build time. The other one is the input sphere radii. As shown in Figure 5.1, when we use varying sphere radii, it appears that the search structures require a longer build time. The reason is the spheres overlap more buckets when using varying radius spheres. As a result, we are duplicating more primitive IDs in the sorted primitive IDs. The surprising fact is that the sphere radii do not affect the kd-tree build time as much even though the number of duplicate

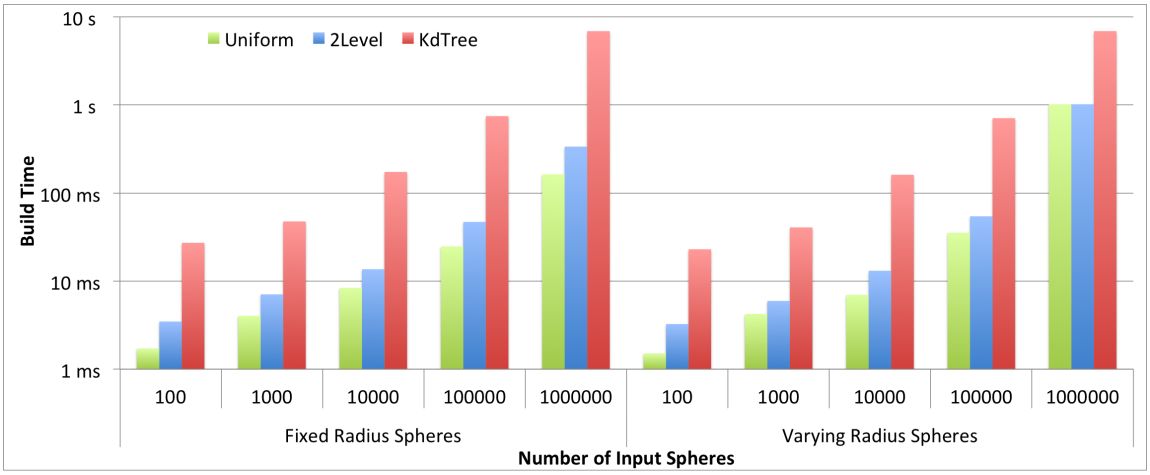


FIG. 5.1. *Build time of uniform grid (green bar), 2-level grid (blue bar), and kd-tree (red bar) when using different number of input spheres and different schemes of sphere radii. Overall the build time is linear to the number of input spheres. The sphere radii have a greater impact on uniform grid and 2-level grid than kd-tree.*

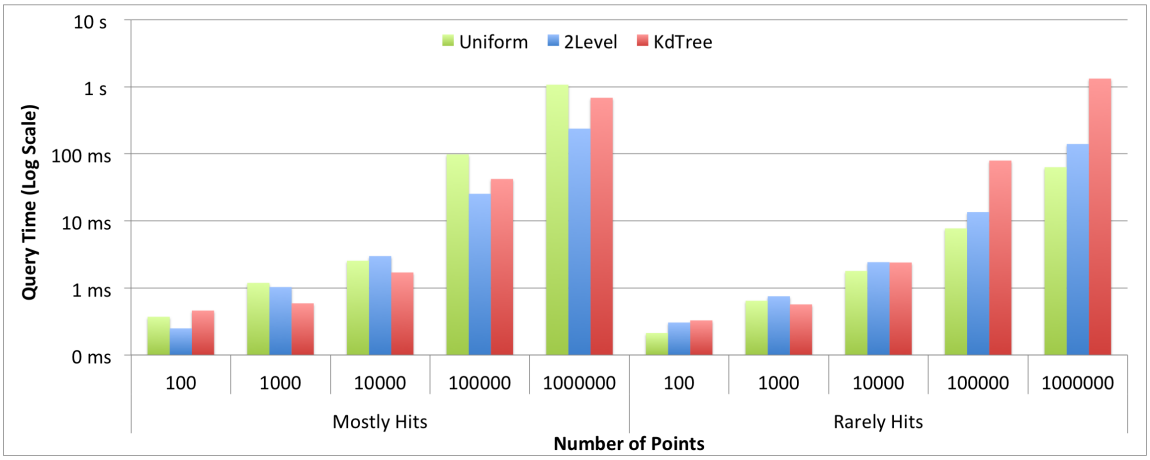


FIG. 5.2. *Query time of uniform grid (green bar), 2-level grid (blue bar), and kd-tree (red bar) when using different number of input spheres and different hit rates. Overall the query time is linear to the number of input spheres. Uniform grid and 2-level grid are able to take advantage of low hit rates while kd-tree performs worse with low hit rates.*

primitives increase. That is because kd-tree uses the PISTON implementation and the PISTON only accepts points as input, so it avoids the duplicating spheres overhead.

5.2. Query Time. Query time is the time to locate primitives in a search structure. We construct test points to query with the same number as input spheres. There are 2 schemes of the test points as described above, original sphere centers and uniformly random points. The major difference of these 2 schemes is the hit rate. On the one hand, the hit rate is 100% when querying original sphere centers. On the other hand, the hit rate should be close to 0% when querying at random coordinates. In Figure 5.2, the uniform grid and the 2-level grid performs better at query time when the query points rarely hit because there are a lot of

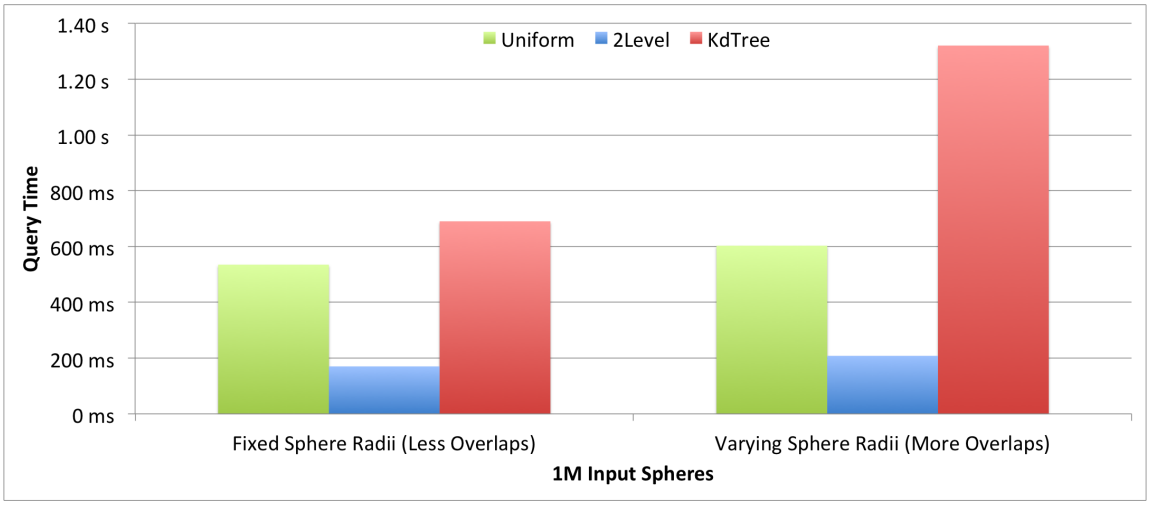


FIG. 5.3. *Query Time vs. Sphere Radii with 1 million input spheres: In both cases, the 2-level grid achieves the fastest query time while the kd-tree is the slowest. The query time is slower with varying radius spheres, where kd-tree almost doubles the query time when switching to varying radius spheres.*

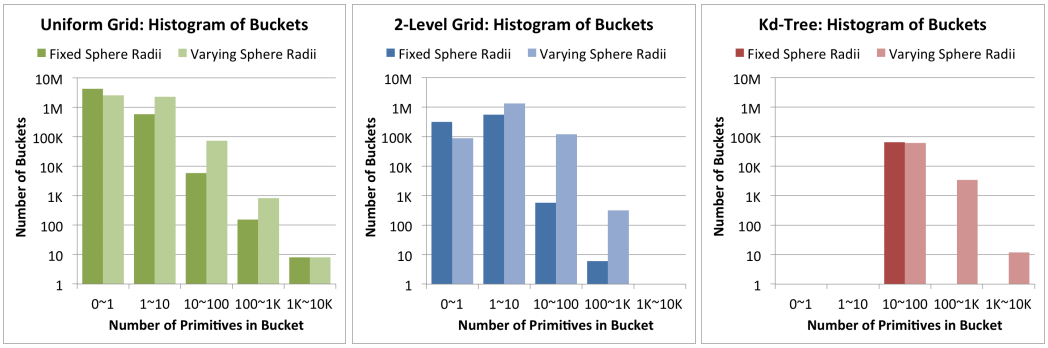


FIG. 5.4. *Histogram of the distribution of primitives in buckets for each search structure. All of the figures above are in log-log scale. The buckets in Uniform Grid (Left/Green) have extremely imbalanced primitive counts, where there are many empty buckets and there are a few buckets that have large amount of primitives. 2-Level Grid (Mid/Blue) produces a relatively balanced histogram. Kd-Tree (Right/Red) is very balanced with fixed radius spheres where the number of spheres in all buckets are in the range of 10 100. However, it produces even more imbalanced histogram than uniform grid with varying radius spheres.*

buckets in these two search structures that are empty. Therefore many of the threads finish instantly. On the other hand, kd-tree is unable to take advantage of misses because there are no empty buckets in kd-tree and furthermore each thread in kd-tree has to go through the tree traversal loop.

Other than hit rates, input sphere radii also have a large impact on query time. As shown in Figure 5.3, for 1 million input spheres, kd-tree almost doubles the query time when we switch to use varying radius input spheres while the query time for uniform grid and 2-level grid only increase a little bit. When we use varying radii, the spheres are likely to overlap more buckets in the search structures, especially in kd-tree. As shown in Figure 5.4, for uniform grid and 2-level grid, when we switch from fixed radius spheres to varying radius spheres, there are more buckets with more primitives, but not too much. However for kd-tree, it is really

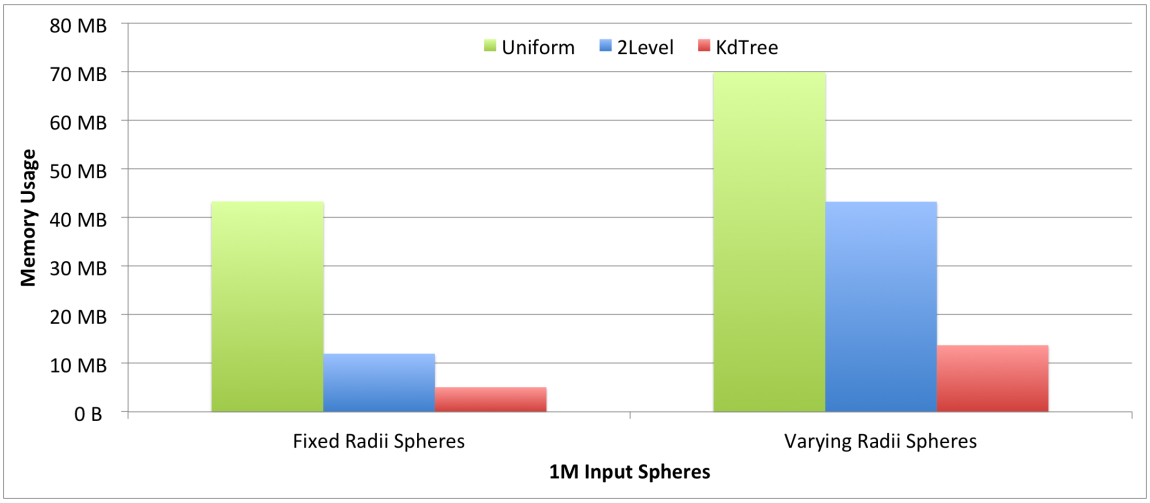


FIG. 5.5. *Memory usage of uniform grid (green bar), 2-level grid (blue bar), and kd-tree (red bar) when using different schemes of sphere radii. The search structures ordered from low to high memory usage are kd-tree, 2-level grid, and uniform grid. The sphere radii affect memeory usage a lot and it has slightly less impact on uniform grid.*

balanced with fixed radius spheres where all the buckets contain no more than 100 spheres. After we switch to varying radius spheres, kd-tree has even more buckets than the uniform grid in the 1K 10K primitives range. As a result, the kd-tree runs slower on query time when we are using varying input spheres.

5.3. Memory Usage. In Figure 5.5, we can observe that kd-tree uses the least memory, almost 8-10 times less than the memory usage of uniform grid. The 2-level grid is also faster than uniform grid, but still 2-3 times slower than kd-tree.

The behavior of memory ranking from kd-tree to uniform grid is expected. Recall that all 3 search structures require a sorted primitive IDs array, which approximately uses the same amount of memory. As a result, the memory usage depends on the amount of buckets in each search structure. Since uniform grid is a flat grid with fine resolution, it has the most number of buckets. Therefore, the uniform grid takes the most memory. 2-level grid provides a limited hierarchy. It uses a coarse top level grid and finer detailed leaf level grids. As a result, it reduces the number of buckets, so it takes less memory. For kd-tree, our approach guarantees there are only $\log(n)$ leaf buckets, which is significantly less than the other 2 search structures. As a result, kd-tree uses the least memory.

One important factor of memory usage is the input sphere radii. As in Figure 5.5, when using varying radius spheres as input, all the search structures use more memory approximately by a factor of 2. When we use varying radius spheres as input, the spheres overlap significantly more buckets in the search structures. As a result, it results in a larger sorted primitive IDs array, therefore uses more memory. Note that the overlap effect appears more in the 2-level grid and kd-tree because they both have a finer resolution at the leaf level, as shown in Figure 5.4. As a result, sphere radii have a larger impact on 2-level grid and kd-tree.

6. Conclusions. While uniform grid has low build time, short query time and high memory usage, kd-tree is like the complete opposite. It provides long build time, long query time and low memory usage. 2-level grid seems to be the middle ground of uniform grid and kd-tree. It provides a limited hierarchy yet maintains the simplicity. As a result, it depends on the distribution of the input data and the type of operations to determine which search structure to use. If the input data is uniformly distributed in the volume, then we

can use a uniform grid as the search structure. If memory is the constraint, then kd-tree provides a good solution. For a more balanced approach, we can always choose 2-level grid.

7. Future Work. In uniform grid, we can specify a grid density to determine the resolution of the grid, and in 2-level grid, we can specify both top level grid density and leaf level grid density. The values we have now are coming from both [5,6]. Since they were designed for ray tracing, the grid density values might not fit our case completely. As a result, we will try different grid density values and compare how they perform with primitive locating.

In the construction of kd-tree, we use the PISTONs implementation. The main reason is that the implementation of kd-tree requires a lot of segmented operations, such as segmented scan, segmented sort, etc. Since we are using DAX, and DAX does not yet provide these segmented operations, we are unable to have the full implementation in DAX. Also, the PISTON kd-tree is built for points and we modified it to use primitive centroids, which is not the perfect condition. Therefore, we should be able to better utilize DAX and have a better implementation of the kd-tree.

8. Acknowledgement. This work was supported in part by the DOE Office of Science, Advanced Scientific Computing Research, under award number 10-014707, program manager Lucy Nowell.

This work was supported in part by the Director, Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy under Contract No. 12-015215, through the Scientific Discovery through Advanced Computing (SciDAC) Institute of Scalable Data Management, Analysis and Visualization.

Sandia National Laboratories is a multi-program laboratory operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration.

REFERENCES

[1] B. CHOI, R. KOMURAVELLI, V. LU, H. SUNG, R. L. BOCCHINO, S. V. ADVE, AND J. C. HART, *Parallel sah k-d tree construction*, in Proceedings of the Conference on High Performance Graphics, HPG '10, Aire-la-Ville, Switzerland, Switzerland, 2010, Eurographics Association, pp. 77–86.

[2] O. DEVILLERS, *Methodes doptimisation du trac de rayons*, PhD thesis, Universit de Paris-Sud, 1988.

[3] T. IZE, P. SHIRLEY, AND S. PARKER, *Grid creation strategies for efficient ray tracing*, in Interactive Ray Tracing, 2007. RT '07. IEEE Symposium on, 2007, pp. 27–32.

[4] D. JEVANS AND B. WYVILL, *Adaptive voxel subdivision for ray tracing*, in Proceedings of Graphics Interface, vol. 89, 1989, pp. 164–172.

[5] J. KALOJANOV, M. BILLETER, AND P. SLUSALLEK, *Two-level grids for ray tracing on gpus*, Computer Graphics Forum, 30 (2011), pp. 307–314.

[6] J. KALOJANOV AND P. SLUSALLEK, *A parallel algorithm for construction of uniform grids*, in Proceedings of the Conference on High Performance Graphics 2009, HPG '09, New York, NY, USA, 2009, ACM, pp. 23–28.

[7] J. A. LI-TA LO, CHRISTOPHER SEWELL, *Piston: A portable cross-platform framework for data-parallel visualization operators*, Tech. Rep. LA-UR-12-10227, Los Alamos National Laboratory, 2012.

[8] K. MORELAND, U. AYACHIT, B. GEVECI, AND K.-L. MA, *Dax: Data analysis at extreme*.

[9] ———, *Dax toolkit: A proposed framework for data analysis and visualization at extreme scale*, in Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on, IEEE, 2011, pp. 97–104.

[10] M. SHEVTSOV, A. SOUPIKOV, AND A. KAPUSTIN, *Highly parallel fast kd-tree construction for interactive ray tracing of dynamic scenes*, in Computer Graphics Forum, vol. 26, Wiley Online Library, 2007, pp. 395–404.

[11] K. ZHOU, M. GONG, X. HUANG, AND B. GUO, *Data-parallel octrees for surface reconstruction*, Visualization and Computer Graphics, IEEE Transactions on, 17 (2011), pp. 669–681.

[12] K. ZHOU, Q. HOU, R. WANG, AND B. GUO, *Real-time kd-tree construction on graphics hardware*, in ACM SIGGRAPH Asia 2008 papers, SIGGRAPH Asia '08, New York, NY, USA, 2008, ACM, pp. 126:1–126:11.

PARALLEL DETECTION OF STRONGLY CONNECTED COMPONENTS IN MASSIVE GRAPHS: THE MULTISTEP METHOD

GEORGE M. SLOTA* AND SIVASANKARAN RAJAMANICKAM[1]

Abstract.
Finding the strongly connected components (SCCs) of a directed graph is a fundamental problem used in many fields. For example, it is part of computing the block triangular form of a sparse matrix. Tarjan's algorithm is an efficient serial algorithm, but relies on depth-first search which is hard to parallelize. Several parallel algorithms have been proposed, but show poor load balance and slow performance on many real world graphs. This paper introduces the Multistep Method, which combines several parallel SCC finding algorithms to deliver performance results with up to 60× speedup over the serial Tarjan's algorithm, while being capable of fully decomposing a 1.8 billion edge graph in under two seconds. As breadth-first search is a key component for finding SCCs, we also demonstrate a fast implementation which delivers graph traversal speeds up to the equivalent of 18 billion edge traversals per second on a single 16 core shared-memory machine.

1. Introduction. The strongly connected components (SCCs) of a directed graph are the maximal subgraphs where every vertex within the subgraph can reach and can be reached by every other vertex within the subgraph. The decomposition of a directed graph into its strongly connected components is a useful analytic tool in many applications like social network analysis [17], compiler design, radiation transport solvers [16], and computing the block triangular form for linear solvers and preconditioners [19, 21].

The optimal sequential strongly connected component algorithm is based on linear time Tarjan's algorithm [20]. Tarjan's algorithm depends on a depth first search of the graph and is not an easy algorithm to parallelize with good scalability. There are task-based algorithms to compute the SCC [16, 22] for distributed memory computers and coloring based algorithms for both distributed memory and accelerator based approaches [3, 18].

The increase in concurrency available within a single compute-node allows us to explore shared-memory based algorithms for different problems. In addition, the increase in memory footprint allows us to analyze very large graphs that would have been impossible a few years ago using shared-memory-based algorithms. With a difficult-to-parallelize algorithm in distributed memory, a task-based algorithm that can exploit existing task-based shared memory programming models, and a coloring algorithm which depends on fine-grained atomics, the problem of computing the strongly connected components is one of the ideal problem to stress test shared-memory implementations.

Our contributions in this paper include an optimized implementation of existing parallel algorithms using shared-memory programming models, a new parallel algorithm for computing the SCCs for real world graphs that combines the benefits of previous algorithms while at the same time reducing and is up to ∼11× faster on average with 16 cores, a fast atomic and lock-free breadth-first search (BFS) implementation that can reach the equivalent of over 18 billion edge traversals per second (GTEPS) in a single shared-memory node, and an experimental evaluation of several different parallel algorithms.

The rest of the paper is organized as follows. Section 2 describes the existing serial and parallel algorithms for computing SCCs. Section 3 describes our multistep method to compute SCCs. Sections 4 cover the implementation details in shared-memory architectures and Sections 5 and 6 demonstrate the benefits our new algorithms for various real-world graphs.

2. Background. There are several existing serial and parallel algorithms that have been used to determine the SCCs of a graph. This section will give a general overview of several of the most widely used ones, including Tarjan's and Kosaraju's algorithms, the Forward-Backward algorithm, Coloring, and Trimming.

*The Pennsylvania State University, gms5016@psu.edu
[1]Sandia National Laboratories, srajaman@sandia.gov

2.1. Serial Algorithms. The primary serial algorithms used for determining strongly connected components within a graph are Tarjan's [20] and Kosaraju's [1] algorithms. Tarjan's algorithm uses a recursive depth first search (DFS) to form a search tree of explored vertices. The roots of the subtrees of the search tree form roots of strongly connected components. Kosaraju's algorithm performs two passes of the graph. It initially performs a DFS, placing each vertex onto a stack after it has been fully explored. After all vertices have been placed onto the stack, a vertex is popped from the stack and a DFS or BFS search is performed on the transpose of the graph. All vertices that can be reached by this vertex (that have not already been explored a second time) form an SCC.

Although the initial DFS in Kosaraju's algorithm is very difficult to parallelize, the second searches of the transpose can be parallelized quite trivially, due to the fact that the second search needs only to determine reachability, rather than any specific ordering. As Tarjan's algorithm only performs a single search of the entire graph, its work is $O(n+m)$, where n is the number of vertices and m is the number of edges in the graph, and can therefore be considered asymptotically optimal.

2.2. Parallel Algorithms. As the above described serially algorithms are non-trivial to parallelize in a scalable way, different algorithms have been proposed that allow parallel computation of strongly connected components. Two of the most widely used are the recursive Divide-and-Conquer Strong Components algorithm [22] (which has become more widely known as the Forward-Backward algorithm) and the iterative coloring algorithm [18]. We will also discuss the trimming procedure which can be used to quickly eliminate trivial SCCs and decrease the overall amount of work necessary for the primary algorithm. The trimming procedure can be used as a pre-processing step for any of the algorithms described here.

2.2.1. Forward-Backward Algorithm. The Forward-Backward (FW-BW) algorithm is given in Algorithm 9 and can be described as follows: Given a graph G, a single *pivot* vertex v is selected. This can be done either randomly or through simple heuristics. A BFS/DFS search is conducted starting from this vertex to determine all vertices which are reachable (forward sweep). These vertices form the *descendants* set (D). Another BFS/DFS search is performed from v but on the transpose graph. This search (backward sweep) will find the set (P) of all vertices than can reach v called the *predecessors*.

Algorithm 9 Forward-Backward Algorithm

```
1:  procedure FWBW(G)
2:      if G = ∅ then
3:          return ∅
4:      end if
5:      select pivot v
6:      D ← DESC(G, v)
7:      P ← PRED(G, v)
8:      R ← (G \ (P ∪ D)
9:      S ← (D ∩ P)
10:     S ← S ∪ FWBW((D \ S))
11:     S ← S ∪ FWBW((P \ S))
12:     S ← S ∪ FWBW(R)
13:     return S
14: end procedure
```

The intersection of these two sets form an SCC ($S = D \cap P$) that has the pivot v in it. If we remove all vertices in S from the graph, we can have up to three remaining distinct sets: $(D \setminus S), (P \setminus S)$, and *remainder*

R, which is the set of vertices that we have not explored during either search from v. The FW-BW algorithm can then be recursively called on each of these three sets.

Possible parallelism exists for this algorithm on two levels. Primarily, as each of the three sets are distinct, they can be further explored in parallel. In addition, each of the forward and backward searches can be trivially parallelized using a standard parallel BFS or even DFS (since we don't care about ordering, only reachability).

2.2.2. Coloring. The coloring algorithm is given in Algorithm 10. This algorithm is similar to FW-BW in that it proceeds in forward and backwards passes. However, the approach is quite different, as it generates multiple pivots, or more commonly termed, *roots*, during the forward phase and only looks at a subset of G for each pivot on the backwards phase.

Algorithm 10 Coloring Algorithm

```
1:  procedure COLORING(G)
2:      while G ≠ ∅ do
3:          initialize colors(v_id) = v_id
4:          while not fully colored do
5:              for all v ∈ G do
6:                  for all n ∈ N(v) do
7:                      if colors(v) > colors(n) then
8:                          colors(n) ← colors(v)
9:                      end if
10:                 end for
11:             end for
12:         end while
13:         for all unique c ∈ colors do
14:             SCC(c_v) ← PRED(G(c_v), c)
15:             G ← (G \ SCC(c_v))
16:         end for
17:     end while
18: end procedure
```

Given a graph G, the algorithm starts by assigning a set of unique (numeric) *colors* to all the vertices, most easily as the vertex identifiers v_{id} for all $v \in G$. These colors are then propagated outwardly from each vertex in the graph. If a vertex v has any neighbors $n \in N(v)$, where N is the neighbor list, with a color lower than it's own, the neighbors' colors are updated to that of the vertex. This process continues until no more vertices change their color.

We have now effectively partitioned the graph into distinct sets with separate colors c. As we started with v_{id} as our colors, for each distinct c, there is a unique vertex c_v with that identifier. We consider c_v as the root of a new SCC, $SCC(c_v)$. The SCC is then all vertices that can be reached backward from c_v that are also colored with the same color. We remove $SCC(c_v)$ from G, find the rest of the SCCs for all c in the current iteration, and then proceed to the next iteration and continue until G is empty.

Parallelizing this algorithm is easy, as both the forward coloring step across all v and the backward SCC step across all c_v can be parallelized quite trivially.

2.2.3. Trimming Procedure. The trimming procedure was initially proposed as an extension to FW-BW [16], to remove all trivial strongly connected components. The procedure is quite simple as shown in

Algorithm 11: all vertices that have an in degree or out degree of zero (without self-loops) are removed. This can be performed iteratively or recursively as well, as removing a vertex will change the effective degrees of it's neighbors. We call performing a single iteration of trimming as *simple* trimming and performing trimming iteratively as *complete* trimming. This procedure is very effective in improving the performance of the FW-BW algorithm, but can be beneficial to use before running other algorithms, as well.

Algorithm 11 Trimming Algorithm

```
1:  procedure TRIM(G)
2:      while not fully trimmed do
3:          for all v ∈ G do
4:              if degree_in(v) = 0 then
5:                  G ← (G \ v)
6:              else if degree_out(v) = 0 then
7:                  G ← (G \ v)
8:              end if
9:          end for
10:     end while
11: end procedure
```

2.3. Other Work. There has been other more recent work aimed at further improving upon the preceding algorithms, as well as developing newer algorithms to further decompose the graph and/or improve performance and scalability.

One example is the OBF algorithm of Barnat et al. [4], which, like coloring, aims at every iteration to further decompose the graph into multiple distinct components each containing a single SCCs. The OBF decomposition step can be performed much quicker than coloring, however. Barnat et al. [3] further implemented the OBF algorithm as well as FW-BW and coloring on the Nvidia CUDA platform, demonstrating considerable speedup over equivalent CPU implementations.

More recently, Hong et al. [11] demonstrated several improvements to the FW-BW algorithm and trimming procedure through expanding trimming to find both 1-vertex and 2-vertex SCCs, further decomposing the graph after the first SCC is found by partitioning based on connected components, and implementing a dual-level task-based queue for the recursive step of FW-BW to improve times and reduce overhead for task-based parallelism.

3. The Multistep Method. We will now introduce our algorithm for graph SCC decomposition, the Multistep method. The reason for this name comes from the fact that it is a combination of the previously described algorithms stepped through in a certain order. This section will give our justifications for developing the algorithm in this way, as well as provide detail into our algorithm's specifics.

3.1. Observations. The FW-BW algorithm can be quite efficient if a graph has relatively small number of large and equally-sized SCCs, as the leftover partitions in each step would on average result in similar amounts of parallel work. The FW and BW searches could also be efficiently parallelized in this instance.

However, the structure of most real world graphs is very different. From observations, most real-world graphs have one giant SCC containing a large fraction of the total vertices, and many many small SCCs and often disconnected SCCs that remain once the large SCC is removed [17]. Running a naïve implementation of FW-BW would result in a large work imbalance after the initial SCC is removed, where the partitioning of the remaining sets would be heavily dominated by the remainder set. Additionally, using a naïve task-based

parallelism model would add considerable overhead as each new task might only be finding an SCC of a few vertices in size before completing. In general, the size of the recursive tree and therefore the overall runtime of the FW-BW algorithm is dominated by the total number of SCCs that are in the initial graph.

Conversely, the coloring algorithm is quite efficient when the graph contains a large number of small and disconnected components, as the runtime of each step is dominated by the diameter of the largest connected component in the graph, or the number of steps needed to do the full coloring. This also leads to very poor performance on real-world graphs, as the time for each iteration can be very large while the largest SCCs remain, and there is no guarantee that these SCCs will be removed in any of the first few iterations.

It is also important to note that below a certain threshold of the number of vertices in a graph, the general parallel overhead results in worse performance when compared to simply using Tarjan's or Kosaraju's serial algorithm.

3.2. Description of Method. Based on the above observations, we therefore have developed what we term as the Multistep Method. This method aims at maximizing the advantages and minimizing the drawbacks of trimming, FW-BW, coloring, and a serial algorithm by applying them in sequence to decompose large real-world graphs into their strongly-connected components.

An overview of the algorithm is given in Algorithm 12. There are four primary phases of the algorithm. The first phase is trimming. We choose to do only a single iteration of trimming, as experiments have shown that the vertices trimmed in the second or subsequent iterations can be more efficiently handled in the coloring or serial phases.

Algorithm 12 Multistep Algorithm

1: **procedure** MULTISTEP(G)
2: $Trim_{simple}(G)$
3: **select pivot** $v = max(v_{dg}), where\ v_{dg} = d_{in}(v) * d_{out}(v)$
4: $FWBW_{SCC}(G, v)$
5: **while** $NumVerts(G) > V_{cutoff}$ **do**
6: $ColoringStep(G)$
7: **end while**
8: $Tarjan(G)$
9: **end procedure**

In the second phase, we select an initial pivot vertex as the vertex in the graph that has the largest multiplicative sum of its in degree and out degree. This is an attempt to increase the chances that our initial pivot is contained within the largest SCC. Although there is no guarantee to ensure that, in practice with real world graphs it acts as a very good heuristic.

With the chosen pivot we do one iteration of modified FW-BW algorithm. Our changes to the FW-BW iteration avoid considerable work during this phase by not computing the three sets D, P and R and leaving all the vertices that are not part of the one SCC we computed to the next step.

As we do not partition the vertices into three sets, but compute just one SCC, we do a full forward search from the pivot and on the backward search explore vertices that were already marked as explored on the forward search. While doing a BFS for the backward search, if we encounter a vertex not already marked, we avoid adding it to the next level queue.

For a simple proof as to why this will work, assume by contradiction that a predecessor vertex p_i we find during the backward phase that was not marked during the forward phase has a predecessor p_j that was marked during the forward phase. This cannot happen as if the predecessor was previously marked, then

the original predecessor p_i, being a descendant of p_j, would therefore have been marked as well. Since we know that, in order to be in the SCC, a vertex must be marked as both a descendant and predecessor, then we can safely ignore these p_i. For certain graphs, this can result in a considerably shorted search during the backward phase.

At the end of the FW-BW step, we simply take the remaining vertices and pass them all off to coloring. We run coloring until the number of vertices remaining crosses below a certain threshold (determined experimentally), and then pass off the still remaining vertices to the final step, which is just Tarjan's serial algorithm.

4. Implementation Details and Optimizations. This section will provide a bit more detail into some of the implementation specifics. All code was written in C++ using OpenMP for shared-memory parallelization. We achieve most of our performance by using thread-owned queues, boolean arrays, and various techniques to avoid work, as well as avoiding atomic or locking operations whenever possible by simply accounting for any possible race condition.

4.1. Graph Representation and other Data Structures. We use a standard compressed sparse row representation for our graph. Since most of our algorithms require both in and out edges, $O(2m+2n)$ storage is required for both the in and out edge and degree lists. In order to avoid having to explicitly modify the graph structure, we keep a boolean array to signify whether a given vertex is still valid or not and unmark the vertex when the SCC it is contained in is completely found. We also maintain a current running list of the numeric identifiers of which vertices are still active, to greatly speed up loops that look across all vertices remaining in the graph.

In order to track which SCC a vertex in, another integer array is maintained. At the start of finding a new SCC (FW-BW, each root in each coloring step, each root in Tarjan's) a running atomically-updated count value is incremented and noted. If a vertex is found to be in that certain SCC this noted value is assigned to it. All trimmed vertices are assigned the same value for simplicity.

4.2. Trim Step. We consider two different approaches for trimming. For simple trimming, we only need to look at the degree values as initially set when the graph was created. Therefore, we just need to do a single pass through all vertices, retrieve their in/out degrees, and flip their valid boolean if either is zero.

Complete trimming is a bit more complex. To greatly speed up processing during complete trimming, we create current and future queues and an additional boolean array of values to signify for each vertex if they are currently placed in the future queue. We place all vertices in the current queue to begin with.

We then determine the in and out degrees for all vertices contained in the current queue. If a vertex has an effective in or out degree of zero, then it is marked as no longer valid. Additionally, any valid vertices that the removed vertex was pointing to or had pointing at it is then placed in the future queue and marked as such. After the current queue is empty, the queues are swapped, the marks reset, and this process is repeated for as many iterations as necessary. The queues are used to avoid having to look through all vertices at each iteration, as it has been observed that the long *tendrils* [6] of vertices in lots of real world graphs tend to result in long tails of iterations where only a few vertices are removed at a time. The marking is done to prevent a vertex from being placed in the future queue multiple times. To avoid the synchronization overhead that would be required with a parallel queue, we maintain separate queues for each thread and combine them into the next level queue at the end of each step. We perform a similar operation with our BFS, as we will explain later.

As we will later show, although complete trimming is easily paralellizable and can be quite fast, with the queues and marking being done very similarly to how we will soon describe our BFS, we find simple trimming to generally perform better overall. It is observed that a single iteration will remove the vast majority of vertices than *can* be removed, it does not require the explicit calculation or tracking of changing

degrees at each iteration, and does not need queues or other such structures to maintain. Simply passing off the vertices not trimmed in the first iteration to be handled by coloring or the serial algorithm gave us better performance for all tested graphs.

4.3. Large SCC Search Step. For most graphs, determining the initial large SCC takes a majority of the overall runtime. Because this initial search is essentially just two breadth-first searches, we have developed and attempted to optimize for performance a BFS using techniques developed recently from a number of sources.

4.3.1. Parallel BFS. An overview of the BFS used in our Multistep approach is given in Algorithm 13. We will now describe some of the optimizations and design choices we implemented.

Algorithm 13 Multistep BFS

```
1:  procedure MULTISTEPBFS(G, p)
2:      queue ← p
3:      do_hybrid ← false
4:      while queue ≠ ∅ do
5:          if hybrid = false then
6:              for all v ∈ queue do in parallel
7:                  for n ∈ N_out(v) do
8:                      if visited(n) = false then
9:                          visited(n) ← true
10:                         thread_queue ← n
11:                     end if
12:                 end for
13:             end for
14:         else
15:             for all v ∈ G do in parallel
16:                 if visited(v) = false then
17:                     for all n ∈ N_in(v) do
18:                         if visited(n) = true then
19:                             visited(v) ← true
20:                             thread_queue ← v
21:                             break
22:                         end if
23:                     end for
24:                 end if
25:             end for
26:         end if
27:         Synchronize
28:         Single thread does
29:             hybrid ← EvaluateHybridSwitch()
30:             if hybrid = false then
31:                 for all thread_queue do
32:                     for all v ∈ thread_queue do
33:                         queue ← v
34:                     end for
35:                 end for
36:             end if
37:         end single thread
38:         Synchronize
39:     end while
40: end procedure
```

A typical BFS optimization is to use a bitmap of length n to signify whether or not a vertex has already been visited, and to avoid further exploring the vertex if it has been. A bit vector is able to fit completely in the last level cache on modern server-grade CPUs for graphs of up to tens of millions of vertices. It is assumed that by staying in cache, a quick boolean check is possible and accesses to main memory are minimized.

However, our experimentation has demonstrated that a boolean array actually outperforms a bitmap for our test environment. The likely reason for this is two-fold. Firstly, in order to calculate the address for a specific bit, at least an integer division, remainder operation, and bit shift is required. Additionally, since the CPU on our testbed only guarantees atomic read/writes starting only at the byte level [12, s. 8.1.1], either explicit atomic operations are needed or a more complex bit read/write function is required [9]. By using a byte-addressed boolean array and avoiding explicit locks, we see considerably faster runtimes. Although avoiding explicit locks might result in extra work (two threads see a vertex as unexplored, set the same bool to the same value, and put the vertex in the next level queue twice), experimentation has shown this to be of minimal concern.

As mentioned previously, we avoid additional locks and atomic operations by giving each threads its own next level queue. At the end of each level, the vertices from each thread queue are collected and placed in the frontier for the next level by a single thread. Although this is necessarily done is serial, experimentation has shown it to be very fast, taking on the order of milliseconds even for millions of nodes, and greatly outperforming a shared queue with locks. This step can also be skipped when we are going to run the hybrid bottom-up BFS, which will be discussed next.

4.3.2. Hybrid BFS Approach. A hybrid bottom-up approach to the BFS was recently introduced by Beamer et al. [5]. They noted that at certain levels of a BFS in real world graphs (small world, scale free), it is actually more efficient to simply look in the reverse direction. Instead of all vertices currently on the frontier looking at all their children, all unvisited vertices simply attempt to find their parent on the frontier. It is not even necessary to check if the parent is explicitly on the frontier, but only if the parent has already been marked as visited (the child would have already been discovered if the parent's level is one previous to that of the current frontier).

Beamer et al. found that this approach will vastly decrease the total number of edge examinations needed during the BFS, improving search times by over $3\times$. Upon implementing their approach as described with parameters $(\alpha = 15, \beta = 25)$, we noticed considerable speedup as well. A different design choice we had to implement was to maintain the thread queues while we are currently running the bottom-up hybrid as opposed to explicitly rebuilding the queue from scratch when we switch the hybrid off. This is due to the fact that we do not maintain any explicit BFS tree as we only require the visited array to determine the SCC, so we have no ability to track BFS level on a per-vertex basis.

4.3.3. Graph Partitioning Across Sockets. A final optimization we implemented was a per-socket graph partitioning and exploration scheme similar to the ones described in Agarwal et al. [2] and Chhugani et al. [9]. In this, we attempt to reduce a majority of cross-socket data travel by partitioning the graph across the memory closest to separate sockets, and only loading edges of a vertex for a thread running on the CPU closest to the socket by maintaining separate per-socket queues. We use the hwloc library [7] to enforce locality. At the end of each level, vertices discovered by a socket not containing its edges are passed over to the queue for the socket that has them. When we hit a level where we decide to switch to the bottom-up approach, the vertices that are examined by each socket are simply the ones that is contained in that partition.

Another approach we attempted for our two-socket Compton testbed was by placing all in degree data on one socket and all out degree data on the other, and then running both the forward and backward steps concurrently. We then attempted to include the bottom-up approach in this scheme, by enforcing level

synchronization between the two searches and moving the boolean visited arrays across socket when the switch occurred. A variety of switch heuristics were explored, without much effect.

Although these partitioning approaches improved parallel scaling, it was only in a limited number of instances that actual runtimes improved due to the additional overhead. A wide variety of parameters and optimizations were explored, including reverting to actual bitmaps, partitioning based on number of vertices and number of edges, and even randomly permuting the input graphs to improve work balance, among others. However, no considerable and definitive improvements were ever noted over simply using our BFS with the bottom-up hybrid, so this approach was abandoned.

4.4. Coloring Step. The coloring step of our coloring algorithm is implemented quite similarly to our BFS and complete trimming algorithm, and is given in Algorithm 14. Initially, all valid vertices are assigned a color as their vertex id v_{id}, or index, in our graph structure. All valid vertices are then placed into the queue. For all vertices in parallel that are contained in the queue, we check to see if they have a higher color than their children. If they do, the color is passed to the child, and both the parent and child are placed in the thread's next level queue, and globally marked as such for all threads to see.

We place the parent in the queue as well to avoid explicit locks. It is possible and very likely that two parents will have higher colors than a shared child, creating a race condition. Both parents will once again examine their children to make sure that either the color that was given or a higher one has been placed. Additionally, since only a higher color can be assigned, we can ignore the race condition created if a parent has their own color overwritten before they assign their previous one to the child. This approach has been experimentally determined to be much faster than explicit locks.

We also tried to avoid locks instead by going bottom-up and having children look at their parents' and own color and take the largest, avoiding the race condition entirely. However, this is much slower in practice, because either all vertices need to be examined at each iteration, or the out vertices of the child need to be examined to create the queue, effectively doubling the amount of memory transfer needed at each iteration.

Our parallel SCC finding step is fairly standard, as it is a trivial algorithm to parallelize. We simply determine the root vertices by finding all unique colors in the graph, and then run a serial DFS on the transverse graph from each root, only looking at vertices with the same color as the root. We use a DFS here, since there isn't further room for parallelism, and experimentation has shown our serial DFS to be faster than our serial BFS.

4.5. Serial Step. We implement a simple and efficient recursive Tarjan's for our serial algorithm. We chose Tarjan's as our serial algorithm over Kosaraju's, based mainly on superior experimental runtimes. Although Kosaraju's can benefit from parallelization during the backwards step, the benefit when the graph has a small number of (usually) disconnected SCCs, is quite small if not negative.

We experimentally determined that a cutoff of about 100,000 remaining vertices is a relatively good heuristic for switching to the serial algorithm, although this is hardware specific. Some graphs benefit from running coloring all the way to completion, while some others would benefit more from switching to serial sooner. However, determining this cutoff without prior knowledge of the graph is quite difficult (possibly some calculation based on the number of steps needed to fully color the graph?), and the difference is usually close to negligible.

4.6. FW-BW Implementation. For comparison, we have also implemented the fully recursive FW-BW algorithm. For the recursive calls at the end of each FWBW procedure, we use task parallelism via `#pragma omp task`. Each recursive call recieves an array listing the numeric identifiers of vertices currently active in that call. As opposed to a boolean array identifying which vertices are globally still active, we use the integer array that was previously described as being used for marking which SCC each vertex is in. For each new call, all of the vertices active for that call have the same unique (for that specific

Algorithm 14 Multistep Coloring

```
1:  procedure MULTISTEPCOLORING(G)
2:      while NumVerts(G) > cutoff do
3:          for all v ∈ G do
4:              color(v) ← v_id
5:              queue ← v
6:              in_next_queue(v) ← false
7:          end for
8:          while queue ≠ ∅ do
9:              for all v ∈ queue do in parallel
10:                 for n ∈ N_out(v) do
11:                     if color(v) > color(n) then
12:                         color(n) ← color(v)
13:                         if in_next_queue(n) = false then
14:                             in_next_queue(n) = true
15:                             thread_queue ← n
16:                         end if
17:                     end if
18:                 end for
19:                 if any n changed color then
20:                     if in_next_queue(v) = false then
21:                         in_next_queue(v) = true
22:                         thread_queue ← v
23:                     end if
24:                 end if
25:             end for
26:             Synchronize
27:             Single thread does
28:                 for all thread_queue do
29:                     for all v ∈ thread_queue do
30:                         queue ← v
31:                         in_next_queue(v) ← false
32:                     end for
33:                 end for
34:             end single thread
35:             Synchronize
36:         end while
37:         for allunique c ∈ colors do in parallel
38:             SCC(c_v) ← PRED(G(c_v), c)
39:             G ← (G \ SCC(c_v))
40:         end for
41:     end while
42: end procedure
```

call) value marked, which is retrieved from a global value. To check if a vertex is active for the call, we simply need to check what value it has marked. For the active vertices not found in the SCC for the current call, the global value is noted and atomically incremented by three, with the three skipped values being assigned to vertices based on if they are in the R, $(D \setminus S)$, or $(P \setminus S)$ set.

5. Experimental Setup. Experiments were performed on a 2 socket machine with 64GB RAM and Intel(R) Xeon(R) E5-2670 CPUs at 2.60GHz each having 20MB last level cache (*Compton*). Compton was running RHEL 6.1 and compilation was done with `icc` version 13.1.2. The `-O3` optimization parameter

Network	n	m	d_{avg}	d_{max}	Dia.	# SCCs	max SCC
Friendster	66M	1.8B	53	5.2K	32	3M	63M
Italy Web	41M	1.2B	28	10K	~830	30M	6.8M
WikiLinks	26M	0.6B	23	39K	~170	6.6M	19M
HV15R	2M	183M	140	484	~85	25K	2M
LiveJournal	4.8M	69M	14	20K	18	970K	3.8M
WikiTalk	2.4M	5.0M	2.1	100K	9	2.2M	500K
CA Road Net	2.0M	2.8M	1.4	7	849	1.7M	302
Google Web	876K	5.1M	5.8	460	22	410K	430K
XyceTest	1.9M	8.3M	4.2	246	~91	400K	1.5M
R-MAT_20	560K	8.4M	15	24K	9	210K	360K
R-MAT_22	2.1M	34M	16	60K	9	790K	1.3M
R-MAT_24	7.7M	130M	17	150K	9	3.0M	4.7M
R-MAT 2v128e	2.0M	128M	121	8.7K	6	1.0M	1.0M
$G(n,p)$ 2v128e	2.0M	128M	64	107	6	1	2.0M

TABLE 5.1
Network sizes and parameters for all networks used in our analysis.

was used with the `-fopenmp` flag. Environmental variable `KMP_AFFINITY` and/or hwloc was used to control thread locality when needed.

Several large real world graphs were used in the course of this work. They are listed in Table 5.1. These graphs were retrieved from a number of sources, namely the SNAP Database [14], the Koblenz Network Collection [13], and the University of Florida Sparse Matrix Collection [10]. The R-MAT [8] and $G(n,p)$ networks were generated with the GTGraph [15] suite using the default parameters.

Friendster and LiveJournal are social networks. Italy Web and Google Web are web crawls of the It domain and Google, respectively. WikiLinks is the cross-link network between articles on Wikipedia, and WikiTalk is a network of edits across user pages. XyceTest is a Sandia National Labs electrical simulation network and HV15R is a computational fluid dynamics matrix of a 3D engine fan. R-MAT_20/22/24 are R-MAT graphs of scale 20, 22, and 24, respectively, while R-MAT 2v128 is an R-MAT graph set with 2M vertices and 128M edges. $G(n,p)$ 2v128e is a $G(n,p)$ random graph of the same size. Finally, CA Road Net is a graph of the road network of California. For the graphs that were undirected (Friendster and CA road network) we assign a random direction to each edge based on a random boolean initially seeded with `srand(0)`. This was done to transform the problems from finding only connected components to that of finding strongly connected components.

These graphs were selected to represent a wide mix of graph size and structure. The number of SCCs and max SCC both play a large role in the general performance of SCC decomposition algorithms, while the average degree and graph diameter can have a large effect on the BFS necessarily used for these algorithms.

6. Results. In this section, we are going to compare our Multistep algorithm to the parallel baseline FW-BW and Coloring algorithms, as well as to serial Tarjan's algorithm. Additionally, we are going to give justification for the algorithmic choices we have made and their influences on performance.

6.1. Comparison of Algorithms. Figure 6.1 gives a comparison on runtime with 16 cores on Compton for Coloring, Forward-Backward, Multistep, and Tarjan's algorithm on several graphs. Multistep is run as previously described, doing simple trimming, the large SCC removal, coloring until 100K vertices remain,

and then serial Tarjan's. Coloring and FW-BW both run complete trimming, as Coloring greatly benefits from it in the first steps, while FW-BW benefits from it in the later steps. As is apparent from the Figure, Multistep demonstrates the fastest performance on all tested graphs with the exception of the CA road network, where the serial algorithm performs fastest.

The likely explanation for Multistep's performance on CA road network graph is twofold: Firstly, the network is not very large, so parallel overhead can dominate the runtime, contributing to Tarjan's superior relative performance. Secondly, the structure of the graph created by randomly assigning edges to the originally undirected network is very unique. Due to the low average degree and very wide diameter, random edge assignments creates a very high diameter connected component with many small strongly connected components. This structure lends to particularly fast performance of coloring with very slow performance of FW-BW, as can be seen. This is not seen on Friendster due to it's much higher degree, which results in it maintaining its social-network-like structure with one massive and several small strongly connected components. It should also be noted the CA road network is the only graph tested where our initial pivot selection heuristic fails to land within the massive SCC, obviously due to the fact that it does not have one.

As mentioned previously, the number of SCCs (after trimming), is the dominating factor for the runtime of FW-BW, while the largest diameter of the SCCs is the dominating factor for coloring. For graphs with very high numbers of small but non-trivial SCCs (ItWeb, CARoad, WikiLinks), running FW-BW with naïve OpenMP task parallelism results in such high overhead that runtimes are several orders of magnitude greater than the other algorithms. Coloring however, shows fairly consistent performance, about running as fast as Tarjan's for all networks.

This effect is further illustrated in Figure 6.1, which shows parallel scaling of the algorithms on the Friendster and LiveJournal networks. Friendster only has several hundred SCCs after trimming, while LiveJournal has several hundred thousand. As can be seen on LiveJournal, switching from simple recursion in serial to two threads with task-based parallelism almost doubles the runtime. This is observed on several other graphs as well.

6.1.1. Strong Scaling. Figure 6.3 gives strong scaling of our Multistep approach on CA Road Net, HV15R, ItWeb, LiveJournal, R-MAT_24, R-MAT_2v128e, and Wikilinks. Running on 16 cores, we observe a $3\times$ to $5\times$ speedup on all graphs. Although we can utilize hyperthreading to run up to 32 threads on the test systems processors, we observe little to none (and sometimes worse) further speedup by doing so. Runtimes for 16 threads on 8 cores was observed to scale fairly well, and is within about 10% of the presented performance for all algorithms and graphs. As this is a heavily memory-constrained problem, and we are running on a dual socket system, we attempted a variety of different methods to improve scaling, which will be discussed later.

6.1.2. Weak Scaling. Finally, we analyze weak scaling of Multistep, Coloring, and FW-BW by using the R-MAT_20, 22, and 24 graphs. The number of vertices, edges, SCCs, and maximal SCC size increases by approximately $4\times$ from R-MAT_20 to R-MAT_22 and from R-MAT_22 to R-MAT_24. Figure 6.4 gives the results. From this graph, it appears that our approach scales better than either Coloring or FW-BW.

6.2. Bottom-Up BFS and hwloc. In order to improve runtimes and scaling of Multistep beyond 16 threads, we attempted several different methods, the results of which are give in Figure 6.2. For these results, we consider the total time to do both the forward and backwards sweeps while finding the initial massive SCC.

Our baseline approach (FWBW) is a simple parallel BFS utilizing our boolean 'bitmap'. In this, we do the complete forward and backward sweeps as would be done while running the FW-BW algorithm. The first improvement is the hybrid bottom-up approach as described in a preceding section. An immediate $3\times$ to $4\times$ improvement is readily apparent on both graphs. Next, we decrease the work needed on the backward

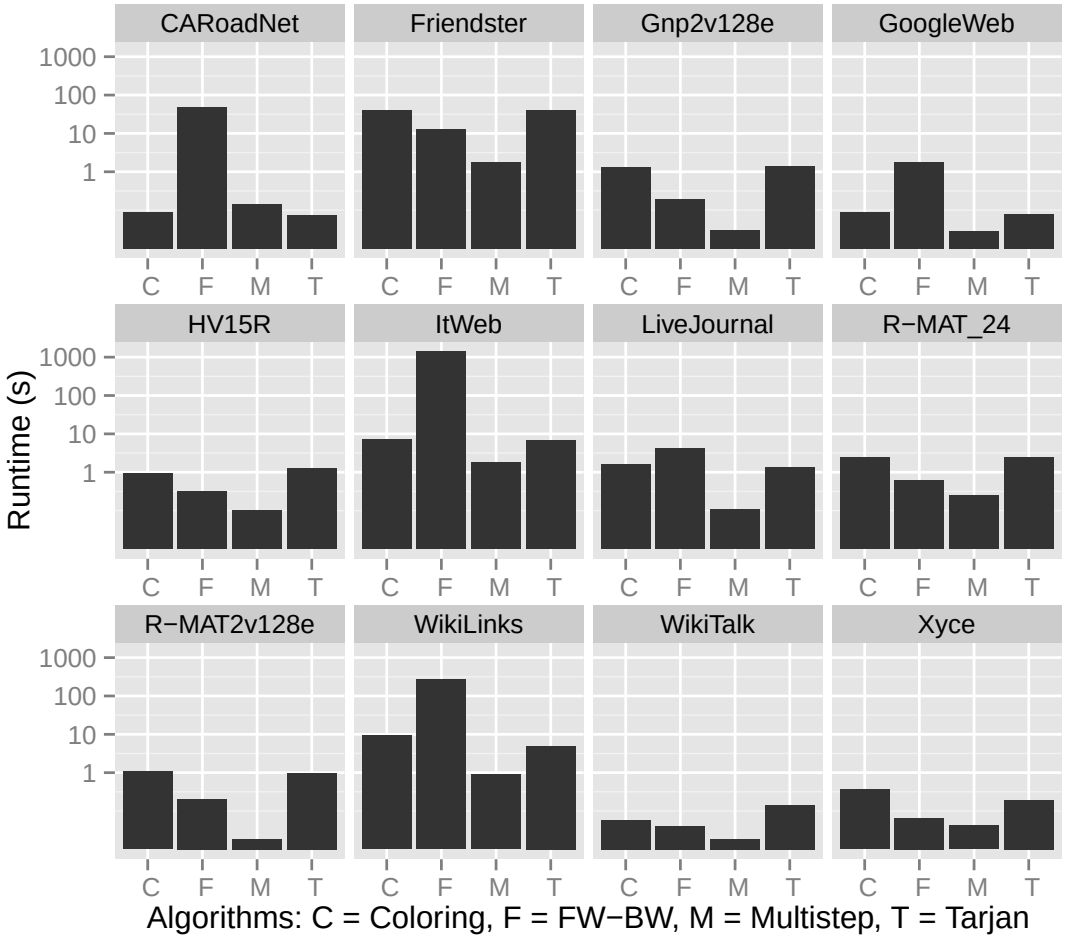
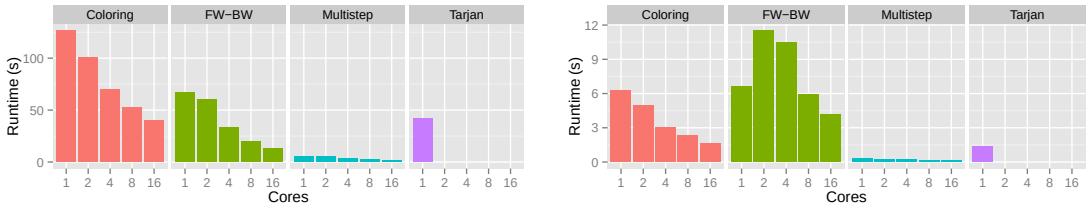


FIG. 6.1. *Runtimes on 16 cores for Coloring, FW-BW, Multistep, and Tarjan on a number of different graphs.*

sweep by recognizing which vertices weren't discovered on the forward sweep and not further exploring them (MS). This is the approach we use for Multistep.

As can be seen on both graphs while using the FWBW-Hybrid and Multistep approaches, minimal to no improvement is seen with hyperthreading. In an attempt to improve parallel scaling, we also investigate partitioning our graph across both sockets.

The first attempted method (MS-hwloc) simply partitioned the graph based on an even split of the edge adjacency lists. Splitting based on the number of vertices was also investigated, but found to be much slower in practice due to severe balance issues. We also produced graphs with an even split of both vertices and edges by doing random permutations of vertices (MS-hwloc Permutation). However, runtimes degraded severely on smaller graphs by doing this, likely due to the loss of graph structure-based locality of vertices and edges. On the larger networks such as WikiLinks, the graph structure's locality doesn't play as much a role due to the wider spread of vertex IDs, so most new edges will need loaded from memory anyways.



(a) Scaling on Friendster

(b) Scaling on LiveJournal

FIG. 6.2. *Parallel scaling of algorithms on Friendster (n=66M, m=1.8B, numSCCs=3M, maxSCC=63M) and LiveJournal (n=4.8M, m=69M, numSCCs=970K, maxSCC=3.8M)*

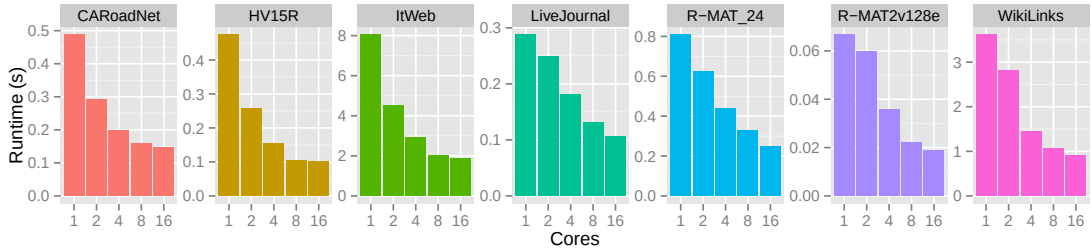


FIG. 6.3. *Parallel scaling of Multistep on up to 16 cores for several graphs.*

Although not attempted, a smarter partitioning or permutation methodology might be more beneficial.

The final attempted partitioning method simply put the out edge adjacency and degree lists on one socket and the in edge lists on the other with both forward and backward sweeps being performed concurrently (MS-hwloc Concurrent).

As can be seen in both Figures, none of the hwloc-based methods, while improving scaling through all 32 threads, decreased runtimes overall. This is most likely attributable to all of the extra overhead required, such as explicitly determining ownership of each vertex, maintaining separate queues, etc.

6.3. Performance of BFS. Overall, our BFS implementation gives very fast single node performance, with the equivalent of over 18 GTEPS on the R-MAT2v128e graph. We use the term 'equivalent', as while a traditional BFS will examine every edge in the connected component it is exploring, the hybrid approach does not. However, we still calculate our GTEPS values as $Runtime/E_c$, where E_c is the number of edges that a traditional BFS would have visited. Additionally, it should be explicitly noted that we do not maintain a BFS tree, as we only care to mark the vertices to find the SCC. The performance of our BFS implementation on every test graph is given in Figure 6.6.

The performance of a standard BFS, and especially the bottom-up BFS, has a high dependence on the average degree in the graph. This strong relationship is exemplified in Figure 6.7, which correlates the GTEPS performance of each of our test graphs as a function of the graphs' average degrees.

6.4. Trimming. Finally, we give justification to our *simple* trimming procedure, as opposed to the complete trimming procedures employed in previous work. Figure 6.4 gives the runtimes at each level of our Multistep procedure for LiveJournal and WikiLinks utilizing simple, complete, as well as no trimming. For both of these graphs, as well as most other ones observed during testing, complete trimming decreases the

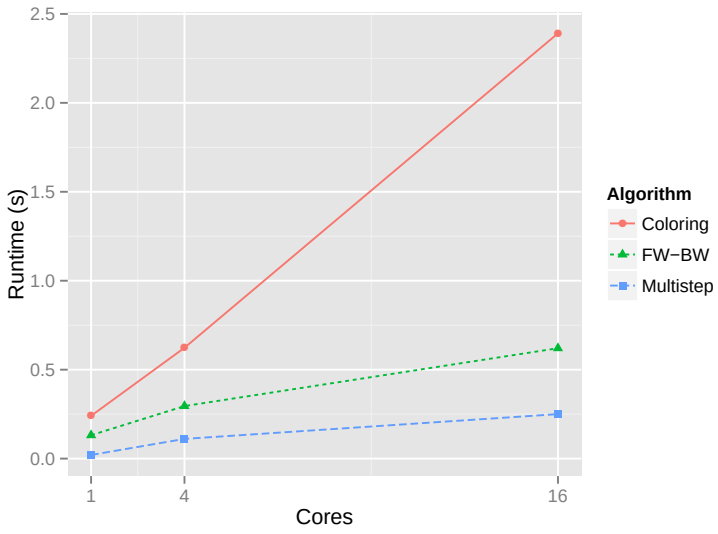


FIG. 6.4. *Approximate weak scaling across R-MAT_20/22/24 (n=560K/2.1M/7.7M, m=8.4M/34M/130M, numSCCs=210K/790K/3.0M, maxSCC=360K/1.3M/4.7M)*

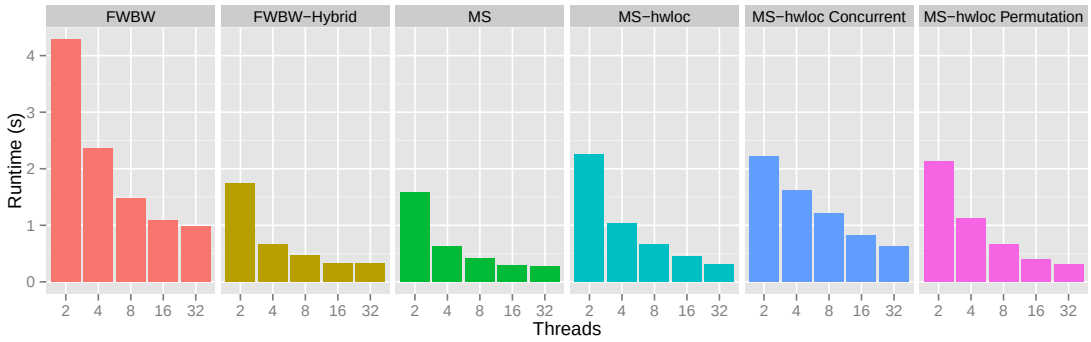
costs of the FW-BW and coloring steps. It has also been observed that complete trimming has the additional benefit of sometimes completely skipping the serial step, by letting coloring find all of the remaining SCCs in parallel in a single step when the graph still has over 100K active vertices remaining. Likely due to the fact that most trivial components are removed, letting only the larger and not usually inter-connected strong components, which coloring can easily find, remain.

However, the benefits of complete trimming never balance out the additional costs, in some instances resulting in overall performance worse than no trimming at all. The reason for this is due to the iterative nature of trimming, and the occurrence of the long *tails* of small numbers of vertices being trimmed during each iteration. This is especially apparent on certain graphs, such as WikiLinks and the It Web graph. A recursive approach, while faster in serial, is very difficult to parallelize. Additionally, complete trimming requires each examined vertex to look at all of its neighbors for validity to get the current effective in and out degrees, while simple trimming only needs to look once through both of the degree lists. A single iteration of simple trimming also removes a substantial majority of all possible vertices that can be removed for most of the tested graphs, which makes even running single extra iteration of complete trimming usually 'not worth it'.

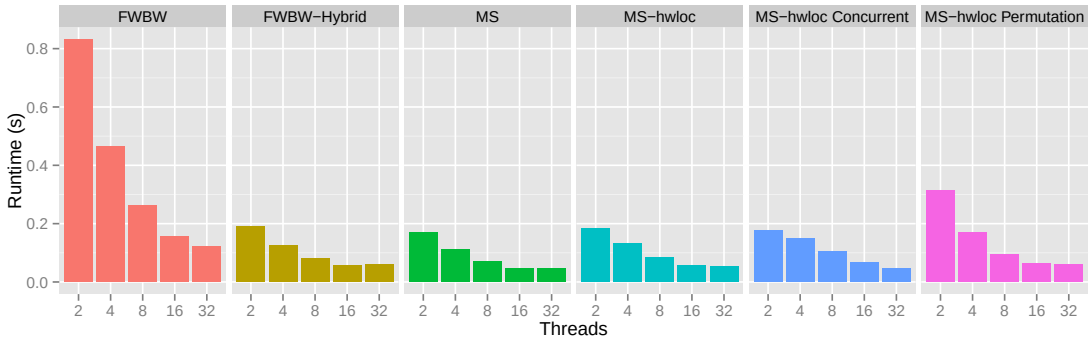
7. Conclusions. This paper introduced the Multistep method, a combination of previous algorithms, for fast parallel strongly-connected component decomposition of billion-edged directed graphs. We also evaluated the existing algorithms which partially comprise the Multistep method, and offered several improvements and optimizations.

Future work might further investigate the possibility of enforcing hardware-based graph partitioning, especially on larger systems with multiple sockets and/or NUMA nodes. Additional improvements might also be made to our trimming and coloring procedures.

Acknowledgments. This work was supported by the DOE Office of Science through the FASTMath SciDAC Institute. Thanks are extended to Erik G. Boman for a number of helpful ideas, conversations, and



(a) BFS methods on Wikilinks



(b) BFS methods on LiveJournal

FIG. 6.5. *Parallel scaling of algorithms on WikiLinks (n=26M, m=0.6B, numSCCs=6.6M, maxSCC=19M) and LiveJournal (n=4.8M, m=69M, numSCCs=970K, maxSCC=3.8M)*

support.

REFERENCES

[1] A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, *Data Structures and Algorithms*, Addison-Wesley, 1983.
[2] V. ARGARWAL, F. P. D. PASETTO, AND D. A. BADER, *Scalable graph exploration on multicore processors*, in Supercomputing, 2010.
[3] J. BARNAT, P. BAUCH, L. BRIM, AND M. CEVSKA, *Computing strongly connected components in parallel on cuda*, in Parallel and Distributed Processing Symposium (IPDPS), 2011 IEEE International, IEEE, 2011, pp. 544–555.
[4] J. BARNAT AND P. MORAVEC, *Parallel algorithms for finding sccs in implicitly given graphs*, Formal Methods: Applications and Technology, 4346 (2006), pp. 316–330.
[5] S. BEAMER, K. ASANOVIĆ, AND D. PATTERSON, *Direction-optimizing breadth-first search*, in Supercomputing, 2012.
[6] 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.
[7] F. BROQUEDIS, J. CLET-ORTEGA, S. MOREAUD, N. FURMENTO, B. GOGLIN, G. MERCIER, S. THIBAULT, AND R. NAMYST, *hwloc: a generic framework for managing hardware affinities in hpc applications*, in Proceedings of the 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP2010), 2010.
[8] D. CHAKRABARTI, Y. ZHAN, AND C. FALOUTSOS, *R-MAT: A recursive model for graph mining*, in SDM, 2004.
[9] J. CHHUGANI, N. SATISH, C. KIM, J. SEWALL, AND P. DUBEY, *Fast and efficient graph traversal algorithm for cpus:*

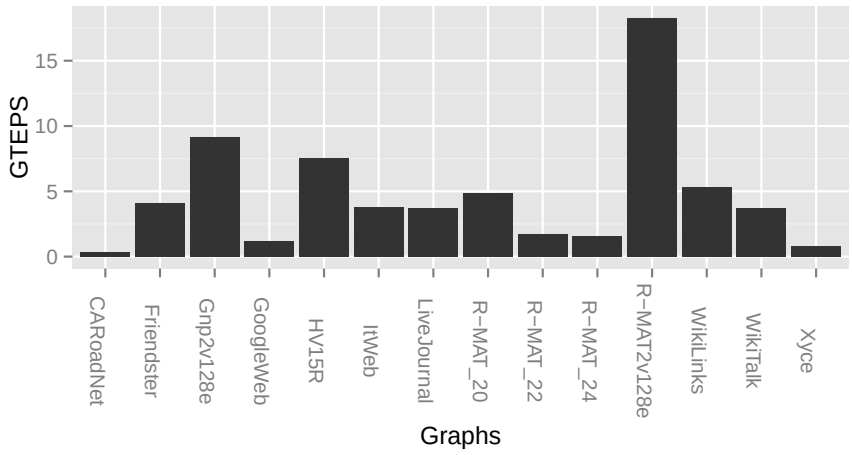


FIG. 6.6. *GTEPS performance of our BFS implementation on all test graphs.*

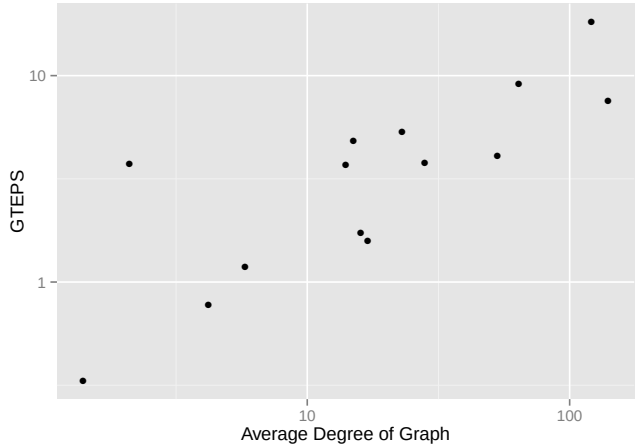


FIG. 6.7. *Correlation between BFS performance and the average graph degree.*

Maximizing single-node efficiency, in Supercomputing, 2012.

[10] T. A. DAVIS AND Y. HU, *The university of florida sparse matrix collection*, ACM Transactions on Mathematical Software, 38 (2011), pp. 1–25.

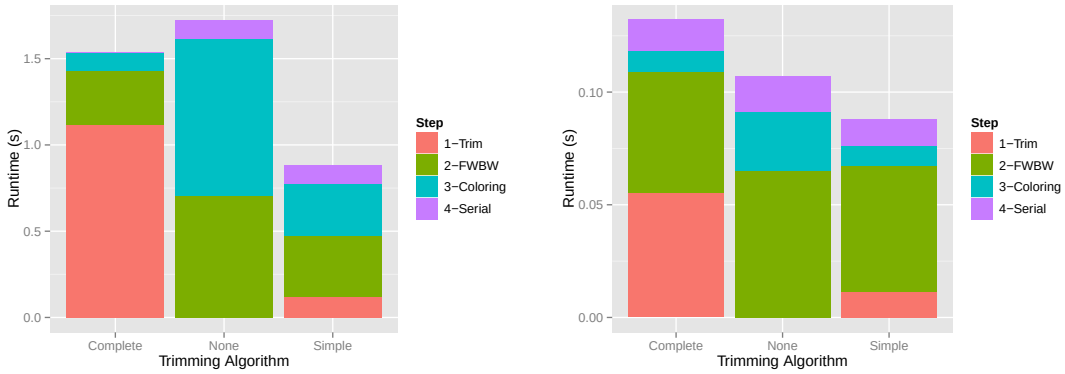
[11] S. HONG, N. C. RODIA, AND K. OLUKOTUN, *Technical report: On fast parallel detection of strongly connected components (scc) in small-world graphs*, tech. rep., Stanford University, 2013.

[12] INTEL, *Intel 64 and IA-32 Architectures Software Developer's Manual, System Programming Guide, Part 1*, vol. 3A, Intel Press, 2011.

[13] J. KUNEGIS, *KONECT - the koblenz network collection*. `konect.uni-koblenz.de`, last accessed 31 July 2013.

[14] J. LESKOVEC, *SNAP: Stanford network analysis project*. `http://snap.stanford.edu/index.html`, last accessed 3 July 2013.

[15] K. MADDURI AND D. A. BADER, *GTgraph: A suite of synthetic graph generators*. `http://www.cse.psu.edu/~madduri/software/GTgraph/`, last accessed 31 July 2013.



(a) Per-step times on WikiLinks

(b) Per-step times on LiveJournal

FIG. 6.8. *Justification for simple instead of complete trimming using WikiLinks and LiveJournal as examples*

[16] 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.

[17] 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.

[18] S. ORZAN, *On Distributed Verication and Veried Distribution*, PhD thesis, Free University of Amsterdam, 2004.

[19] 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.

[20] R. E. TARJAN, *Depth first search and linear graph algorithms*, SIAM Journal of Computing, 1 (1972), pp. 146–160.

[21] H. K. THORNQUIST, E. R. KEITER, R. J. HOEKSTRA, D. M. DAY, AND E. G. BOMAN, *A parallel preconditioning strategy for efficient transistor-level circuit simulation*, in Computer-Aided Design-Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on, IEEE, 2009, pp. 410–417.

[22] 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.

Advanced Architectures and System Software

The articles in this section have an overarching theme of architectures and system sofware. Topic areas of these articles include quantum information processors, resilience mechanisms and the energy case for it, test environments for operating system development, and improving hard disk drive performance.

Scholten and Blume-Kohout deal with the distinguishability of quantum information processors and present analytical work showing that the distinguishability depends on the eigenvalues of sequences of quantum gates implemented by the quantum information processors. *Mills et al.* use an analytical model that accounts for power consumptions and failures to study the performance of checkpoint and replication-based techniques on current and future systems. They propose a replication protocol that can save 40% of the consumed energy while also being 40% faster in exascale-class machines. *Kocoloski et al.* describe a test architecture that allows development on the Kitten OS and runtime stacks without access to specialized hardware. Their techniques allow deployment of Kitten Light Weight Kernel virtual machines without being limited by the underlying hardware. *Crume et al.* use machine learning techniques for improved hard disk drive performance. They show hard disk drive positioning times and request scheduling can be accurately predicted using a neural net, and show that this approach results in lower errror than decision trees in all test sets.

S. Rajamanickam
M.L. Parks
S.S. Collis

July 22, 2014

DISTINGUISHABILITY OF QUANTUM INFORMATION PROCESSORS

T. SCHOLTEN* AND R. J. BLUME-KOHOUT[1]

Abstract. We consider the problem of distinguishing quantum information processors (QIPs). We show how classical statistical distinguishability and quantum state distinguishability can induce a distinguishability measure between QIPs. However, such measures are difficult to calculate in an analytical fashion. Under different assumptions, we present analytical work showing that the distinguishability depends on the eigenvalues of sequences of quantum gates implemented by the QIP.

1. Classical Computer Verification. Given the ubiquity of computers in society today, as well critical functions automated and maintained by them[1], it is vitally important that computer hardware and software behave not only in a logically consistent manner, but also consistent with our design intentions. To meet this goal, stringent testing of computer software and hardware is necessary. Some tools - such as reviews, proof-of-correctness techniques, and direct simulation - have been developed to aid engineers and software designers in the testing of computers [1]. Additionally, by changing how programs are actually written (such as the elimination of the infamous GO TO statement [6]), mistakes and mishaps are further reduced.

Sometimes though, errors slip through the cracks. For instance, Intel Corporation had a design error in a chip manufactured in 1994, causing some special floating-point operations to give incorrect results [12]. Such a chip error was not detected during the usual verification process. This is not too surprising, as verification is an incredibly hard task. If we assume the computer implements some function $f(\mathbf{x})$ on an n-bit string $\mathbf{x}$, there are 2^n possible inputs to check. Thus, it is just not practical to check every input. A trade-off of sorts is in order. By checking that some inputs yield correct results, we can (hopefully) guarantee some level of correctness for other inputs.

1.1. Towards QIP verification. By a QIP, we mean a simple device - harnessing a few qubits at most - which performs some elementary information processing task. It is believed QIPs will provide significant computational advantages in problems regarding the factoring of large numbers [17], as well as providing provably secure cryptography [14]. In a way, a QIP should be thought of as more general than a quantum computer (QC), insofar as every QC is a QIP. However, QIPs do not necessarily have to be universal, nor scalable, as we will explain in section 4.

In contrast to classical information processors, few tools exist to verify QIPs. Tools already developed include randomized benchmarking [13], as well as process tomography [4]. Why have more tools not been developed? Perhaps the most significant complication arises when we consider the types of things a QIP can do to quantum information. In a classical chip, there are certain fundamental gates (AND, NOT, etc) which are concatenated together to perform more complicated logical operations. A QIP can, in principle, execute a continuum of gates, so there are simply more gates which have to be checked.

The primary way to avoid the "infinite gates" problem is to use ε-universal gate sets[2]. That is, given some gate U we wish to implement, one can use the *Solovay-Kitaev algorithm* [5] for constructing U from a small number of primitive gates, up to an arbitrarily small error ε. Thus, if we build a QIP which uses just those primitive gates, then we only need to verify a small number of gates.

In what follows, we wish to propose protocols for experimentalists in the field of quantum information, computation, and/or optics can use to verify that their hardware works the way they intended.

*Physics and Astronomy Department, University of New Mexico, tlschol@sandia.gov

[1]Sandia National Laboratories, rjblume@sandia.gov

[1]Some simple examples include: traffic light timing, aircraft control, and maintaining a particular ambient temperature and humidity in a room. Another, more advanced example would be Google's self-driving car project [18].

[2]The ambitious reader is urged to consult reference [16].

2. Verification, Distinguishability, and Distances. What would it mean to "verify" a QIP? In general, if you are handed a generic black box, there are at least four general statements you could make about it:

1. I have no idea what this box does. (Quantum process tomography will tell you!)
2. I have some idea of what this box does, say X, and I can enumerate a countable set of possible alternatives, say A,B,C, etc. (Quantum hypothesis testing will help you!)
3. I have an idea of what this box *should* do, say X, but it might also do something else, and I do not know what. (Quantum verification will tell you!)
4. I have an idea of what this box *should* do, say X, but if it doesn't do that, then it does Y. (Quantum verification will tell you!)

Clearly, scenario 1 is the most general. However, we focus our attention on scenarios 2, 3, and 4, for the following reason. If we have a protocol for scenario 4, then we can solve the problem posed in scenario 2. On the other hand, if no protocol exists, the problem in scenario 3 cannot be solved. Thus, the problem of pairwise verification (4) is sufficient for distinguishing amongst countable alternatives (2), and is necessary to solve the general problem of verification (3).

Yet how does distance figure into the picture? An intuitive notion of distance is "How far?". Moreover, things which are close together might be hard to tell apart. We therefore often equate the idea of distinguishability to the idea of distance. We should note that distance doesn't always have to mean distance in space - it is just a general notion that if things are farther apart, they have a larger distance. A well-established example of this principle is the Rayleigh criterion. If we observe an image from a source, the criterion tells us whether we should claim the source consists of a single source, or many. It thus turns the question "How many sources?" into the question "If there were two or more, how far apart would they have to be before I could tell them apart?"

Heuristically, if we can distinguish two things, then we should say they have some non-zero distance between them. If they are maximally distinguishable (even if only in principle), we should assign a maximal distance. If they are undistinguishable, then we should say they are distance 0.

In the rest of this report, we discuss ways of turning the above heuristic into a calculable and informationally useful quantity, where we focus on the problem of verification of QIPs as specified in scenario 4 above. A key conception we will explore is how to quantify the *resources* necessary to verify QIPs. Such resources could include the number of times a particular experiment must be repeated, the number of experiments we would have to do, or the amount of time one has to spend doing a particular experiment. These notions will be made clearer in subsequent sections. Section 3 presents helpful background. Section 4 describes previous work on the subject. Our model of a QIP is given in 5. Sections 5.5 and 6.3 present two approaches we attempted in defining distinguishability of QIPs. Sections 7 and 8 present an alternative analytical approach under differing assumptions. Conclusions are given in section 9.

3. Quantum Systems, States, Gates, and Measurements. Readers already familiar with the field can skip this section with no loss of continuity, although we do introduce the notation we will use for the rest of the report. We will explain what states, gates, and measurements are in the quantum computation formalism. For a more detailed reference, consult [15].

3.1. Systems. Before one can speak of quantum states, one must speak of quantum systems. By a system, we mean any physical object or configuration of objects whose behavior exhibits quantum-mechanical phenomena. An obvious example is atoms themselves. However, experimentalists have demonstrated even macroscopic systems can exhibit quantum behavior [8].

3.2. States. In quantum mechanics, it is postulated that every physical system can be represented by a state. A state ρ is a Hermitian $d \times d$ matrix satisfying $tr(\rho) = 1$ and $\rho \geq 0$. In the parlance of quantum

information, ρ represents the state of a system called a *qudit*. In general, we will refer to ρ as a *density matrix*.

One representation of ρ comes from expansion in an arbitrary basis of d^2 Hermitian matrices:

$$\rho = \sum_{i=1}^{d^2} \rho_i M_i \qquad M_i = M_i^\dagger \quad \rho_i \in \mathbb{R} \quad \sum_{i=1}^{d^2} \rho_i tr(M_i) = 1 \tag{3.1}$$

We can also collect the coefficients of this expansion together in a $d^2 \times 1$ vector:

$$|\rho\rangle\rangle = \begin{pmatrix} \rho_1 \\ \rho_2 \\ \vdots \\ \rho_{d^2} \end{pmatrix} \in \mathbb{R}^{d^2} \tag{3.2}$$

If $d = 2$, then ρ represents the state of a qubit, and the expansion and vector representations take on particularly pleasing forms:

$$\rho = \frac{1}{2}(I + \mathbf{r}\cdot\sigma) = \frac{1}{2}\left(I + \sum_{i=1}^{3} r_i \sigma_i\right) \qquad |\rho\rangle\rangle = \begin{pmatrix} 1 \\ r_x \\ r_y \\ r_z \end{pmatrix} = \begin{pmatrix} 1 \\ \mathbf{r} \end{pmatrix} \tag{3.3}$$

The matrices $\{\sigma_i\}_{i=1}^3$ are the usual spin-1/2 Pauli matrices.

The vector $\mathbf{r}$ is known as the *Bloch vector*, and satisfies $0 \leq |\mathbf{r}| \leq 1$. Thus, we can visualize qubits as vectors which live inside the unit ball in $\mathbb{R}^3$. If $|\mathbf{r}| = 1$, then we say the state is *pure*. A pure state may also be written as the projection onto some vector in $\mathbb{C}^2$. That is, if ρ is pure, then we have $\rho = |\psi\rangle\langle\psi|$ for some vector $|\psi\rangle$. If a state is not pure (i.e. $|\mathbf{r}| < 1$), then the state is said to be *mixed*. Mixed states can be written as convex combinations of pure states though:

$$\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i| \qquad \sum_i p_i = 1 \tag{3.4}$$

3.3. Gates. Having specified what a quantum state is, we need to specify how the state changes as we control it. Within a discrete time formalism, such dynamics are often referred to as quantum operations, quantum processes, quantum maps, or superoperators; we will refer to them as quantum *gates*, usually denoted by G. The action of a gate on a density matrix is denoted by $G[\rho]$. Gates act on states via a previously specified *program*, which is classical in nature. That is, the program specifies which gate acts at which time-step.

The best representation for the gate G depends on the representation given for ρ. A common representation - used when ρ is represented as a $d \times d$ matrix - is the Choi decomposition:

$$G[\rho] = \sum_{ij} \chi_{ij} M_i \rho M_j^\dagger \qquad \sum_{ij} \chi_{ij} M_i M_j^\dagger = I \tag{3.5}$$

The matrix χ_{ij} is called the *process matrix* for G. The matrices $\{M_i\}$ are required to form a basis for $d \times d$ matrices, but can otherwise be arbitrary, up to the condition in (3.5).

An alternative representation is realized when ρ is represented as a $d^2 \times 1$ vector. Then, the gate G can be written as a $d^2 \times d^2$ matrix which simply multiplies the vector:

$$G[\rho] = G|\rho\rangle\rangle \tag{3.6}$$

In this representation, we can compose quantum gates very easily: if we apply H to G, which is itself applied to ρ, we have

$$H[G[\rho]] = HG|\rho\rangle\rangle \tag{3.7}$$

Thus, composition turns into regular matrix multiplication. In the Choi decomposition representation, it is not obvious - nor easy! - to do composition.

Regardless of representation, a quantum gate G does need to satisfy one very important criterion. It must be *physically realizable*. This means that it must map density matrices onto density matrices, even if the gate acts trivially on some other part of the state. (For instance, a gate might flip one qudit, but do nothing to another.) This condition is referred to as *complete positivity*:

$$(G \otimes I)[\rho_{AB}] \geq 0 \quad \forall\, \rho_{AB} \geq 0 \tag{3.8}$$

where ρ_{AB} denotes any bipartite state. Rather than check (3.8) for all bipartite states, it is instead sufficient for the gate to satisfy

$$(G \otimes I)[|\psi\rangle\langle\psi|] \geq 0 \tag{3.9}$$

where $|\psi\rangle\langle\psi|$ is an arbitrary maximally entangled state. Under the condition in (3.8), the Choi representation takes on the following form:

$$G[\rho] = \sum_i K_i \rho K_i^\dagger \quad \sum_i K_i K_i^\dagger = I \tag{3.10}$$

The representation in (3.10) is called the *Kraus representation*, and is commonly used instead of the Choi decomposition.

Another condition, unrelated to the physicality of the gate, is that the gate be *trace-preserving*[1]:

$$tr(G[\rho]) = 1 \tag{3.11}$$

In what follows, we usually use the representation in which G is a matrix. Considering the case of qubits, G is a 4×4 matrix; the requirement for G to be trace-preserving means the first row is given by $(1,0,0,0)$, while the rest is (as yet) unspecified:

$$G = \begin{pmatrix} 1 & \mathbf{0} \\ \mathbf{t} & S \end{pmatrix} \quad \mathbf{0} = (0,0,0) \;\; \mathbf{t} = (t_x, t_y, t_z) \;\; S \in GL(\mathbb{C}, 3) \tag{3.12}$$

One can check the action of G is to map the vector $\mathbf{r}$ to $S\mathbf{r} + \mathbf{t}$. The vector $\mathbf{t}$ and the matrix S are arbitrary, up to the requirement of complete positivity specified in equation (3.8).

3.4. Measurements. In general, a quantum measurement on a qudit is specified by a set of Hermitian $d \times d$ matrices $\mathscr{M} = \{E_1, E_2, \cdots E_n\}$ such that

$$0 \leq E_i \leq I \;\forall\, i \quad \sum_i E_i = I \tag{3.13}$$

[1] Trace-decreasing gates do exist, but we do not consider them here. Such gates always correspond to an operation in which we post-select on a particular outcome. In contrast, trace-preserving maps can be implemented deterministically.

Such a set is called a "Positive Operator Valued Measure[1]" (POVM), and each E_i is called a POVM effect. As in the case of density matrices, we can represent effects as $d \times d$ matrices or $d^2 \times 1$ vectors:

$$E_i = \sum_{j=1}^{d^2} e_{ij} M_j \qquad |E_i\rangle\rangle = \begin{pmatrix} e_{i1} \\ e_{i2} \\ \vdots \\ e_{id^2} \end{pmatrix} \in \mathbb{R}^{d^2} \tag{3.14}$$

For the case of qubits, the POVM effects take on a simple form:

$$E_i = \alpha_i I + \mathbf{e}_i \cdot \boldsymbol{\sigma} \tag{3.15}$$

If we measure a POVM, the probability we obtain outcome i, when we measure on a state ρ, is given by

$$p_i = tr(E_i \rho) = \langle\langle E_i \,|\, \rho \rangle\rangle \tag{3.16}$$

The first quantity is used in the representation in which E_i and ρ are matrices; the second in the representation where they are vectors.

4. A General Model of a QIP. As mentioned in section 1.1, a QIP can be thought of as a device which uses a few qubits to do information processing tasks. A QIP could thus be thought of as a full-scale quantum computer satisfying DiVincenzo's criteria [7], except that it need not be scalable. Additionally, a QIP receives program instructions from a classical computer - it is only the data which is quantum.

The canonical model of a QIP works as follows. We have access to certain quantum gates, the set of which we will denote as G. An algorithm or subroutine executed by the QIP is specified by a particular sequence of gates from the set $G = \{G_1, G_2 \cdots G_K\}$:

$$\mathscr{S}_G = G_{S(L)} \circ G_{S(L-1)} \circ \cdots \circ G_{S(1)} \tag{4.1}$$

where $G_i \in G \; \forall \; i$, G_{i+1} is implemented after G_i, and $\circ$ denotes composition: $(A \circ B)[\rho] = A[B[\rho]]$. We use the notation $\mathscr{S}_G$ to make explicit the idea the gates in this sequence come from the set G. We note it depends implicitly on the choice for the maps $S(i)$, described below.

The map $S(i) : \{1, 2, \cdots, L\} \to \{1, 2, \cdots, K\}$ maps the position of the gate in the button-pushing sequence to an index in the set G. (Basically, $S(i)$ takes as its input the i^{th} button push, and tells us which gate from G to put there.) It is specified by a classical string. As such, it renders the QIP as a programmable device. To grant us the physical ability to manipulate the QIP, we assume the QIP has on it buttons whose labels correspond to either preparing ρ, measuring $\mathscr{M}$, or implementing one of the quantum gates.

The state of the art in verification of QIPs is process tomography [4]. By inputting a known state ρ, measuring a known POVM $\mathscr{M}$, and picking sequences of gates from G one is able to invert the measurement statistics and determine the matrix elements of the gates in G. By changing the order in which we push the buttons for the gates, we specify a new function S, as well as a new sequence $\mathscr{S}_G$.

However, this model suffers from the following problem - if you are handed a QIP and told the state it prepares is ρ, how do you *know, to absolute certainty* the state truly is ρ? What about the measurement? It would seem you have to take the experimentalist on faith. If not, the only way to verify the QIP is to use another QIP, generating a situation in which one must verify every QIP to verify just one!

[1] The word 'measure' here refers to the mathematical idea of a measure, not the idea of doing a measurement.

5. A black box model of information processing. To avoid the situation in which we must trust the experimentalist, we need a new model for a QIP, one which specifies as little as possible without simply saying "I know absolutely nothing".

In our model, a QIP is specified by a tuple $(\rho, \mathscr{M}, G)$, where ρ is an unknown density matrix, and $\mathscr{M} = \{E, I-E\}$ is an unknown 2-outcome POVM[1]. G represents a set of quantum gates.

We consider the problem in which we are handed a QIP whose tuple is $(\rho, \mathscr{M}, F)$ or $(\rho, \mathscr{M}, H)$, where $F = \{F_1, F_2, \cdots F_K\}$ and $H = \{H_1, H_2, \cdots H_K\}$. We are asked to determine whether the QIP uses the gates in set F or those in set H. (We note this problem corresponds to the situation described by scenario 4 on page 4.)

Our model significantly relaxes the assumptions present in the general model regarding states and measurements. Usually, a state prepared by a QIP is not known to exact precision; there is always some error. Likewise for the POVM. Our model treats all possible states and measurements as being on equal footing, which we think is a helpful feature.

By construction, we see the only thing we can do is the following:

1. Push the button to prepare ρ.
2. Push some, but not necessarily all, of the buttons which implement quantum gates.
3. Push the button to measure E.

One could argue a bootstrapping approach could be applied to this problem. For instance, why do we even need step 2? Could we not just make ρ and measure E many times? Would that not specify some property of ρ we could calculate and use? To answer this, we must return to the definition of the problem above. Both ρ *and* E are unknown to us. Therefore, it is actually impossible for us to use the naive procedure of just measuring ρ with E to learn anything about ρ. We must necessarily push some of the K buttons to implement gates.

5.1. Black Box Statistics. Suppose we push a sequence of some (or all) of the K buttons which implement quantum gates a total of $L > 0$ times. We do not assume each button is pushed, nor do we assume each button can only be pushed once. Hence, L can be greater than, less than, or equal to K. The representation of this sequence was given in equation (4.1):

$$\mathscr{S}_G = G_{S(L)} \circ G_{S(L-1)} \circ \cdots \circ G_{S(1)} \tag{5.1}$$

In what follows, we pick the representation of the gates to be that described by (3.6), so that composition corresponds to matrix multiplication. The state we obtain after this sequence is simply $\mathscr{S}_G[\rho] = \mathscr{S}_G|\rho\rangle\rangle$. The probability of obtaining an outcome "Yes" is given, using equation (3.16), by[2]

$$p = tr\left(E\mathscr{S}_G[\rho]\right) = \langle\langle E|S_G|\rho\rangle\rangle \tag{5.2}$$

These probabilities are the only quantities we can measure using the QIP. Thus, they are what we will have to use to predict which gate set the QIP is using. **Therefore, the problem of determining the whether the QIP uses the set F or H is related to determining the difference between the statistics generated by sequences of gates in F and of those in H.**

As was mentioned in section 2, we wish to quantify the resources necessary to solve this problem. Below we present protocols used to distinguish classical probability distributions, and discuss the resources required to achieve reasonable distinguishability.

[1] For simplicity, we will denote the outcomes of the measurement as "Yes", associated with effect E, and "No", associated with effect $I-E$. Computer scientists might use "0" and "1", whereas physicists would use $|0\rangle\langle 0|$ and $|1\rangle\langle 1|$, or even $|\uparrow\rangle\langle\uparrow|$ and $|\downarrow\rangle\langle\downarrow|$. Some of the more ambitious among us might use $|\leftarrow\rangle\langle\leftarrow|$ and $|\rightarrow\rangle\langle\rightarrow|$. "Yes" and "No" strike the authors as finding a happy medium.

[2] We use p to denote this probability to avoid writing $p(\text{"}Yes\text{"})$ all the time. Clearly, $p(\text{"}No\text{"}) = 1-p$.

5.2. Distinguishability of Probability Distributions. A classical probability distribution over n variables can be represented as a vector $\mathbf{p} \in \mathbb{R}^n$. Such a vector must necessarily satisfy $\mathbf{p} \geq 0$ and $\sum_i p_i = 1$.

5.3. Single-Shot Distinguishability. If we demand that we have to choose between two probability distributions $\mathbf{f}$ and $\mathbf{h}$ and we are compelled to choose, and we can only sample from the distribution once, then the probability we are correct depends on the one norm between the two vectors:

$$P_{corr} = \frac{1}{2}\left(1 + \frac{1}{2}|\mathbf{f}-\mathbf{h}|_1\right) = \frac{1}{2}\left(1 + \frac{1}{2}\sum_{i=1}^{n}|f_i - h_i|\right) \tag{5.3}$$

For most interesting cases, $\mathbf{f}$ is close to $\mathbf{h}$ (in the Euclidean sense), and so $P_{corr} \sim \frac{1}{2} + \varepsilon$. Clearly, we need to have access to more samples so as to raise P_{corr}.

5.4. Many-Copy Distinguishability. If we have the ability to sample from the distribution many times, the best measure of distinguishability comes from the Chernoff bound [2]. For distributions $\mathbf{f}$ and $\mathbf{h}$, the probability of guessing correctly, given access to N samples from the distribution, goes as

$$P_{corr} \sim 1 - e^{-NC(\mathbf{f},\mathbf{h})} \tag{5.4}$$

where $C(\mathbf{f},\mathbf{h})$ is the *Chernoff exponent*, given by

$$C(\mathbf{f},\mathbf{h}) = -\log \min_{0\leq s\leq 1} \sum_{i=1}^{n} f_i^s h_i^{1-s} \tag{5.5}$$

Although the Chernoff exponent looks difficult to calculate, there are convenient bounds on it in terms of another quantity, known as the *classical fidelity* or *statistical overlap*, defined as $F(\mathbf{f},\mathbf{h})$:

$$F(\mathbf{f},\mathbf{h}) = \left(\sum_{i=1}^{n} \sqrt{f_i h_i}\right)^2 \tag{5.6}$$

From the classical fidelity, we have

$$-\frac{\log F}{2} \leq C \leq -\log F \tag{5.7}$$

So perhaps by using the Chernoff exponent or the classical fidelity, we could define a measure of distinguishability of QIPs, as explained in the next section.

5.5. Towards classical QIP distinguishability. The idea of statistical distinguishability would lead us to consider defining a distance between the QIPs by using the classical statistics generated by them. Using the two distinguishability measures given in the previous sections, a natural measure of distinguishability of the QIPs could be

$$D(F,H) = \frac{1}{2} + \frac{1}{4}\max_S |\mathbf{f}-\mathbf{h}|_1 \quad \text{or} \quad D(F,H) = 1 - \max_S e^{-NC(\mathbf{f},\mathbf{h})} \tag{5.8}$$

where $\mathbf{f} = (f, 1-f)$ and $\mathbf{h} = (h, 1-h)$. The probabilities f and h are defined using (5.2), where we take $G = F$ and $G = H$, respectively. The maximization is carried out over the sequence functions S, which includes an implicit optimization over the sequence length L.

Unfortunately, numerical work indicates that trying to maximize over S results in a function which is neither monotonic nor smooth in those variables. Therefore, we need to move to a regime in which the quantum-mechanical nature of the black box. In the next section, we consider distinguishability measures for quantum states.

6. Common Distinguishability measures for Quantum States. As in the case of classical probability distributions, various distinguishability measures for quantum states are motivated by different operational concerns [9]. Consider the following problem: we are handed one of two states - say either ρ or σ - and asked to choose which state we were given. In analogy with the previous analysis, we consider two cases.

6.1. Single-Shot Distinguishability. If we have access to only one copy of the state, then the *Helstrom measurement* provides us with the information needed to make the best choice [11]. Using this measurement, the probability of choosing correctly is related to the *trace distance* D_T by

$$P_{corr} = \frac{1}{2}\left(1 + \frac{1}{2}tr|\rho - \sigma|\right) \equiv \frac{1}{2}\left(1 + D_T(\rho,\sigma)\right) \tag{6.1}$$

In the event ρ and σ are qubit states with Bloch vectors $\mathbf{r}$ and $\mathbf{s}$ respectively, the trace distance is given by

$$D_T(\rho,\sigma) = \frac{D_E(\mathbf{r},\mathbf{s})}{2} \tag{6.2}$$

where D_E is the usual Euclidean distance. Further, if ρ and σ are qubit pure states, which we will denote by $\rho = |\psi\rangle\langle\psi|$ and $\sigma = |\phi\rangle\langle\phi|$, then

$$D_T(|\psi\rangle\langle\psi|, |\phi\rangle\langle\phi|) = \sqrt{1 - |\langle\psi|\,\phi\rangle|^2} \tag{6.3}$$

6.2. Many-Copy Distinguishability. As in the case of classical probability distributions, the many-copy case involves a Chernoff exponent. However, it is sometimes easier to use the *quantum fidelity* $F(\rho,\sigma)$, defined by

$$F(\rho,\sigma) = \left(tr\sqrt{\sqrt{\rho}\sigma\sqrt{\rho}}\right)^2 \tag{6.4}$$

The fidelity has the property that is is *doubly concave*:

$$F\left(\sum_i p_i |\psi_i\rangle\langle\psi_i|, \sum_i p_i |\phi_i\rangle\langle\phi_i|\right) \geq \sum_i p_i F(|\psi_i\rangle\langle\psi_i|, |\phi_i\rangle\langle\phi_i|) \tag{6.5}$$

From the fidelity, one can define a distance metric between states, known as the *Bures-Uhlman* metric:

$$D_{BU}(\rho,\sigma) = \sqrt{2\left(1 - \sqrt{F(\rho,\sigma)}\right)} \tag{6.6}$$

In the case of qubit pure states, the fidelity takes a particularly simple form:

$$F(|\psi\rangle\langle\psi|, |\phi\rangle\langle\phi|) = |\langle\phi\,|\,\psi\rangle|^2 \tag{6.7}$$

If we have N copies of the quantum state ρ or σ, the fidelity is

$$F(\rho^{\otimes N}, \sigma^{\otimes N}) = tr\left(\sqrt{\sqrt{\rho^{\otimes N}}\sigma^{\otimes N}\sqrt{\rho^{\otimes N}}}\right)^2 = (F(\rho,\sigma))^N \tag{6.8}$$

6.3. Towards quantum QIP verification. How does the above formalism relate to our problem? Recall that our QIP prepares an unknown state ρ, and after the sequence of gates specified by (5.1), the state is $\mathscr{S}_G[\rho]$. Therefore, one measure of distinguishability between the gate sets F and H is to minimize the fidelity between the output state if the gate set is F and that if the gate set is H:

$$D(F,H) = 1 - \min_{S,\rho} F(\mathscr{S}_F[\rho], \mathscr{S}_H[\rho]) \tag{6.9}$$

Why do we minimize over states? Although our QIP prepares an unknown state, the distinguishability between $(\rho, \mathscr{M}, F)$ and $(\rho, \mathscr{M}, H)$ can only be less than the number given in (6.9). It should be noted that to achieve that number, not only does ρ have to picked correctly, but so does the POVM $\mathscr{M}$.

From the double concavity property, we see that to minimize the fidelity, we need to minimize the fidelity with respect to pure states. As (6.9) is written, the sequences $\mathscr{S}_F$ and $\mathscr{S}_H$ come from very general gate sets. To make the minimization more tractable, we need to make assumptions regarding the nature of the sets F and H to be discussed in the next section.

7. Simplifying Assumptions. A general gate as described by equation (3.12) can not only rotate the Bloch vector of the state of a qubit, it can also translate it, shrink it, and distort the Bloch sphere into an ellipse. If we wanted to do the minimization described by (6.9), it might be easier to consider gate sets for which some of these actions do not occur.

Moreover, these assumptions should be in align with what we would want a QIP to actually do. For instance, if a gate simply takes every state to $I/2$, then such a gate is actually a bad one to use in a QIP, as it essentially erases all the information we could have obtained about the state (and subsequently, the computation)! Usually, we assume the gates are pure rotations in $SO(3)$, meaning they take the form

$$M_i = \begin{pmatrix} 1 & \mathbf{0} \\ \mathbf{0} & R_i \end{pmatrix} \quad R_i \in SO(3) \tag{7.1}$$

This representation is helpful when we think of the Bloch vectors. In what follows though, the fidelity is easier to work with when we think of the density matrices as 2×2 matrices, and not 4×1 vectors. Therefore, we will specify the gate as a 2×2 unitary matrix. Abusing notation slightly, we will still write $G = \{G_1, G_2, \cdots G_K\}$ to denote a gate set, but now, $G_i \in SU(2)$.

Under this assumption and notational switch, every sequence $\mathscr{S}_G$ specified by equation (5.1) is itself just unitary matrix:

$$\mathscr{S}_G = \prod_{i=1}^{L} G_{S(i)} \in SU(2) \tag{7.2}$$

As such, the problem of distinguishing F from H is related to the problem of distinguishing the two unitary matrices generated from products of elements of F and H.

With this notation in place, let us re-consider the distinguishability measure in (6.9):

$$D(F,H) = 1 - \min_{S,|\psi\rangle} |\langle\psi|\mathscr{S}_F^\dagger \mathscr{S}_H|\psi\rangle|^2 \tag{7.3}$$

In the above equation, we notice that, even though $S_F^\dagger$ and S_H represent the product of unitary matrices from the two sets, the product $S_F^\dagger S_H$ is itself a unitary matrix. Let us use the notation $\mathscr{U}_{FH}$ to denote this matrix. Then, (7.3) takes the suggestive form

$$D(F,H) = 1 - \min_{S,|\psi\rangle} |\langle\psi|\mathscr{U}_{FH}|\psi\rangle|^2 \tag{7.4}$$

Therefore, the problem of distinguishing F from H is related to finding an optimal sequence S such that $\mathscr{U}_{FH}$ is distinguishable from the identity I.

One way to think about this characterization is to recognize that, within the Bloch sphere representation of the state, the action of the sequence is to rotate the Bloch vector, such rotation depending on which gate set we have. So we can picture the evolution of the Bloch vector as two tracks on the Bloch sphere, where each track is generated by the action of one of the gate sets. If we enter a rotating reference frame which rotates with one of the tracks, thereby freezing on of the tracks in place, the other track evolves under the action ($SO(3)$ representation) of $\mathscr{U}_{FH}$.

We have thus transformed the problem of distinguishing F from H into the problem of distinguishing I from $S_F^\dagger S_H$. Equivalently, it is the problem of distinguishing I from $\mathscr{U}_{FH}$.

8. Optimal States for Unitary Distinguishability. In light of the previous section, we consider here the question of what states will maximize the value of equation (7.4). We focus on states primarily because the optimization over sequences S has no analytical solution, yet[1].

As in the previous section let $\mathscr{U}_{FH}$ be an element of $SU(2)$. We expand $\mathscr{U}_{FH}$ in its eigenbasis:

$$\mathscr{U}_{FH} = e^{i\theta_+}|u_+\rangle\langle u_+| + e^{i\theta_-}|u_-\rangle\langle u_-| \tag{8.1}$$

Then, one can use the method of Lagrange multipliers to show that the optimum state $|\psi\rangle$ for maximizing (7.4) is given by

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|u_+\rangle + |u_-\rangle) \tag{8.2}$$

The distinguishability is then given by

$$D(F,H) = \frac{1}{2}\left(1 - \min_S \cos(\theta_+ - \theta_-)\right) \tag{8.3}$$

The geometric intuition for this result is as follows. The two eigenvectors of $\mathscr{U}_{FH}$ form an orthonormal basis for $\mathbb{C}^2$. As $|\psi\rangle \in \mathbb{C}^2$, it is impossible for $|\psi\rangle$ to be orthogonal to both eigenvectors simultaneously. To avoid having an overlap which is biased towards one eigenvector or the other - which would change the relative weighting of the eigenvalues - one simply splits the difference, and takes an equal superposition of both.

In passing, we note the above expression seems similar in spirit to that of (6.3). Whether such a connection can be made more formal is at this time unknown.

9. Conclusions. Distinguishability of QIPs remains an open question. In pursuing an answer, we must recognize QIPs are not classical computers, which means quantum analogues of classical protocols may not be the best solution for a verification process. The best we can hope for is distinguishability up to some level of (possibly statistical) error.

Even within the realm of known distance measures between quantum states, it is not trivial to generalize to sets of quantum gates. Additionally, to avoid assuming a QIP prepares a known state, one must optimize over all states, a task which is usually hard to do. However, by assuming QIPs use gates which are unitary, some progress can be made in finding a distinguishability measure, and it is possible to do the minimization. The analytical minimization over sequences does remain difficult to calculate though.

[1] Addtionally, convex optimization techniques inform us that to minimize a convex function over several variables, it suffices to minimize over each individually [3]. Although (7.4) may not be convex in S, it is certainly convex in the state.

We are optimistic that it should be possible to define distinguishability measures for QIPs without having to specify an input state or POVM. We may have to augment our model with additional assumptions besides that of unitarity of the gates in order to arrive at a tractable solution.

A distinguishability measure on quantum gates should ideally contain within it an interpretation of a protocol or process by which experimentalists can use the measure to check their quantum devices. How such a measure should be defined remains to be seen, but our work thus far does indicate it should be possible.

10. Acknowledgements. T.S. thanks ABQ Ride for providing transportation to and from Sandia.

REFERENCES

[1] W. ADRION, M. BRANSTAD, AND J. CHERNIAVASKY, *Validation, Verification and Testing of Computer Hardware*, Computing Surveys, 14 (1982), pp. 159–192.
[2] K. AUDENAERT, J. CALSAMIGLIA, R. MUNOZ-TAPIA, E. BAGAN, L. MASANES, A. ACIN, AND F. VERSTRAETE, *Discriminating States: The Quantum Chernoff Bound*, Physical Review Letters, 98 (2007), pp. 1–4.
[3] S. BOYD AND L. VANDENBERGHE, *Convex Optimization*, Cambridge University Press, 2004.
[4] I. CHUANG AND M. NIELSEN, *Prescription for experimental determination of the dynamics of a quantum black box*. http://arxiv.org/abs/quant-ph/9610001, October 1996.
[5] C. DAWSON AND M. NIELSEN, *The Solovay-Kitaev Algorithm*. http://arxiv.org/abs/quant-ph/0505030, August 2005.
[6] E. DIJKSTRA, *Go To Statement Considered Harmful*, Communications of the ACM, 11 (1968), pp. 147–148.
[7] D. DIVINCENZO, *Topics In Quantum Computers*. http://arxiv.org/pdf/cond-mat/9612126v2.pdf, December 1996.
[8] J. FRIEDMAN, V. PATEL, W. CHEN, S. TOLPYGO, AND J. LUKENS, *Quantum superposition of distinct macroscopic states*, Nature, 406 (2000), pp. 43–46.
[9] C. FUCHS, *Distinguishability and Accessible Information in Quantum Theory*, PhD thesis, University of New Mexico, 1995.
[10] A. GILCHRIST, N. LANGFORD, AND M. NIELSEN, *Distance measures to compare real and ideal quantum processes*, Physical Review A, 71 (2005), pp. 1–14.
[11] C. HELSTROM, *Detection Theory and Quantum Mechanics*, Information and Control, 10 (1967), pp. 254–291.
[12] INTEL CORPORATION, *FDIV Replacement Program*. http://www.intel.com/support/processors/pentium/fdiv/wp/, November 1994.
[13] E. KNILL, D. LEIBFRIED, R. REICHLE, J. BRITTON, R. BLAKESTAD, J. JOST, C. LANGER, R. OZERI, S. SEIDELIN, AND D. WINELAND, *Randomized benchmarking of quantum gates*, Physical Review A, 77 (2008).
[14] R. KÖNIG, S. WEHNER, AND J. WULLSCHLEGER, *Unconditional security from noisy quantum storage*. http://arxiv.org/pdf/0906.1030v4.pdf, September 2011.
[15] M. NIELSEN AND I. CHUANG, *Quantum Computation and Information*, Cambridge University Press, 2000.
[16] J. PRESKILL, *Physics/CS 219 Problem Set 4*. http://www.theory.caltech.edu/~preskill/ph219/prob4_09.pdf, 2009.
[17] P. SHOR, *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*, SIAM Journal on Computing, 26 (1997), pp. 1484–1509.
[18] S. THRUN, *What we're driving at*. http://googleblog.blogspot.com/2010/10/what-were-driving-at.html, October 2010.
[19] W. WOOTTERS, *Statistical distance and Hilbert space*, Physical Review D, 23 (1981).

ENERGY CONSUMPTION OF RESILIENCE MECHANISMS IN LARGE SCALE SYSTEMS

BRYAN MILLS, TAIEB ZNATI, RAMI MELHEM* AND KURT B. FERREIRA, RYAN GRANT[1]

Abstract. As HPC systems continue to grow to meet the requirements of tomorrow's exascale-class systems, two of the biggest challenges are power consumption and system resilience. On current systems, the dominant resilience technique is checkpoint/restart. It is believed, however, that this technique alone will not scale to the level necessary to support future systems. Therefore, alternative methods have been suggested to augment checkpoint/restart – for example process replication. In this paper we address both resilience and power together, this is in contrast to much of the competing work which does so independently. Using an analytical model that accounts for both power consumption and failures, we study the performance of checkpoint and replication-based techniques on current and future systems and use power measurements from current systems to validate our findings. Lastly, in an attempt to optimize power consumption for replication, we introduce a new protocol termed *shadow replication* which not only reduces energy consumption but also produces faster response times than checkpoint/restart and traditional replication when operating under system power constraints.

Disclaimer: This work is currently under submission to the Euromicro International Conference on Parallel, Distributed and network-based Processing (PDP 2014).

1. Introduction. The race to build the worlds first exascale-class system has been underway for the last 10 years and many challenges remain. Two of the biggest challenges facing these future systems are power and resilience [13], each a direct result of the massive amount of parallelism necessary to achieve this goal. Delivering exascale performance could require a system over a million sockets, each supporting many cores [1]. This would result in a system with many-millions of cores including increases in memory modules, communication networks and storage devices. With this explosive growth in component count will come a sharp decrease in the overall system reliability and an increase in system power requirements.

System power is a leading design constraint on path to exascale, established by the DOE at no more than 20MW [1]. This challenges the research community to provide a 1000x improvement in performance with only a 10x increase in power. It is expected that exascale-class machines will be capable of consuming more power than that set by the power cap. For example a system might have 150,000 sockets each consuming 200 watts of power at full speed, therefore if all sockets were operating at full power we would be consuming 30 mega-watts. To stay under the 20 mega-watt limit we would need to power off 50,000 of these sockets, or reduce the power consumption of some or all of the cores to stay within budget. While this may seem inefficient, as more hardware is available than can be supported by the power infrastructure, not all applications will be capable of fully utilizing system.

Maintaining efficiency will also be a significant challenge due to the increasing number of expected faults. As the number of components grow, system failures will become routine. Therefore, any resilience scheme must consider its effect on the application's energy and power consumption. In today's systems the response to faults mainly consists of restarting the application, including those components of its software environment that have been affected by the fault. To avoid full re-execution, these techniques checkpoint the execution periodically. Upon the occurrence of a hardware or software failure, recovery is then achieved by restarting the computation from a known good checkpoint.

Given the anticipated increase in failure rate and the time required to checkpoint large-scale compute- and data-intensive applications, it is predicted that the time required to periodically checkpoint an application and restart its execution will approach the system's mean time between failures (MTBF) [5]. Consequently, applications will make little forward progress, thereby reducing considerably the overall performance of the system [14, 16].

*University of Pittsburgh Department of Computer Science, bmills,znati,melhem@cs.pitt.edu
[1]Sandia National Laboratories, kbferre,regrant@sandia.gov

To increase overall system performance process replication has been proposed as a scalable fault tolerance method that can be more efficient in exascale-class systems [5]. A major criticism of replication is the necessary additional resources, especially when considering the power limitations imposed in exascale-class machines.

The objective of this paper is to compare the power and energy consumption of coordinated checkpointing and replication techniques. By looking at fault tolerance from the prospective of power we have found opportunities for making power-aware optimizations to replication. To this end, we propose *shadow replication*, a power-aware process replication protocol which provides faster response times and is more efficient than both checkpoint/restart and traditional replication. We show that *shadow replication* can save 40% of the consumed energy while also being 40% faster in exascale-class machines.[1]

The remainder of the paper is organized as follows: section 2 provides a description of the resilience methods explored in this paper. Section 3 introduces the analytical framework used to model the behaviors of these methods in large-scale systems, presenting this analysis in Section 4. We validate our methods in current systems using experimental data presented in Section 5. In Section 6 we provide a brief review of work in this area and have concluding remarks in 7.

2. Resilience Methods.

2.1. Coordinated Checkpointing. Coordinated checkpoint/restart methods are the most widely used fault tolerance method in HPC environments. The dominant reason for this popularity is its simplicity to implement and the natural synchronization points required are present in most applications. In coordinated checkpointing, all running processes periodically pause their execution and write their state to a stable storage device. Once they have finished writing, the application proceeds with its execution. In the event of a failure, all processes restore from the last good checkpoint and resume execution collectivly from that point.

2.2. Uncoordinated Checkpointing. Another approach that has been suggested to improve the performance of checkpointing systems is uncoordinated or asynchronous checkpointing [2, 10, 11]. In these systems, nodes generally checkpoint and restore from local storage without the synchronization used by coordinated checkpointing. To support a node restoring from a local asynchronous checkpoint, nodes in this approach keep a log of recent messages that they have sent. When a node restores from a previous checkpoint, it can then replay reception of messages using remote nodes' logs.

While this approach can increase checkpointing performance, it also generally assumes the availability of local storage. In addition, logging increases the latency of messaging operations and potentially takes significant amounts of space. Finally, asynchronous checkpointing approaches can result in cascading rollbacks; recent work attempts to bound the amount of rollback that may be necessary [7], but also places non-trivial limits on application behavior.

2.3. Traditional Replication. Traditional replication is a method in which each application process is replicated on independent computing nodes, such that if one process fails its replica process can continue executing as if the failure did not occur. This is also referred to as process replication and has long been deployed in mission critical applications. Replication in HPC has largely been dismissed because of the additional resources required, but in recent years has been revisited [5] because of the increased failure rates expected in exascale-class machines. Furthermore, in this paper we show that power-aware replication techniques can mitigate this concern by using fewer resources.

2.4. Replication Optimizations. In this work we propose and analyze the potential of a number of power-aware optimizations to tradtional replication termed *stretched* and *shadow* replication.

[1] Example shown in detail in Section 4, assumes a socket MTBF of 25 years and 53,000 sockets.

2.4.1. Stretched Replication. Stretched replication works on the assumption that performing work slowly can save energy. This is typically done through the use of dynamic voltage and frequency scaling (DVFS); while this is widely available in modern HPC environments it is rarely used. Stretched replication is a naïve approach which slows down the execution of all processes to the slowest possible speed while maintaining the applications targeted response time.

There are at least two reasons one might want to reduce the execution speed of nodes in an HPC environment. The first we have already mentioned, reducing the execution speed might be necessary to satisfy power limits. Another reason is that coordinated checkpointing already significanly increases the application's time to solution due to the checkpoints and restarts. If we can increase reliablity by slowing down while still staying below this checkpoint slowdown, we can maintain an efficiency greater than that of traditional checkpointing.

2.4.2. Shadow Replication. Stretched replication assumes that by going slower one can always save energy regardless of how long it actually takes to complete the job. This is not always the case in today's computers because machines require a base amount power to operate regardless of the processor speed, we refer to this as the machine's "overhead" power. Previous studies [6, 8] have shown this to be somewhere between 50-85% of the computing nodes' total power. Our experimental results show an overhead power of 60-67%. As the time to solution is increased it results in more energy consumption over the entire job, resulting in a balance between time and power consumption.

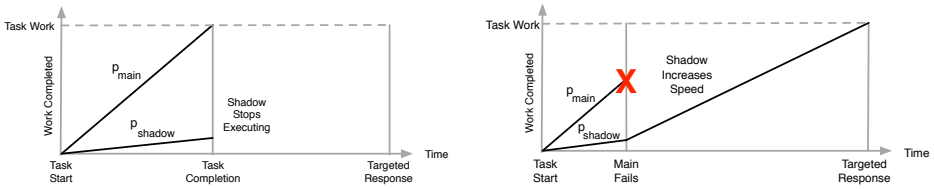
Exploring the energy consumption of replication led to the idea that the replica processes could execute at different execution speeds, while still guaranteeing a response time as good or better than that provided by checkpointing. For each process, shadow replication associates a suite of "shadow processes", whose size depends on the "criticality" and performance requirements of the underlying application. In order to overcome failure, the shadow executes concurrently with the main process, with the shadow and main processes on separate computing nodes. To minimize power, when multiple shadows of a single process exist, each replica shadow operates at decreasingly lower processor speeds. The successful completion of the main process results in the immediate termination of all shadow processes. If the main process fails, the primary shadow process takes over the role of the main process and resumes computation, possibly at an increased speed, in order to complete the task at a targeted response time.

Depending on the occurrence of failure, two scenarios are possible. The first scenario, depicted in Figure 2.1(a), takes place when no failure occurs[1]. In this scenario, the main process executes at the optimum processor speed, namely the speed necessary to achieve the desired level of fault-tolerance, minimize power consumption and meet the target response time of the supported application. During this time, the main process completes the total amount of work required by the underlying application. However, the shadow process, executing at a reduced processor speed, completes a significantly smaller amount of the original work. Because the likelihood of an individual socket failure is low, this scenario is most likely to occur, resulting in a relatively small amount of additional energy consumption to achieve fault-tolerance.

The second scenario, depicted in Figure 2.1(b), takes place when failure of the main process occurs. Upon failure detection, the shadow process increases its processor speed and executes until completion of the task. The processor speed at which the shadow executes after failure is determined such that the targeted response time is acheived. To maximize energy savings failure detection should occur as soon as possible but if this is not possible the shadow process could compute the time the main process should complete, then if that time is reached and the shadow has not been notified, it can assume a failure has occurred.

As mentioned earlier, it is expected we will have additional sockets that will need to be powered off due

[1]For the purpose of this discussion, only a single shadow is considered. The discussion can be easily extended to deal with multiple shadow processes



(a) Case of no failure (b) Case of failure

FIG. 2.1. *Shadow replication execution model.*

to the power consumption limits. Given a power limit and socket power consumption there will be a fixed number of sockets available at any one time. Coordinated Checkpoint/restart would use all of the available sockets to perform work and in the event of failure rollback all sockets, therefore staying under the power limit by restricting the number of sockets used. Traditional replication would take half of the available sockets and use them as replicas, which has been shown to increase system efficiency in exascale-class machines. In contrast, shadow replication has the ability to use additional sockets because the replica sockets are consuming less power by running at a reduced speed. This has the added benefit of allowing additional sockets to work as main processes while still providing system resilience as in traditional replication. In the event of failure there is the potential delay in the time to solution because of the replica's slower execution speed. However, because of the ability to use additional sockets we can show that the expected time to solution is actually faster than both checkpoint/restart and traditional replication methods.

3. Analytical Framework. To evaluate these methods on exascale-class machines we develop a framework of analytical models that represent the energy consumed for each of these methods. We first develop a model that describes the power consumption of a process or group of processes for a given type of work. Next, we derive a model that describes the expected amount of time those processes will be performing a given type of work. By combining these we can then estimate the total expended energy for a given application and system.

3.1. Computational Model. We consider a distributed computing environment of a large number of collaborative tasks (equivalent to ranks in MPI) which communicate frequently. The successful execution of the application depends on the successful completion of all tasks. Therefore, the failure of a single task delays the entire application. We assume the application is perfectly parallizable and has a total amount of work to accomplish, W. The work is assumed to be evenly divided into N tasks which also correspond to the number of system sockets. Because the work is evenly divided each socket will have $W_{task} = \frac{W}{N}$ work to complete. We assume an application with strong scaling, therefore as the number of sockets increases, the amount of work each individual process performs decreases. The amount of work is given in number of clock cycles and each computing socket has a variable speed, σ, given in clock cycles per second. Therefore the total solution time for an application when all sockets are operating at maximum speed is $T_s = \frac{W_{task}}{\sigma_{max}}$.

Each task also has an associated targeted response time, t_{resp}, which is the maximum time that the process will complete it's task. We will represent the targeted response time as a laxity factor, α, of the minimum response time. For example, if the minimum response time is 100 seconds and the targeted response time is 125 seconds, the laxity factor is 1.25. In contrast, checkpointing techniques assume that if a failure occurs the system must always have enough time to re-execute. In our framework this results in

$\alpha = 2.0$. [1]

3.2. Power Model. We start by considering the dynamic CPU power which is known to be affected by the execution speed of the processor. Specifically, one can reduce the dynamic CPU power consumption at least quadratically by reducing the execution speed linearly. Dynamic CPU power, P, can be determined by knowing the chip activity factor, A, the capacitance C, operating voltage, V, and the frequency, f. The dynamic CPU power is therefore represented by the function $P(A,C,V,f) = A \times C \times V^n \times f$, where $n \geq 2$. In this paper we will be performing DVFS, resulting in a execution speed of σ. Because we are scaling both voltage and frequency, the last component allows us to represent power as the function $p(\sigma)$, represented by a polynomial of at least second degree, $p(\sigma) = \sigma^n$ where $n \geq 2$. In the remainder of this paper we assume that the dynamic power function is the cubic, $P(\sigma) = \sigma^3$.

Next, consider the "overhead" power which is consumed regardless of the speed of the processor. This includes both CPU static leakage and all other components consuming power during execution (memory, network, etc.). In this work we define overhead power to be a fixed factor, ρ, of the power consumed when the CPU is operating at full speed. The percentage of overhead power in a system is thus defined as $\frac{\rho}{\rho+1}$. By reducing the execution speed one can only change the dynamic power, the overhead power remains constant.

The energy consumed by a socket executing at speed σ during an interval $[t_1,t_2]$ is given by:

$$E_{soc}(\sigma,[t_1,t_2]) = \int_{t=t_1}^{t_2} (\sigma^3 + \rho\sigma_{max}^3)dt = (\sigma^3 + \rho\sigma_{max}^3)(t_2 - t_1) \tag{3.1}$$

The last component we consider is the energy consumed during an I/O operation, assumed to occur while writing or recovering a checkpoint. We handle this case separately because studies [4] have shown that this operation consumes a significant amount of energy. Similar to overhead power, we define maximum I/O power as a factor of the CPU when operating at full speed. This factor is defined as γ.

$$E_{io}([t_1,t_2]) = \int_{t=t_1}^{t_2} (\gamma\sigma_{max}^3)dt = (\gamma\sigma_{max}^3)(t_2 - t_1) \tag{3.2}$$

3.3. Failure Model. A failure can occur at any point during the execution of the main task and the completed work is unrecoverable. Because the tasks are executing on different computing nodes we assume failures are independent events and that only a single failure can occur during the execution of a task. We further assume that a probability density function, $f(t)$, exists which expresses the probability of the main task failing at time t. In the remainder of this paper we use this exponential probability density function, thus $f(t) = \frac{1}{M_{soc}} e^{-t/M_{soc}}$ where M_{soc} is the socket MTBF.

3.4. Checkpointing Energy Model. Coordinated checkpointing periodically pauses tasks and writes a checkpoint to stable storage. If any one socket fails then this checkpoint is read into memory and used to restart execution. Daly [3] computes the expected total wall clock time, t_w, given the original total solve time (T_s), a system MTBF (M_{sys}), checkpoint interval (τ), checkpoint time (δ) and recovery time (R). System MTBF is dependent upon the number of sockets and the socket MTBF, M_{soc}, this assumes that socket failures are independent events.

$$T_w = M_{sys} e^{R/M_{sys}} \left(e^{(\tau+\delta)/M_{sys}} - 1\right)\left(\frac{T_s}{\tau} - \frac{\delta}{\tau+\delta}\right) \tag{3.3}$$

[1] This assumes a single failure, if multiple failures occur checkpointing has the potential to have $\alpha > 2.0$.

Given this we can define the estimated energy required for a single process using checkpoint and restart (CPR) as E_{cpr}.

$$\begin{aligned} E_{cpr} =& E_{soc}(\sigma_{max}, [0, T_w]) \\ &+ E_{io}([0, \delta)]) \times \frac{T_s}{\tau} + E_{io}([0, R)]) \times \frac{T_w}{M_{sys}} \end{aligned} \tag{3.4}$$

The first part of this equation is because at any given time all processes are either working, writing a checkpoint or restoring from a checkpoint and all sockets are always executing at σ_{max}. The second part adds the energy required to write or restore from a checkpoint times the number of times we will be writing or recovering from a checkpoint. Lastly, we multiply this by the number of sockets, N.

3.5. Replication Energy Model. In this section we develop a model which represents the energy consumption of a replica pair. We then use this model to determine the energy consumption of the combination of replication and checkpointing. This is equivalent to replacing a single socket in the checkpointing model described above with a replica pair. Shadow replication has a main process executing at a single execution speed denoted as σ_m. One of the primary goals of high performance computing is to achieve maximum possible system throughput. Thus when we apply shadow replication to this environment we assume that the execution speed of the main process should be the maximum possible execution speed, $\sigma_m = \sigma_{max}$. If no failure occurs then the task will be completed as soon as possible, known as the minimum response time. In contrast, the shadow process executes at two different speeds, a speed before failure detection, σ_b, and a speed after failure detection, σ_a. This is depicted in Figure 3.1.

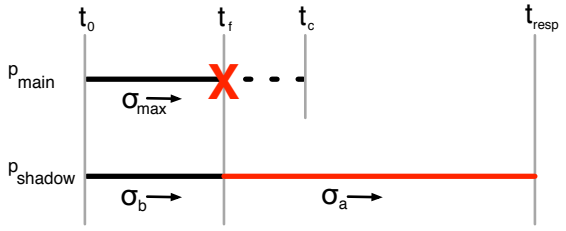
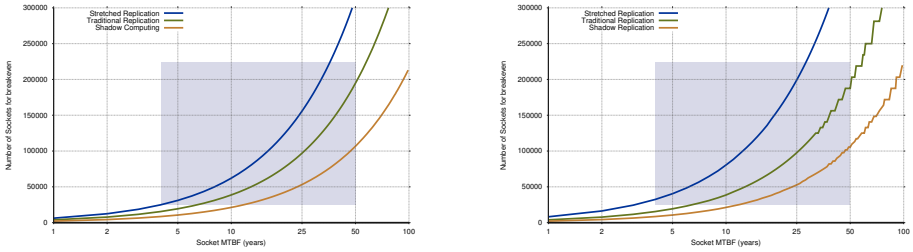


FIG. 3.1. *Overview of Shadow Replication for HPC*

We define some specific time points signaling system events. The time at which the main process completes a task, t_c, is given as $t_c = W_{task}/\sigma_{max}$. Additionally, we define t_f as the time at which a failure in the main process is detected. If no failure is detected we assume that $t_f = t_c$, this is done just to make the formulations below easier to understand. We can also define the time at which the shadow process will complete a task regardless of a failure as, $t_r = t_f + (W_{task} - \sigma_b t_f)/\sigma_a$.

We define the expected energy of a shadow replica-set as the summation of the expected energy consumed by the main and shadow process given our failure model. We assume at most one failure in the main process occurs but this can easily be extended to support multiple failures. In our analysis we have found that multiple failures made no discernible difference for socket MTBF's exceeding one year – a reliability easily reached for future systems.



(a) Breakeven Energy (b) Breakeven Time

FIG. 4.1. *Breakeven points for both energy and time given a fixed checkpoint time of 15 minutes and a system overhead power of 60%.*

$$\begin{aligned} E_{rep} = &\int_{t=0}^{t_c} (E_{soc}(\sigma_{max},[0,t]) + E_{soc}(\sigma_b,[0,t]))f(t)dt \\ &+ \int_{t=0}^{t_c} E_{soc}(\sigma_a,[t,t_r])f(t)dt \\ &+ (1 - \int_{t=0}^{t_c} f(t)dt)(E_{soc}(\sigma_{max},[0,t_c]) + E_{soc}(\sigma_b,[0,t_c])) \end{aligned} \tag{3.5}$$

The first part of this equation represents the expected energy consumed by the main and shadow process before a failure occurs in the main process. This is the summation of the expected energy consumed by the main plus the energy consumed by the shadow given our failure model over the total duration, 0 to t_c. The second part of this equation is the expected energy consumed by the shadow after failure occurs. The duration of this is from the time of failure, t_f, until the shadow completes execution, t_r. The last part is the expected energy consumed by the main and shadow processes in the event that no failure occurs.

This equation is then used as an objective function in the construction of an optimization problem used to find energy optimal execution speeds. We let the speed of the shadow process after failure be $\sigma_a = \sigma_{max}$, this is because we can trade the power consumed by the main process with the shadow process after failure of the main. This reduces the unknowns in the objective to the speed of the shadow process before failure, σ_b. Using traditional optimization techniques we take the derivative, set the result to zero and solve for σ_b. Additionally, we must constrain σ_b such that if the main process fails the shadow process will be able to complete the given work, W, by the targeted response time, R. This is known as the "work constraint" and is represented by the following inequality.

$$t_c * \sigma_b + (t_{resp} - t_c) * \sigma_{max} \geq W \tag{3.6}$$

In addition to providing a model for shadow replication this can be used to represent traditional replication and stretched replication. Traditional replication would be represented by letting $\sigma_m = \sigma_b = \sigma_a = \sigma_{max}$. Stretched replication is represented by letting $\sigma_m = \sigma_b = \sigma_a = \frac{W_{task}}{t_{resp}}$.

4. Analysis. Our analysis finds several system parameters to be important in determining which fault tolerant method is most efficient.

- I/O Bandwidth - This dictates how long it will take to write or recover a checkpoint.
- System Size - The number of total sockets.
- Socket MTBF - Reliability of a single socket in the computing system.

- Overhead Power - The overhead power consumed by the socket, as described in Section 3.2.

When comparing fault tolerant methods, we calculate the energy consumption and time using the power, failure and energy models described in the previous section.

4.1. Scaling and Failure Rates. We compare fault tolerance efficiencies by identifying the breakeven point at which the replication technique is equivalent to that provided by coordinated checkpointing. We use two different breakeven metrics, the expected energy consumed and the expected time to solution. These are related to one another because energy is a function of time, but due to overhead power they are not equivalent. All of the area above the breakeven curve is where the replication technique is more efficient than coordinated checkpointing. Breakeven values are calculated by computing the energy and time required for coordinated checkpointing and comparing that to the energy and time required for the replication technique, when those values match that is the breakeven point.

Figure 4.1(a) shows the energy breakeven point varying system size and socket MTBF using a fixed checkpoint time of 15 minutes. These results show that shadow replication can provide a significant energy savings over traditional replication. For example when socket MTBF is 25 years traditional replication is viable at $96,700$ sockets whereas shadow replication is more efficient at $53,100$. This represents a 46% energy efficiency gain. Unfortunately, stretched replication turns out to be less energy efficient because of the increased time to solution and the presence of overhead power.

Shadow replication achieves this energy savings by slowing down the replicas, this raises the question of how this will effect the expected time to solution. Figure 4.1(b) plots the time to solution breakeven point, and shows that even though shadow replication slows the replicas the expected time to solution is actually shorter than that provided by traditional replication. For example, when socket MTBF is 25 years traditional replication is viable at $97,600$ sockets, whereas shadow replication is more efficient at just $52,700$ sockets, representing a 46% improvement in expected time to solution.

The improvement in time to solution is because shadow replication can utilize additional sockets while consuming the same power because the replicas are consuming less power. This is illustrated in Table 4.1 which shows the active socket counts allowable given a 20 mega-watt fixed power budget. Both stretched and shadow replication have the ability to use additional sockets because they reduce the power consumed by the individual sockets. Stretched replication reduces the power consumed by all processes equally whereas shadow replication only reduces the power consumed by the replica sockets. This is the reason that the expected time to completion of shadow replication outperforms traditional replication. Stretched replication is able to add additional nodes but because it also reduces the processor speed of the main processes, the time to solution is higher than both traditional and shadow replication.

In pure replication the total amount of work remains constant but the the number of sockets is half of that available to coordinated checkpointing. Our model assumes a strongly scaled application which is a fair comparison because each socket would have less work to accomplish in coordinated checkpointing, thus in a failure free case it would be faster than replication techniques. However, because with replication there are two sockets instead of one, the MTBF for the pair is greater than that provided in the single-socket case. The change in MTBF is what allows replication to out perform coordinated checkpointing at large scale. In shadow replication, instead of assuming half of the original sockets are replicas, we calculate the energy optimal σ_b for $\alpha = 2.0$. Then "add" additional sockets remaining under the original power limit, but continuing to use half the sockets as replicas. Stretched replication is similar to shadow replication but both the replica and main use less power.

The conclusion is that shadow replication is both more energy efficient and produces solutions faster than traditional replication in power-limited systems. This is true for the majority of the exascale design space, illustrated by the region in the grey box in Figure 4.1. We assumed a fixed checkpoint time of 15 minutes [5] and a overhead power of 60% which are reasonable system parameters given expected exascale

Overhead %	Method	# Sockets	# Main Sockets
60%	Checkpointing	100,000	100,000
60%	Traditional Replication	100,000	50,000
60%	Stretched $\alpha = 2.0$	153,846	76,923
60%	Shadow $\alpha = 2.0$	124,998	62,499
80%	Checkpointing	100,000	100,000
80%	Traditional Replication	100,000	50,000
80%	Stretched $\alpha = 2.0$	120,230	60,115
80%	Shadow $\alpha = 2.0$	110,636	55,318

TABLE 4.1
Available sockets assuming a 20 mega-watt power limit and 200W per socket.

I/O bandwidth and increased system efficiencies. In the next sections we further relax these assumptions and study the models sensitivity to these parameters.

4.2. Scaling at Different Checkpoint I/O Rates. The checkpoint write times have a significant effect on the efficiency of coordinated checkpointing. These times are directly related to the available I/O bandwidth, as modeled in [14]. Figure 4.2 uses these models to determine the energy breakeven points for I/O bandwidth rates from 500GB/s to 50TB/s, representing a wide range of potential values for an exascale-class machine. For space reasons we only show results for shadow replication, other replication techniques follow a pattern similar to that in Figures 4.1(a) and 4.1(b). Shadow replication is viable for all but very extreme levels of I/O bandwidth.

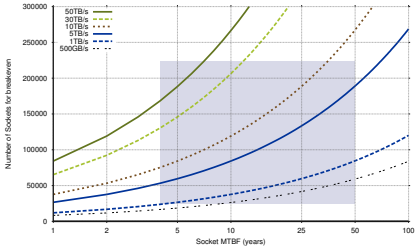


FIG. 4.2. *Shadow replication energy breakeven for different I/O bandwidths. Assumes 16Gb per socket.*

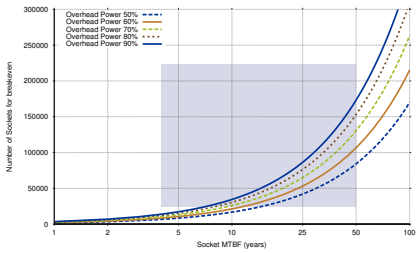


FIG. 4.3. *Shadow replication energy breakeven for various overhead power percentages. Checkpoint time of 15 minutes.*

4.3. Scaling at Different Overhead Power. Table 4.1 illustrated that the number of available sockets decreases as the percentage of overhead power increases. Shadow replication can only reduce dynamic power consumption, leaving it with less power headroom to improve efficiency. This means it can take advantage of fewer main sockets as the available power headroom decreases. Figure 4.3 shows the affect overhead power has upon the energy breakeven point. As expected, as the power overhead increases the potential energy savings also decreases, moving the breakeven point further out into the exascale domain. The conclusion is that overhead power does have an effect upon shadow replication but even if the overhead is 100% it will be no worse than traditional replication. It is expected that future hardware will actually reduce this overhead and therefore make shadow replication more efficient.

5. Implementation and Evaluation. To provide additional insight into the energy consumption of the methods described in this paper and validate the results of our models, we implemented the replication tech-

niques in MPI and measured the energy usage of several applications. Coordinated checkpointing is well established in the HPC community. Instead of implementing our own version we choose to use BLCR in our experiments. Replication has previously been implemented within MPI [16] but existing implementations did not have the ability to change the execution speeds of the replicas. We implemented replication as a MPI profiling library for OpenMPI that allows us to execute unmodified MPI applications using the replication techniques discussed in this paper. To determine optimal execution speeds for shadow computing, an estimate of the total amount of work is needed. Most production job queue systems require applications to estimate their execution time before submitting a job and we use that estimate to determine the execution speeds in our library.

To measure the power consumption of each of these techniques we use a computing cluster that is equipped with component-level power measurement instrumentation. This cluster contains 104 nodes, each with a AMD Llano Fusion, which is a 4-core AMD K10 x86 paired with a 400-core Radeon HD 6550D. Additional details on the power measurement and validation of the system can be found in [9].

To evaluate the different methods of fault tolerance we selected three different MPI applications. The first one is a production application called LAMMPS [15] which is a molecular dynamics code. The two others are mini-applications from Sandia's mantevo suite [17]: HPCCG (a conjugate gradient solver) and miniFE (an implicit finite element method). These applications are important as they represent a range of computational techniques, are frequently run at very large scales on leadership class systems, and represent key simulation workloads for the U. S. Department of Energy.

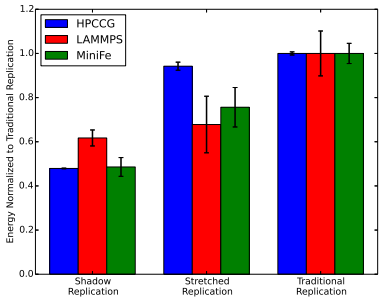


FIG. 5.1. *Experimental results of the energy savings achieved by different replication schemas.*

5.1. Experimental Results. The purpose of these small-scale experiments are to demonstrate that the power-aware replication techniques can provide measurable energy savings for actual HPC applications. In Figure 5.1 the average total energy consumption of multiple application runs are shown for each replication technique, normalized to the energy consumed by traditional replication. This shows that power-aware replication techniques do indeed reduce overall energy but also demonstrates that the amount of savings is application dependent, as previous studies have found [8]. HPCCG and miniFe both demonstrate the maximum energy savings over traditional replication. This is because they are simple applications that are processor bound. Looking at LAMMPS, which is a production application, one can see that the energy savings follows the same trend but the amount of energy saved is less than for the mini-applications. While it is hard to predict exactly what the energy savings will be, it is clear that our proposed techniques do have the potential to save energy.

To confirm our assumptions about overhead power we looked at the component level energy usage over runs of real applications. In Figure 5.2 we show the percentage of energy consumption by component for

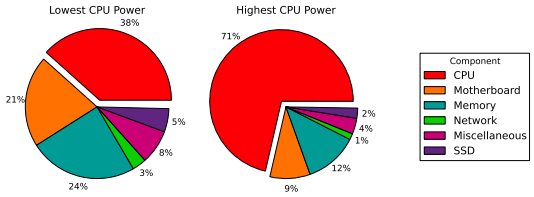


FIG. 5.2. *Component level energy usage for LAMMPS*

multiple runs of the LAMMPS application. The first chart is the energy consumed when running LAMMPS at the lowest possible execution speed. In this case, the CPU consumes 39% of the overall energy. The second chart shows the energy consumption when running at full power in which the CPU consumes 72% of the overall energy. From this the estimated amount of overhead power is 67%, we observed a similar pattern for other applications, concluding that overhead power in our system is 60-67%.

6. Related Work. The analysis of energy/power concern and resilience for HPC is still in its formative stages and as such we are only aware of two papers which look at the energy consumption of fault tolerance schemes. In [4], the authors measure energy consumption of the three main tasks associated with checkpoint-restart methods: writing the checkpoint, recovering from a checkpoint and message logging. They make no attempt to optimize these tasks nor do they explore replication techniques. In [12], the authors look at three different checkpointing techniques and evaluate the energy consumption required. They find that uncoordinated checkpointing with parallel recovery was the best technique at both small and large scale saving up to 17% at 256,000 sockets. Also they show that as the number of sockets grows beyond 256,000 the trend in energy savings of parallel recovery is decreasing. Our work shows that replication increases energy savings as the system size grows, in contrast to this fault tolerance technique.

While not focused upon energy consumption, there is much research attempting to expand checkpoint techniques to exascale-class machines. This work revolves around two main concepts, reducing the checkpoint time and enhancing uncoordinated checkpointing. To our knowledge there is no work looking at optimizing the energy/power consumption of replication to make it a more viable solution for exascale-class environments.

7. Conclusions and Future Work. In this work we show under what circumstances replication is the most energy and time efficient fault tolerance mechanism available. Furthermore, we show the benefit of power-aware modifications to replication. Replication can be made 40% more time and energy efficient using a simple protocol termed *shadow replication*. This savings makes replication a viable fault tolerance solution through the majority of the exascale-class design space. Additionally, we demonstrate at small scale that the proposed modifications actually produce energy savings for actual HPC workloads. Most importantly, this work demonstrates the need to consider power as a first order design constraint in fault tolerance methods for exascale systems.

While these results are promising, there are several avenues of future work being pursued. First, due to the reduced speed of the replicas one concern is the growth of the message queues between replica and their leaders, which occupy memory on the replica nodes. The rate of this queue growth depends highly upon an application message rate. While we are still investigating possible solutions, the most promising draws an analogy between these queues and the message logs used in uncoordinated checkpointing techniques. The idea, therefore, is to use an applications send-determinism property [7] to reduce the size of message logs while avoiding cascading rollbacks. Lastly, we are continuing to explore other power-aware modifications

to replication and ways to efficiently pair replication with checkpointing.

REFERENCES

[1] S. AHERN, A. SHOSHANI, K.-L. MA, A. CHOUDHARY, T. CRITCHLOW, S. KLASKY, V. PASCUCCI, J. AHRENS, E. W. BETHEL, H. CHILDS, J. HUANG, K. JOY, Q. KOZIOL, G. LOFSTEAD, J. S. MEREDITH, K. MORELAND, G. OSTROUCHOV, M. PAPKA, V. VISHWANATH, M. WOLF, N. WRIGHT, AND K. WU, *Scientific discovery at the exascale, a report from the doe ascr 2011 workshop on exascale data management, analysis, and visualization*, 2011.

[2] J. AHN, *2-step algorithm for enhancing effectiveness of sender-based message logging*, in SpringSim '07: Proceedings of the 2007 spring simulation multiconference, 2007, pp. 429–434.

[3] J. T. DALY, *A higher order estimate of the optimum checkpoint interval for restart dumps*, Future Gener. Comput. Syst., 22 (2006).

[4] M. EL MEHDI DIOURI, O. GLUCK, L. LEFEVRE, AND F. CAPPELLO, *Energy considerations in checkpointing and fault tolerance protocols*, in Dependable Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP 42nd International Conference on, 2012.

[5] K. FERREIRA, J. STEARLEY, J. H. LAROS, III, R. OLDFIELD, K. PEDRETTI, R. BRIGHTWELL, R. RIESEN, P. G. BRIDGES, AND D. ARNOLD, *Evaluating the viability of process replication reliability for exascale systems*, in Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC '11, New York, NY, USA, 2011, ACM.

[6] R. GE, X. FENG, S. SONG, H.-C. CHANG, D. LI, AND K. W. CAMERON, *Powerpack: Energy profiling and analysis of high-performance systems and applications*, IEEE Transactions on Parallel and Distributed Systems, 21 (2010).

[7] A. GUERMOUCHE, T. ROPARS, E. BRUNET, M. SNIR, AND F. CAPPELLO, *Uncoordinated checkpointing without domino effect for send-deterministic mpi applications*, in Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium, IPDPS '11, Washington, DC, USA, 2011, IEEE Computer Society.

[8] J. H. L. III, K. T. PEDRETTI, S. M. KELLY, W. SHU, AND C. T. VAUGHAN, *Energy based performance tuning for large scale high performance computing systems*, in 20th High Performance Computing Symposium, Orlando Florida, 2012.

[9] J. H. L. III, P. POKORNY, AND D. DEBONIS, *Powerinsight - a commodity power measurement capability*, in The Third International Workshop on Power Measurement and Profiling in conjunction with IEEE IGCC 2013, Arlington Va, 2013.

[10] Q. JIANG AND D. MANIVANNAN, *An optimistic checkpointing and selective approach for consistent global checkpoint collection in distributed systems*, in Proceedings of the 2007 IEEE International Parallel and Distributed Processing Symposium, Mar. 2007.

[11] D. B. JOHNSON AND W. ZWAENEPOEL, *Recovery in distributed systems using asynchronous and checkpointing*, in Proceedings of the seventh annual ACM Symposium on Principles of distributed computing, 1988, pp. 171–181.

[12] E. MENESES, O. SAROOD, AND L. V. KALÉ, *Assessing energy efficiency of fault tolerance protocols for hpc systems*, in SBAC-PAD, 2012.

[13] U. D. OF ENERGY OFFICE OF SCIENCE, *The opportunities and challenges of exascale computing*, 2010.

[14] R. OLDFIELD, S. ARUNAGIRI, P. TELLER, S. SEELAM, M. VARELA, R. RIESEN, AND P. ROTH, *Modeling the impact of checkpoints on next-generation systems*, in Mass Storage Systems and Technologies, 2007. MSST 2007. 24th IEEE Conference on, 2007.

[15] S. J. PLIMPTON, *Fast parallel algorithms for short-range molecular dynamics*, Journal Computation Physics, 117 (1995).

[16] R. RIESEN, K. FERREIRA, J. R. STEARLEY, R. OLDFIELD, J. H. L. III, K. T. PEDRETTI, AND R. BRIGHTWELL, *Redundant computing for exascale systems*, 2010.

[17] SANDIA NATIONAL LABORATORY, *Mantevo project home page*, 2010.

A VIRTUAL CLUSTER TEST ENVIRONMENT FOR OPERATING SYSTEM AND RUNTIME DEVELOPMENT

BRIAN J. KOCOLOSKI*, KEVIN T. PEDRETTI[1], RYAN E. GRANT[2], AND DAVID DEBONIS[3]

Abstract. Lightweight kernels (LWKs) are valuable operating system substrates in the high performance research community, but their deployment in new environments can be problematic as they are built for a set of highly specialized hardware that is typically only found in supercomputers and large-scale data centers. Developing and testing new features that leverage this hardware typically requires access to such a specialized system, and this access can be very expensive. The problem is particularly relevant for collaborators on the Kitten LWK project at Sandia National Laboratories that work on developing such features, but have difficulty deploying them on Kitten without access to Sandia's specialized systems.

In this paper, we describe a virtual test architecture that allows development on the Kitten OS and runtime stacks without access to specialized hardware. Our solution, which leverages a modification to the Portals message passing interface, makes it simple for users to deploy a cluster of fully-featured Kitten LWK virtual machines in a way that is not limited by the underlying hardware. We also discuss some interesting software techniques that Kitten leverages, including various different uses of Portals, to support scalable, high performance networking for user-level applications.

1. Introduction. Lightweight kernels (LWKs) are ideal choices for supercomputing operating systems (OSes) for a variety of reasons. They provide the bare minimum level of software support to interface users with hardware, and as such they avoid inducing overhead to applications with complicated resource allocation schemes. Accordingly, LWKs have shown to provide better performance, particularly at large scales, than their more full-fledged counterparts [4, 5]. In addition to improved performance, researchers are attracted to LWKs as OS substrates because they have relatively small, simple code bases that provide little resistance to change and thus lend themselves to a variety of research uses [2, 6]. This is in stark contrast to the painstaking experience of trying to add a small feature to a complicated OS architecture like Linux.

Although LWKs are beneficial for these reasons, some factors have limited their deployment to date. First and foremost, by their nature LWKs do not have widespread driver support for commodity hardware devices, as they are built specifically for the specialized hardware required in supercomputers and large-scale data centers. This is typically due to the fact that LWKs are developed from a very small initial code base and functionality is added only as it is desired. This is in contrast to a full-weight kernel (FWK) approach to high performance computing in which a full-fledged OS is stripped down by removing extraneous features with the goal of improving the performance of isolated subsystems. The Kitten LWK[1] developed at Sandia National Laboratories provides a fitting example of this distinction. Kitten now has support for five network devices. For comparison's sake, as of this writing the Linux kernel supports over sixty wireless network devices alone.

Because of this bottom-up, minimalistic approach to development, whenever a user wishes to deploy an LWK on a new or different set of hardware, compatibility issues are likely to arise. Researchers without direct access to supercomputers and high performance systems are confronted with the problem of figuring out how to deploy the kernel on their systems. Specifically, this problem has manifested itself to our research collaborators on the Kitten LWK project that would like to leverage high-speed network protocols and models, including the Portals [3] message passing interface, in Kitten, but do not have access to the systems that Sandia's on site developers do. Frequently these protocols, such as MPI, SHMEM, and various PGAS

*Department of Computer Science, University of Pittsburgh, briankoco@cs.pitt.edu
[1]Sandia National Laboratories, ktpedre@sandia.gov
[2]Sandia National Laboratories, regrant@sandia.gov
[3]Sandia National Laboratories, ddeboni@sandia.gov

[1]Kitten is available at `http://software.sandia.gov/trac/kitten`

models are built to operate over a variety of high-speed network devices which are expensive, or in the case of Portals, simply do not exist in any commodity hardware.

To ameliorate this issue, we have developed a solution that allows users to deploy instances of the Kitten LWK and run any upper-level protocol built for the Portals API in the Kitten user environment by supplementing Portals with functionality for non-specialized hardware. Portals' key contribution in this case is that a protocol built for Portals does not change regardless of how Portals is implemented. Obviously, a high-speed interconnect must be present if performance is the key concern, but in cases where development and testing are the focus, the key is that Portals simply *works*, not necessarily that its performance is *optimal*. In this spirit, Portals has been supplemented with support for a UDP transport, and thus can operate on any system with a UDP stack and hardware capable of transferring IP datagrams. Accordingly, we have also implemented driver support in the Kitten kernel for an Ethernet device, and have incorporated a UDP stack into the kernel that is capable of implementing the functionality required by the Portals UDP transport layer.

It is our vision that these changes can be particularly useful to our collaborators if used in the context of a virtual environment. By using a virtual machine (VM) image that emulates our newly supported Ethernet device, users can quickly and easily deploy a cluster of virtual Kitten instances, connected over the virtual Ethernet connection, and run any high-speed network protocol as long as it uses the Portals API. The key is that users can do this *without any regard to the hardware present on their physical host machine.*

In section 2, we will first discuss the specifics of the virtual architecture that we envision, including details of a virtual cluster that we perceive to be useful for development and testing. Then, in section 3 we will discuss the use of Portals in our architecture, including the support that was added for UDP transports and the factors that went into its integration with Kitten. Finally, we will give an overview in section 4 of the Kitten user-level architecture, including some of the techniques that have been developed to enable high-speed, scalable network performance. We will provide concluding remarks in section 5.

2. A Virtual Development Cluster. Our main goal in this work was to create a development environment that is general enough to be useful to a wide range of users on a wide range of systems, but still expressive enough to allow for the development of high performance protocols that may require specific system and device configurations to achieve optimal performance. Our solution to this goal has two main components. We achieve generality by embedding our development system in a virtual architecture that is deployable on any system that supports the VMWare hypervisor.[1] Additionally, our environment remains expressive enough to support the development of high performance protocols by supporting the Portals message passing interface. In this section, we focus on the former component.

By combining virtualization and hardware emulation, the VMWare hypervisor is an ideal choice for providing an environment that is deployable on a wide range of systems. The hypervisor provides a completely software-based implementation of an Intel e1000 network card that can function in the absence of any such device on the host machine. Additionally, the hypervisor implements a feature called "host-only networking" by which users can connect VMs over a virtual Ethernet network, effectively creating a cluster of VMs. The resulting architecture, which is completely implemented in software, mimics that of a physically networked cluster of machines. Guest operating systems perceive no difference between this architecture and a physical cluster, outside of any performance impacts that they explicitly measure.

Through utilizing these tools, we envision an architecture similar to the one presented in Figure 2.1. Of particular interest in this figure are the following components:

1. The Kitten VMs. These VMs are all running Kitten kernels with e1000 driver support as the guest OS.
2. The front-end development VM. This VM runs the user's favorite Linux flavor as the guest OS.

[1] VMWare Workstation and other VMWare products are available at `http://www.vmware.com`

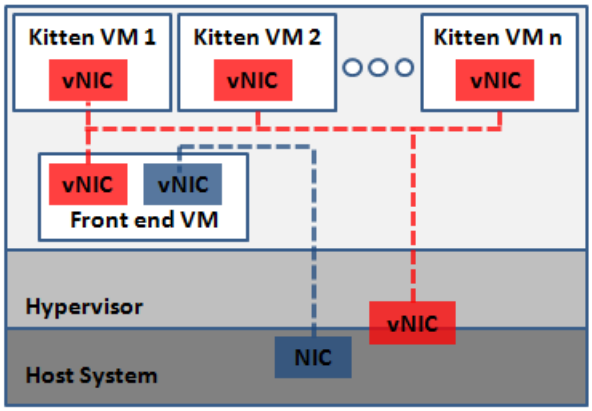


FIG. 2.1. *A virtual LWK development cluster*

3. The host-only network. This network connects the Kitten VMs, the front-end development VM, and the host machine in one network.
4. The externally-accessible network. This network connects the front-end development VM, through the host's physical network interface, to the external network via the NAT protocol.

To generate an architecture similar to this one, a user first needs to create a VM and boot it with their choice of Linux distribution as the OS. This VM will be designated the "front-end development" or simply "front-end" VM. Ideally, if the host system has external network access, the user will create this VM with two virtual network devices: one to support the host-only network, and another to give it external network access through the host device.

Once the user has created the front-end VM, the simplest way to generate a cluster of Kitten VMs is to download the Kitten source tree into the front-end VM and compile the kernel there. Then, to boot the Kitten kernel, the user must create VM templates for the hypervisor to boot, but they do not need to specify any guest OS to the hypervisor. Rather, using the "Preboot Execution Environment" (PXE) feature provided by the hypervisor, the Kitten VMs can request a kernel image to boot over the host-only network. This mode of operation requires that the user first configure both a TFTP server and a DHCP server in the front-end VM to service PXE requests.

Note that of these components, only the externally-accessible network requires the presence of any particular physical hardware device. As long as the host machine has a NIC through which it can access the external network, the hypervisor's "Network Address Translation" (NAT) feature, which effectively allows VMs to "share" the host's IP address, will allow the front-end VM to have external network access as well. This characteristic means that our virtual development cluster is deployable on any system that supports the VMWare hypervisor and has external network access.

3. Upper-Level Network Protocol Support in Kitten. In this section, we will describe the key features that support the development of upper-level network protocols in our virtual development environment.

3.1. The Portals Network Programming Interface. The Portals network programming interface is a low-level API that achieves scalable network performance on high performance systems. Portals, which can be used on scales ranging from small Ethernet environments to massively parallel supercomputers, is a message passing interface originally developed by Sandia National Laboratories and the University of

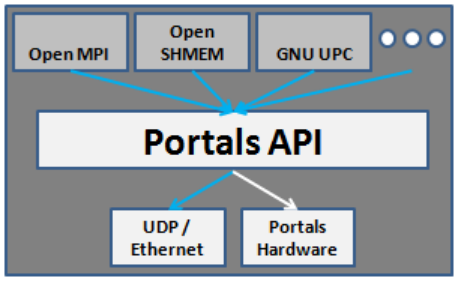


FIG. 3.1. *Upper-level protocols built for Portals don't depend on how on the Portals API is implemented.*

New Mexico; its detailed specification can be found elsewhere.[1] The Portals specification provides building blocks upon which upper-level protocols, such as the Message Passing Interface (MPI), can be efficiently constructed. A key advantage that Portals provides in our development environment, illustrated in Figure 3.1, is that protocols built for Portals do not depend on underlying Portals implementation details. By allowing an additional layer to exist underneath Portals in the network stack, developers are not limited to systems that include Portals hardware. Rather, they can continue to build protocols that leverage the Portals API using a more commodity network device as the underlying transport layer. As demonstrated in Figure 3.1, OpenMPI over Portals is one particular network stack that can leverage Portals for message transportation. Of course, neither OpenMPI nor the other protocols shown in the figure are strictly tied to the Portals API.

An additional advantage of Portals that happens to be unrelated to its use as a high-performance network layer is its utility as a Kitten system service substrate. As we will discuss in section 4, the Kitten user-level environment implements a number of system services that provide various support to applications by leveraging the Portals API. These services do not use Portals for performance reasons, but rather because Portals provides an API that is sufficiently expressive, is relatively easy to program to, and is consistent across the Kitten runtime environment. Using Portals in this fashion obviates the need to develop further OS-level features to support these services.

Prior to this work, Portals could not be deployed in the development environment that we have advocated for two reasons. First, Portals did not support a transport layer capable of communicating over an Ethernet connection. The existing Portals implementation only included support for Infiniband and shared memory transports. Therefore, we have supplemented Portals version 4.0 with the ability to communicate over an underlying UDP transport, allowing it to be deployed over our virtual Ethernet network. The second factor limiting the deployment of Portals in our development environment was that Kitten lacked a TCP/IP network stack capable of transmitting UDP datagrams. As such, we have ported lightweight IP (lwIP[2]), a small, system-independent implementation of the TCP/IP protocol suite, to Kitten and developed a simple UDP layer in Kitten to interface with it. By combining the Portals UDP transport layer, the lwIP TCP/IP implementation and the Kitten e1000 device driver, applications in Kitten can now utilize any upper-level network protocol that is built for the Portals API in our development environment.

4. The Kitten User-level Architecture. For the remainder of this paper, we will focus on the Kitten user-level architecture. Figure 4.1 gives a high-level illustration of the Kitten user-level architecture. Specifically, this diagram illustrates the relationships between the following components:

[1] `http://www.cs.sandia.gov/Portals/portals4-spec.html`

[2] lwIP is distributed under a BSD license. It is available at `http://savannah.nognu.org/projects/lwip`

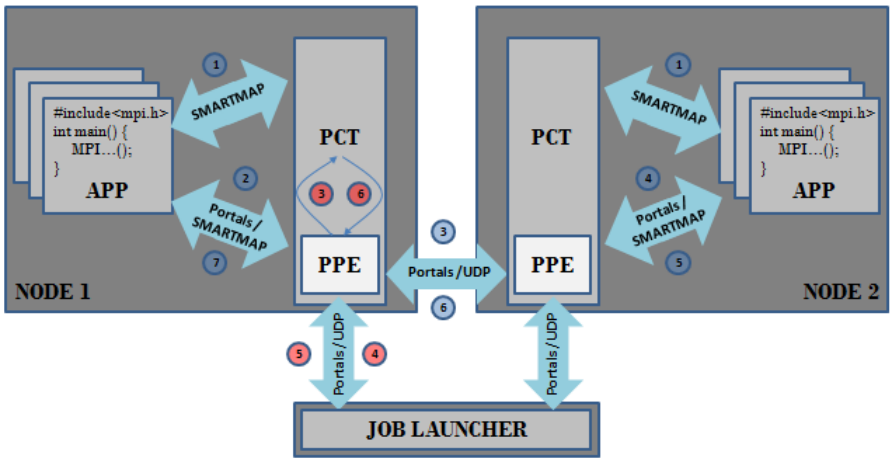


FIG. 4.1. *The Kitten user-level architecture*

- The SMARTMAP shared memory technique
- The application processes (APP)
- The Process Control Thread (PCT)
- The Portals Progress Engine (PPE) *
- The PMI-based job launcher (JOB LAUNCHER) *

While this project focused in general on the integration of each of these components to support the Kitten user-level environment, the bulk of the project's engineering work focused on the latter two components. Many new features were added to the Portals UDP transport to support the PPE operating mode, which is required for deploying Portals in Kitten. Furthermore, the PMI-based job launcher was implemented from scratch. In the following sections, we will describe this architecture in detail, and we will highlight an application's interaction with each component.

4.1. SMARTMAP. SMARTMAP is an operating system technique that allows processes on a multi-core processor to efficiently access each other's memory without involving the kernel [2]. Initially built to support shared memory operations between multiple processes of the same application, SMARTMAP has been integrated into the user-level architecture to support communication between application processes as well as the PCT. SMARTMAP takes advantage of Kitten's very simple approach to memory management. Namely, each process utilizes only one entry in its top level page table mapping structure (PML4, on X86-64). On a 64 bit system, there are 512 entries in this table, which means that 511 of the 512 entries in the PML4 table are unused. SMARTMAP takes advantage of this excess virtual address space by mapping each process' singularly used PML4 entry into every other process' address space.

The mapping strategy for Kitten user-level processes is demonstrated in Figure 4.2. The current convention is that each application maps each other application process, as well as the PCT, into its address space. The PCT also maps each application process into its address space. The diagram is interesting because it applies to every application process as well as the PCT. The only aspect of the picture that changes per process is the field labeled "SELF". Each process maps its own memory twice - once in PML4 entry 0 (that is, each process "owns" the virtual address range 0x0-0x7FFFFFFFFF), and once in the PML4 entry associated with

PML4 Entry	Application Rank	SMARTMAP Slot
511	509	510
...	...	...
3	1	2
2	0	1
1	PCT/PPE	0
0	SELF	N/A

FIG. 4.2. *SMARTMAP memory mappings for Kitten user-level components on x86-64*

```
// 512 GB per PML4 entry
#define SMARTMAP_SHIFT (1 << 39)

void * remote_addr(unsigned int rank, void * addr)
{
    unsigned int slot, pml_entry;

    if (rank == PCT_SRC_ID) // Want to access PCT
        slot = 0;
    else
        slot = rank + 1;

    pml_entry = slot + 1;
    return ((pml_entry << SMARTMAP_SHIFT) | addr);
}
```

FIG. 4.3. *Function for calculating a remote SMARTMAP'd address from a local virtual address*

its application rank.

Together, the following two factors make remote address calculations simple: (1) When a process wants to access an address in another process' address space, the PML4 entry to access is calculated simply and deterministically, depending only on the remote processes' application rank; (2) Addresses in Kitten processes are *symmetric*, meaning the addresses of variables with global scope are the same in every process. The formula for calculating a virtual address for a remote process is given in Figure 4.3.

4.2. The Portals Progress Engine. The Portals Progress Engine (PPE) was implemented as a way to facilitate high performance in situations where many processes generate Portals operations. The PPE allows multiple processes on the same node to issue portals operations without needing to contend for underlying transport resources, such as access to UDP sockets or IB queue-pairs, as well as Portals resources, such as event queues and match entries. Instead, applications share access to these resources by funneling their requests, over a SMARTMAP-facilitated shared memory transport, to the PPE. When an application process wishes to perform a Portals operation, it constructs a message consisting of the specific Portals operation, as well as some book-keeping information about itself, and writes the message into a queue implemented in the shared memory region. The PPE, which is a separate thread inside the PCT, simply loops over this queue looking for Portals requests. Whenever it receives a message, the PPE performs the Portals operation on behalf of the requesting process, and then signals the process by way of a SMARTMAP'd flag that the

operation has completed.

The PPE allows for systems with large numbers of cores to create many application ranks without forcing the processes to contend for underlying system resources, as well as memory and other resources in the Portals layer itself. By providing a high performance communication medium via the SMARTMAP transport, applications can scale to many ranks on the same node without suffering from overhead that would otherwise result.

4.3. The PCT. The Process Control Thread (PCT) is a separate process in the Kitten user-level environment. The first major responsibility of the PCT is to set up the application processes to execute a program. This setup includes creating a specified number of processes and mapping the processes to each other and to the PCT itself via SMARTMAP. Currently, the program to run in the application processes is compiled as raw data from a Linux ELF executable directly into the PCT. Once the PCT has created the address spaces for each process, it copies the raw data from the executable (which must be built statically on Linux) into the Kitten processes' program text segments. The PCT also sets up each of the processes' initial environments.

Once the setup is finished, the PCT either starts each of the processes or waits for a signal from an external process. As part of this work, the PCT has been modified to allow an off node process to communicate, over Portals, that the application ranks should be started. Currently, this communication is accomplished via a Process Management Interface (PMI) [1] message. PMI interfaces process managers with parallel programming libraries in a unified, scalable fashion that eliminates overhead caused by many application ranks contending for programming library resources. In this work, we implemented an additional feature in the PMI API that allows an off-node process to signal to the PCT that it should start the application processes.

The other major responsibility that the PCT has is to service PMI requests from application ranks. The PMI library that has been implemented for Kitten applications uses Portals as the transportation medium. Therefore, when an application generates a PMI request, the request is first embedded in a Portals message. The Portals message is then sent via SMARTMAP to the PPE. At this point, the PCT, which shares an address space with the PPE, either services the message or generates a request to be sent to the master PMI server, if the request cannot be satisfied locally. In either case, once the PCT has a response to the PMI request, the PPE signals to the application that the PMI request is complete.

4.4. Application Execution Flow. With all of the components of the Kitten user-level architecture established, we will now walk through the figure, illustrating the life of an application process, with particular interest on the path that a Portals request generated by the application may take. We again direct the reader's attention to Figure 4.1. There are two general paths that are possible for a Portals message to traverse. The first path is represented by the blue circles in the figure.

1. The PCT initializes the application processes at nodes 1 and 2.
2. An application process at node 1 generates a Portals message and signals the PPE.
3. The PPE sends the message to an off node destination via the UDP transport.
4. The PPE at node 2 receives the message and signals an application process of its arrival.
5. (Optional) The application generates a response and signals the PPE.
6. (Optional) The PPE sends the message back to node 1.
7. (Optional) The PPE at node 1 signals the application process of a response.

The other path that a Portals message can traverse is represented by the red circles. Note that steps 1, 2, and 7 in blue are common for this path as well.

1. The PCT initializes the application processes at nodes 1 and 2.

2. An application process at node 1 generates a Portals message and signals the PPE.
3. The PPE notices that the destination is the PCT and does not forward the message. The PCT receives the message and generates a response or a request.
4. (Optional) The PCT generated a request, which the PPE sends to the job launcher.
5. (Optional) The job launcher generates a response and returns it to the PPE.
6. (Optional) The PPE notices that the destination is the PCT and does not forward the message. The PCT receives the message and generates a response.
7. The PPE signals the application process of a response.

In either sequence of events, the PPE is responsible for transmitting all Portals messages. In the latter case, some attention should be paid to messages delivered to the PCT. Because the PCT and the PPE share an address space, Portals messages sent to the PCT do not need to be copied by the PPE into the PCT's address space, but rather need to copied to the memory descriptor(s) that the PCT had previously created to receive messages. Conversely, when the PCT generates a message to send to the PPE it simply writes to the same queue that application processes write to to signal the PPE; the only difference is that the queue is not shared via SMARTMAP,[1] but again exists in the PCT's own address space.

5. Conclusion. Deploying LWKs can be significantly challenging for those without access to the high performance systems they are built for. This is particularly true for the development of high-speed network protocols that are programmed specifically for specialized low-level network devices and APIs, such as Portals. In this work, we have developed a virtual cluster environment with a Portals network stack that allows LWK and runtime development to proceed without access to specialized hardware. The key distinction is that a protocol built for Portals does not change regardless of whether Portals is implemented in hardware or on top of a commodity network device.

We have also given an overview of the Kitten user-level architecture. The shared memory strategy SMARTMAP allows processes of the same application to access each other's memory without operating system involvement. SMARTMAP also removes extraneous memory-to-memory copies, which are required by other shared memory implementations, because it allows processes to share page tables and write directly into each other's address spaces. With SMARTMAP and Portals as the transportation engines, the Kitten user-level architecture is built to allow applications with many ranks on the same node to avoid overheads that would otherwise occur in more traditional architectures.

REFERENCES

[1] P. BALAJI, D. BUNTINAS, D. GOODELL, W. GROPP, J. KRISHNA, E. LUSK, AND R. THAKUR, *PMI: A Scalable Parallel Process-Management Interface for Extreme-Scale Systems*, in Recent Advances in the Message Passing Interface, 2010.

[2] R. BRIGHTWELL, K. PEDRETTI, AND T. HUDSON, *SMARTMAP: Operating System Support for Efficient Data Sharing Among Processes on a Multi-core Processor*, in Proc. 20th ACM/IEEE Conference on Supercomputing (SC), 2008.

[3] R. BRIGHTWELL, R. RIESEN, B. LAWRY, AND A. MACCABE, *Portals 3.0: Protocol Building Blocks for Low Overhead Communication*, in Proc. 2nd Workshop on Communication Architectures for Clusters (CAC), 2002.

[4] B. KOCOLOSKI AND J. LANGE, *Better than Native: Using Virtualization to Improve Compute Node Performance*, in Proc. 2nd International Workshop on Runtime and Operating Systems for Supercomputers (ROSS), 2012.

[5] J. LANGE, K. PEDRETTI, T. HUDSON, P. DINDA, Z. CUI, L. XIA, P. BRIDGES, A. GOCKE, S. JACONETTE, M. LEVENHAGEN, AND R. BRIGHTWELL, *Palacios and Kitten: New High Performance Operating Systems for Scalable Virtualized and Native Supercomputing*, in Proc. 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2010.

[6] Y. PARK, E. HENSBERGEN, M. HILLENBRAND, T. INGLETT, B. ROSENBURG, K. RYU, AND R. WISNIEWSKI, *FusedOS: Fusing LWK Performance with FWK Functionality in a Heterogeneous Environment*, in Proc. 24th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), 2013.

[1] Well, trivially it is, because all processes are SMARTMAP'd to themselves. See section 4.1.

MACHINE LEARNING OF HARD DISK DRIVE PERFORMANCE

ADAM A. CRUME*, CARLOS MALTZAHN[1], LEE WARD[2], THOMAS KROEGER[3], MATTHEW CURRY[4], RON OLDFIELD[5], AND PATRICK WIDENER[6]

Abstract. Predicting hard disk drive positioning times and request scheduling are crucial parts of predicting hard disk drive performance in general. Existing approaches use white-box modeling and require intimate knowledge of the internal layout of the drive. Automatically learning this behavior is a much more desirable approach, requiring less expert knowledge and fewer assumptions. A barrier to machine learning of hard drive performance has been the existence of periodic behavior with high, unknown frequencies. We show how hard disk drive positioning times and request scheduling can be accurately predicted using a neural net after these frequencies are found using Fourier analysis.

1. Introduction. Hard disk drive modeling may be used as part of a much larger system simulation [19]. Positioning time is a particularly difficult part of the model, due to the high information content and difficulty of extracting this information. Tools such as DIG can extract this information [8]. Unfortunately, they require assumptions about the internal structure of the hard disk drive. This structure is likely to change in the future due to the introduction of shingled hard disk drives or other optimizations, as has happened in the past. Manufacturers do not release this information, so researchers must reverse-engineer a device before modifying DIG to support the new layout.

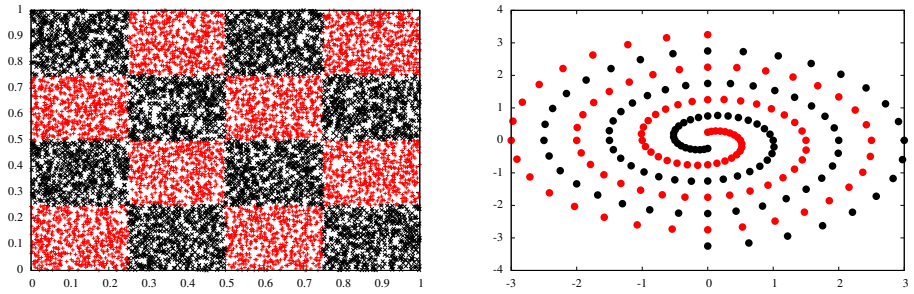
A more desirable approach is to use machine learning to reproduce the behavior with as few assumptions as possible. Since this is a highly non-linear regression problem with interactions between features, neural nets are a reasonable approach. One of the complications in the problem is the existence of unknown, high frequency components caused by the rotational aspect of the drive. Unfortunately, a limitation of neural nets (and many other approaches, including decision trees) is their inability to recognize periodic patterns in the data. This can be seen with the checkerboard problem [25] (fig. 1.1(a)) as well as the two-spirals problem [4, 23] (fig. 1.1(b)).

In fact, Specht and Shapiro found that classifying the points on an 8×8 checkerboard required two hidden layers, one with 200 neurons, and one with 50 [25]. Even so, the accuracy was only 90%. Note that this is equivalent to predicting the sign of $\sin(x)\sin(y)$ for $x, y \in [0, 8\pi]$. Our problem is significantly more complex, with each dimension being five orders of magnitude larger. Clearly, scaling up the neuron count is infeasible. We overcome this by finding these frequencies with Fourier analysis, then feeding them into the neural net explicitly by augmenting the feature vector.

Caching and readahead effects are outside the scope of this report. To avoid these effects, we assume read-only workloads that read individual sectors. In this case, hard drive request latency consists largely of two components:

1. Queue time — the time the request spends in the device's queue, waiting to be processed. Requests may queue to some maximum depth in the device, then be responded to out of order. This can be cast as a classification problem with k classes, where up to k requests are in the drive's queue, and exactly one is scheduled to be processed next.

*Sandia National Laboratories,acrume@sandia.gov
[1]University of California, Santa Cruz, carlosm@cs.ucsc.edu
[2]Sandia National Laboratories,lee@sandia.gov
[3]Sandia National Laboratories,tmkroeg@sandia.gov
[4]Sandia National Laboratories,mlcurry@sandia.gov
[5]Sandia National Laboratories,raoldfi@sandia.gov
[6]Sandia National Laboratories,pwidene@sandia.gov



(a) Checkerboard problem: given X and Y coordinates, determine whether point is in a red or black square

(b) Two-spirals problem: given X and Y coordinates, determine whether point is on the red or black spiral

FIG. 1.1. *Classically hard periodic problems.*

2. Positioning time — the amount of time it takes to start reading sector B, given that sector A was just read. This includes seek time, rotational latency, and settle time. Medium and large seeks are easy to model, because the time can be closely approximated by a simple, smooth function of the logical block numbers (LBNs). Settle time can be modelled as a constant and is easily subsumed into the seek model. Small seeks and rotational latency are difficult to model because these are very high-frequency functions in LBN-space.

Breaking down latency into queuing time and positioning time requires two assumptions: 1) there is exactly one queue, and 2) at most one request may be physically processed at a time. This is true for hard disk drives but breaks down for many other storage devices. Future work may look at predicting latency directly.

We assume that the positioning time and scheduling functions may have components that are locally periodic. This is a fairly lax assumption. First of all, it is well-known that these functions do have locally periodic components for existing hard drives, due to track lengths and track skews that are constant within a serpentine. Periodicity is likely to occur in other devices as well, because of the benefits of regular repetition in designs. Secondly, as described in section 3.1 the cost is minor if these functions do not have locally periodic components.

One approach would be to use a sinusoidal transfer function in the neural net to capture the locally periodic components. Unfortunately, this has a couple of problems: 1) the weights must be tuned very precisely, and 2) many local minima are created near the true frequency values.

These effects are caused by the tunability of the frequencies. Fixing the frequencies externally and feeding in sines and cosines ameliorates these issues. This is the basis of our approach.

One view of the utility of this approach is that it explicitly adds important features (the sines and cosines) that are otherwise hidden. These are theoretically calculable from the existing features, but difficult to learn for many common algorithms including neural nets and decision trees.

1.1. Terminology. The positioning time function is $f(a,b)$, where a and b are the LBNs of the start and end sectors normalized to the range [0,1], and f returns the positioning time in milliseconds. The Fourier transform of f is $\hat{f}(u,v)$.

Unless otherwise specified, times are in milliseconds, distances/periods/lengths are in sectors, and frequencies are in $\frac{\text{cycles}}{237{,}631\,\text{sectors}}$.

2. Related work.

2.1. Predicting request latencies. DiskSim [1] is a well-regarded disk model based on discrete event simulation. It has been validated to produce request-level accuracy. However, it is computationally expensive and difficult to configure for modern disks.

Many analytic models have been created, including work by Lebrecht, Dingle, and Knottenbelt [18]. Analytic models are relatively easy to understand and extremely fast due to their compact formulae. Unfortunately, they require very detailed expert knowledge to create. Often, they are limited to certain classes of workloads and are not useful alone in generalized contexts.

Kelly *et al.* describe a black-box probabilistic model [14] similar to table-based models such as the one by Garcia *et al.* [6]. Requests are categorized based on features including size, LRU stack distance, number of pending reads, number of pending writes, and some RAID-specific information. Table-based models are limited by the table size. If too many dimensions are used, or the granularity within dimensions is too fine, the table grows far too large. This limits the ability of the models to capture smooth transitions or complex cross-dimensional features. The work by Kelly *et al.* ameliorates the issue by essentially not requiring the entire table to be filled in, but the problem is not fully solved.

Mesnier *et al.* create a model for relative performance of storage devices [20]. This first requires workload characterization, which may not be feasible ahead of time for a large-scale system simulation, especially if the workload is dynamic. Furthermore, given a relative model for device B compared to device A, simulating device B still requires a model for device A.

A popular approach is to use regression trees to predict response time. Dai *et al.* predict performance with a combination of regression trees and support vector regression [2]. However, their models are workload-specific, and their prediction errors are based on one-second averages rather than per-request latencies. Wang *et al.* also calculate errors based on windows, using one-minute windows in their case [26].

None of the machine-learning-based approaches have shown low per-request errors.

2.2. Learning periodic functions. As stated in the introduction, traditional neural nets have trouble predicting periodic patterns. Most solutions fall into one of four categories:

1. Direct prediction — Predicts the function directly from the input. These invariably use a fixed interval with a very small number of periods [5, 9, 10, 22, 28]. Extrapolating beyond the training range leads to poor performance [16].
2. Input preprocessing — If the function is known to have period p, map the input x into the range $(0, p)$ using $x \bmod p$. Common examples include time-of-day or day-of-year inputs for functions that are daily or yearly periodic [3, 17]. An alternative is to map it to $\sin(2\pi x/p)$ and $\cos(2\pi x/p)$ [7].
   This leads to very large error if the period is not precisely known ahead of time, or if the input is not exactly periodic. Mapping $kp+r$ for $k \in \mathbb{Z}$, $0 \le r < p \in \mathbb{R}$ should result in r. If the real period is $p+\varepsilon$, then mapping $kp+r = k(p+\varepsilon) - k\varepsilon + r$ yields $r - k\varepsilon \pmod{p+\varepsilon}$. This means that the error ($k\varepsilon$) is proportional to k. Because k corresponds to the number of tracks on a hard disk, k is on the order of a million for modern hard drives. So, if ε is even one millionth of p, the input could be up to 180° out of phase.
3. Periodic activation function — Rather than using $\tanh(x)$ or $\frac{1}{1+e^{-x}}$ as the activation function, use $\sin(x)$. Lack of a saturation region can lead to instability [27]. Periodic activation functions also introduce many local minima [24].
4. Recurrent or delay networks [12, 21] — Feeds the network back into itself, rather than using a feedforward network. Predicts y values from other y values, rather than from x values.

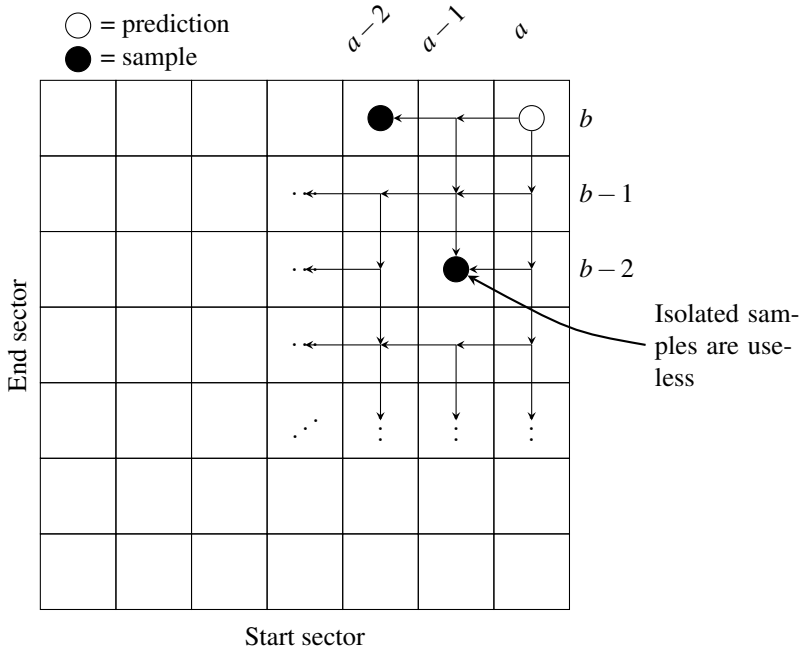


FIG. 2.1. *Predicting positioning times using a recurrent net. Sample sparsity leads to deep recursion.*

The recurrent network approach requires a bit more explanation. This amounts to predicting $\sin(x)$ and $\cos(x)$ from $\sin(x-\delta)$ and $\cos(x-\delta)$, rather than predicting from x. For sine and cosine, this is actually linear:

$$\sin(x) = \sin(x-\delta)\cos(\delta) + \cos(x-\delta)\sin(\delta)$$
$$\cos(x) = \cos(x-\delta)\cos(\delta) - \sin(x-\delta)\sin(\delta)$$

where $\sin(\delta)$ and $\cos(\delta)$ would be constants. The linearity doesn't hold for general periodic functions, but this hints that it is a much easier problem than computing $\sin(x)$ from x directly.

Unfortunately, recurrent neural nets are not feasible for our problem. Since we have a 2D problem, we would predict $f(a,b)$ from $f(a-\delta,b)$ and $f(a,b-\delta)$. Since we only have a very sparse sampling of f, we will almost certainly need to recursively compute $f(a-\delta,b)$ from $f(a-2\delta,b)$ and $f(a-\delta,b-\delta)$, and compute $f(a,b-\delta)$ from $f(a-\delta,b-\delta)$ and $f(a,b-2\delta)$, where $\delta = 1$ sector (fig. 2.1). Limiting the recursive computation to one million predictions would require sampling full rows and columns every thousand sectors. Given that the size of the hard drive is approximately a billion sectors, this comes to two million rows and columns, each with a billion entries. If we assume an average access time of 10 ms, it would take $2 \times 10^6 \cdot 10^9 \cdot 10$ ms $\approx$ 634,000 years to capture the dataset.

3. Fourier analysis. To find periodic behavior in f, the Fourier transform is an obvious place to start. It cannot find periods that are not directly correlated to the value of f, but it is much faster than training a neural net for every period to see which ones are useful. Furthermore, if many periods are useful, their individual impact on the neural net's accuracy may be swamped by noise. Using the Fourier transform allows us to pinpoint useful periods relatively quickly and with high precision.

Let us consider the time it would take to capture the necessary data for a tiny portion of the drive. The first part of the first serpentine on the test hard disk drive is 94 tracks, or 237,631 sectors, which is 0.024% of the drive. Since the range is small enough that seek time is negligible, the average latency is roughly half a rotation, or $8.33/2$ milliseconds. Capturing f for every pair of sectors just for this range would take approximately $8.33/2 \times 237{,}631^2$ milliseconds, or 7.46 years. Obviously, this is infeasible, so we are limited to a very sparse sampling.

The sparsity means that we cannot use the FFT, so we fall back to the brute force method of calculation. First, to keep expressions compact, let's define the vectors $x = (a,b)$ and $\xi = (u,v)$. Then we have:

$$\begin{aligned}\hat{f}(\xi) &= \frac{1}{N}\sum_k f(x_k)e^{-2\pi i x_k \cdot \xi} \\ &= \frac{1}{N}\sum_k f(x_k)\cos(x_k \cdot \xi) + i\frac{1}{N}\sum_k f(x_k)\sin(x_k \cdot \xi)\end{aligned}$$

Only the magnitude is important, and our data is real-valued, so this becomes

$$|\hat{f}(\xi)| = \frac{1}{N}\sqrt{\left(\sum_k f(x_k)\cos(x_k \cdot \xi)\right)^2 + \left(\sum_k f(x_k)\sin(x_k \cdot \xi)\right)^2}$$

Calculating $|\hat{f}|$ for M frequencies and N input points takes $O(MN)$ time. Searching all frequencies in the 2D space leads to infeasible computation time for large datasets or datasets over larger regions of the drive. Since the positioning time depends mostly on the difference between the sectors, the most important frequencies should occur along the diagonal $v = -u$. By searching only this diagonal instead of the entire space, the computation time becomes feasible.

Off-diagonal frequencies occur, but they are mostly related in a straightforward fashion to frequencies on the diagonal. An example would be a dataset that includes track sizes of 10 and 11. Strong coefficients are likely at periods (10, 10), (10, 11), (11, 10), and (11, 11). Since the diagonal provides (10, 10) and (11, 11) (or just 10 and 11 in 1D), the locations of the others could be inferred, although that is not actually necessary.

Other off-diagonal frequencies are likely to be caused by the fact that track length is correlated with the start and end sectors. We believe that we can ignore those and rely on the neural net to pick out which frequencies are relevant for which portion of the disk.

To determine a threshold for strong frequencies, we sample 1000 frequencies at random. The threshold is then set to the mean plus six standard deviations. Assuming the magnitudes are normally distributed, this means that a frequency has roughly a 1 in 500 million chance of being spuriously flagged.

We scan across $v = -u$ with a step size of 0.1, looking for local maxima above the threshold. The peaks were often not centered on integer frequencies, which is why the step size is not 1. After finding a maximum, we perform a local search to fine-tune its position.

Due to structures such as serpentines, f may have higher-order periods. In other words, f may have period p_1 for a subrange, then p_2, then p_1, then p_2, etc., with the switch between p_1 and p_2 being periodic. Currently, we have no method for detecting these explicitly, although we have seen them show up as interesting periods in their own right. These actually cause further problems by reducing the utility of the lower-level periods. The reason is that the multiple regions using period p may be out of phase, meaning that no single sinusoid of period p correlates with f well, causing $|\hat{f}|$ to drop. Essentially, this is a form of mixed interference which can result in a weaker signal than if the signals interfered purely constructively.

3.1. Input augmentation. Once all interesting periods have been found, the input vectors for the neural net are augmented. Given an input (a,b) and periods $p_1,\ldots,p_k$, the augmented input vector is $(a, \cos(2\pi a/p_1), \sin(2\pi a/p_1), \ldots, \cos(2\pi a/p_k), \sin(2\pi a/p_k), b, \cos(2\pi b/p_1), \sin(2\pi b/p_1), \ldots, \cos(2\pi b/p_k), \sin(2\pi b/p_k))$. We use sinusoids of a and b separately rather than sinusoids of $b-a$ because the period changes for each input separately. For example, if a is in a region where p_1 dominates, and b is in a region where p_2 dominates, then $\sin(2\pi a/p_1)$ and $\sin(2\pi b/p_2)$ are useful, but $\sin(2\pi(b-a)/p_3)$ is not likely to be useful for any period p_3.

If no interesting periods are found, then the input vectors are unchanged. This means that for devices with no periodicity, the cost of the periodicity assumption is just the time to search for periods. The neural net is unchanged, and therefore the accuracy is unchanged.

Neural nets are known universal function approximators, so they can find the shape of the periodic waveform by themselves. The *phase* of the periodic signal is actually the important part. Sines and cosines are used because linear combinations result in sinusoids with adjusted phase.

4. Shared weights. The best results were obtained when using subnets with shared weights. This means that the weights on hidden neuron 1 in subnet 1 were forced to be equal to the weights on hidden neuron 1 in subnet 2. The idea is that the subnets map from LBNs to geometrical space, and the main net maps from geometrical space to time. Since the start and end requests share the same LBN to geometry mapping, the subnets should be equal. This is essentially a very simple convolutional neural net with only two instances of the convolving subnet and no overlap of inputs. A similar approach was used by Kindermann *et al.* for solving functional equations with neural nets [15].

More formally, the net assumes f can be expressed as

$$f(a,b) = h(g(a),g(b))$$

where g (the subnet) maps from LBNs to geometry, and h (the main net) maps from geometry to positioning time. Reuse of g, which is equivalent to weight sharing in the neural network, formalizes the assumption that all requests map to geometry in the same way. Technically, f is not restricted as long as the intermediate space is at least as large as the input space, since one could always use $g(x) = x$ and $h = f$, but this predisposes the neural net to learn more useful decompositions.

Each subnet has 32 hidden units and 10 output units, and the main net has 10 hidden units. All neurons use a sigmoidal activation function except the final output, which is linear. The additional hidden units in the configuration with 52 hidden units are intended to compensate for the units removed when subnets are not used.

5. Experimental setup. The device we modeled is a Western Digital Caviar Black WD5002AALX. It has a capacity of 500GB (976,773,168 sectors), rotational speed of 7200 RPM, cache size of 32 MB, and a Native Command Queuing (NCQ) queue length of 32. The drive was connected with SATA. The host runs 64-bit Ubuntu Linux, kernel version 2.6.35-28-server. It has an Intel Core i7 quad-core CPU (plus hyper-threading) running at 2.80 GHz and 12 GB of RAM.

Block-level traces were captured using blktrace. Request latency was defined to be the device-to-completion (D2C) time. This is the time between the operating system I/O scheduler's sending the request to the device driver and receiving a response from the device driver. Tracing was done below the I/O scheduler because that is better understood and will likely be modelled separately. To the best of our knowledge, there is no simple way to capture traces excluding time spent in the device driver, short of bus-level tracing. This requires special hardware setup, though, and the benefit likely to be gained was very small compared to the work required. Because of the speeds of modern CPUs compared to hard drives, the time spent in the device driver should be negligible compared to the time spent in the hard drive.

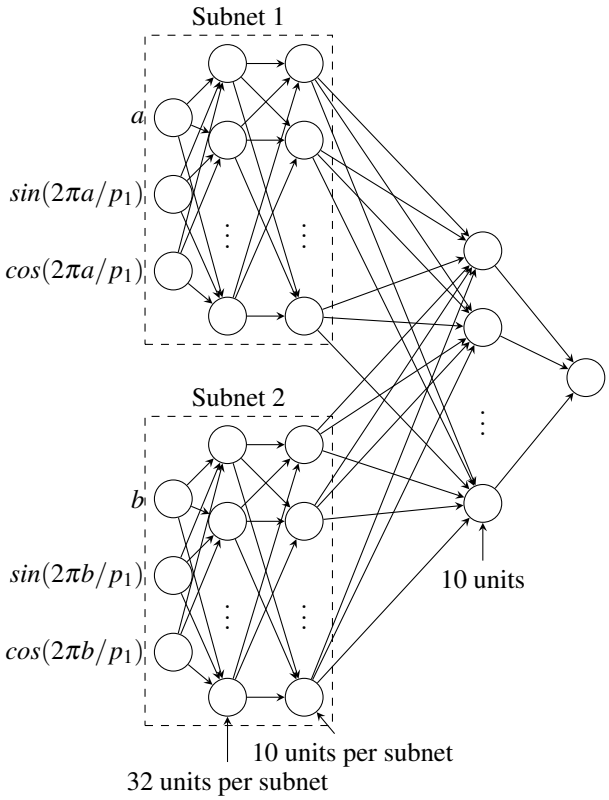


FIG. 4.1. *Network architecture for positioning time prediction when subnets are used. Note that the weights in subnet 1 are identical to the weights in subnet 2.*

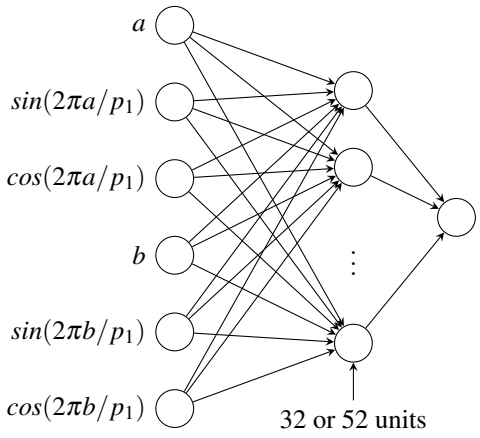


FIG. 4.2. *Network architecture for positioning time prediction when subnets are not used.*

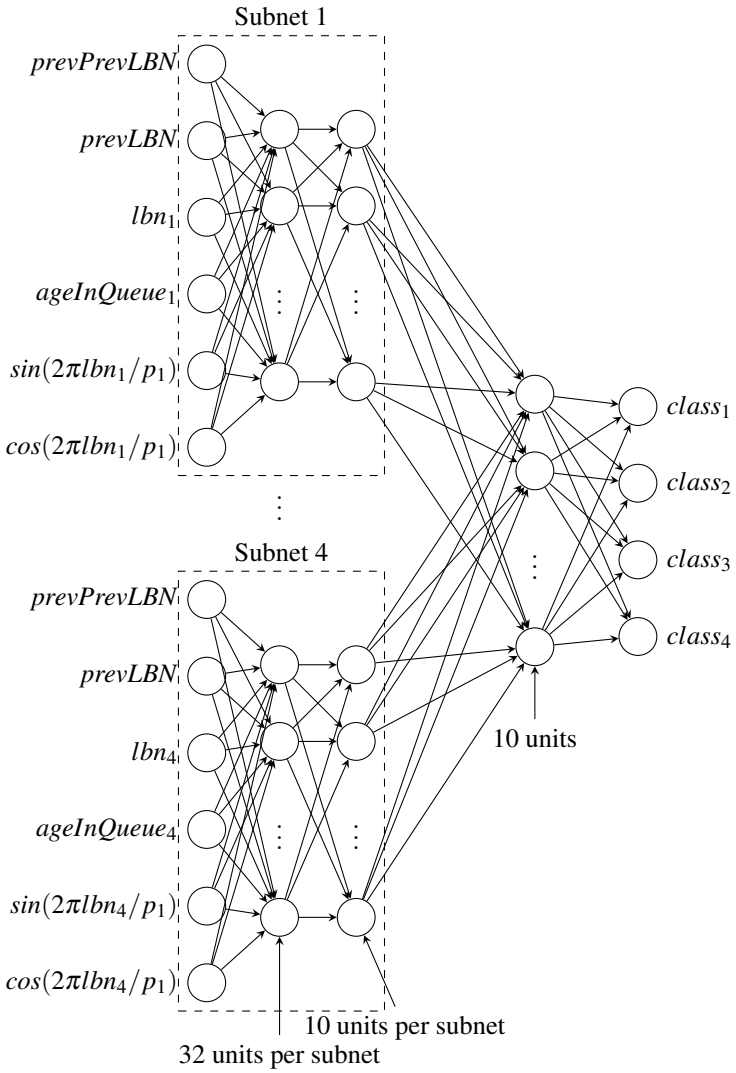


FIG. 4.3. *Network architecture for scheduling prediction when subnets are used. Note that the weights are identical in each subnet.*

The main dataset is of $N = 32{,}000$ random reads of the first 94 tracks, or 237,631 sectors, which is the first 0.024% of the drive. This corresponds to the first part of the first serpentine, so all tracks have the same size (2528 sectors). (It turns out that tracks within the same zone can have different sizes for this hard drive, because they have different sizes on either side of the platter.)

We also used a dataset of $N = 32{,}000$ random reads of the first 32,712 tracks, or 83,167,036 sectors, which is the first 8.51% of the drive. This corresponds to what may be considered the first zone. The track length mostly alternates between 2528 sectors and 2573 sectors with a period of 653,018 sectors.

For positioning time, decision trees were tested with WEKA [11]. In all cases, the minimum leaf weight (minNum) was set to 0. When using bagging, the bag size was set to 100.

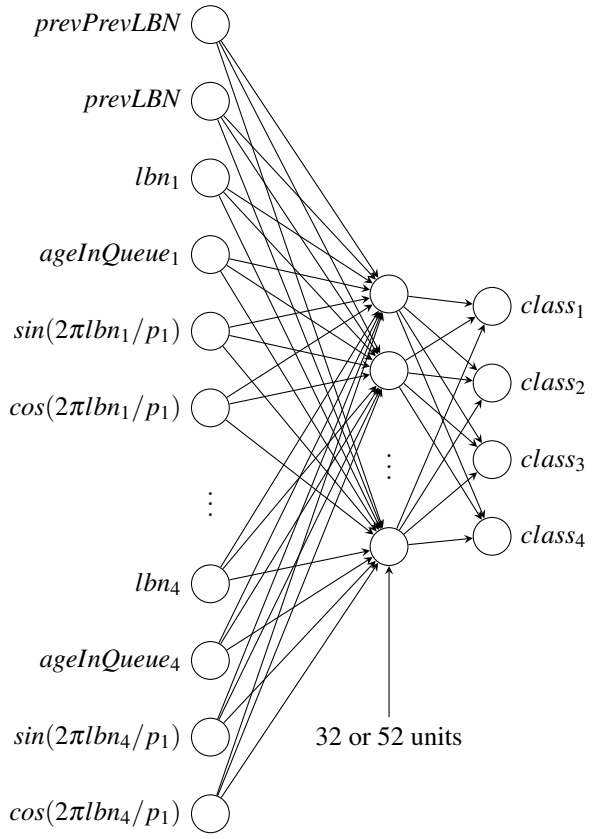


FIG. 4.4. *Network architecture for scheduling prediction when subnets are not used.*

To keep numbers small for the scheduling problem, we tested with a queue size of 4, even though the maximum queue length for our drive is 32. Error is reported as the percentage of instances improperly classified. A constant predictor that chooses the most popular index every time would have an error of 46.9%. The input vectors consisted of the LBNs of the queue entries, the ages of the queue entries, and the LBNs of the two previously processed requests.

For the scheduling problem, decision trees were built with Avatar [13]. This software has the advantage of automatically setting the bag size when using bagging. Unfortunately, Avatar does not currently support regression, which is why the decision trees were built with WEKA for the positioning time problem. Avatar was run on the dataset of random reads over the first part of the first serpentine using:

```
crossvalfc -o avatar --folds=10 -f out --bagging
--no-save-trees --output-accuracies
--output-confusion-matrix --output-predictions
--use-stopping-algorithm.
```

Neural nets were also tested for scheduling prediction, both with and without subnets. The output was a 1-of-k encoding, where the target value for the correct class was 1 and the others were 0.

We trained and tested each configuration five times and reported the mean and standard deviation. For tests with random periods, each run used different random periods between 0 and 5000 sectors.

6. Results. For reference, we ran DIG on our test hard disk drive. From the DIG data, the track length at the beginning of the drive is 2528 sectors, and the skew is 1.1908 ms, which corresponds to 361.37 sectors (given the rotation time of 8.33 ms). We expected to see a dominant period of $2528 - 361.37 = 2166.63$ sectors, which over 237,631 sectors is a frequency of $237{,}631/2166.63 = 109.68$. From the Fourier analysis, the actual dominant period is $237{,}631/107.38 = 2213.05$ sectors, which is close to our prediction of 2166.63.

Positioning time errors are listed in table 8.1, and scheduling errors are listed in table 8.2.

The lowest error achieved with neural nets was 0.830 ms.

When using decision trees, adding the sines and cosines as additional features reduced the error noticeably. Bagging the decision trees reduced error further, but only when the periodic information was included. The lowest error achieved with decision trees was 1.1906 ms.

When using ensembles of decision trees for the scheduling problem, the overall voted error was 32.2%. When adding sines and cosines of the LBNs of the queue entries, the error fell to 20.3%. With neural nets, the lowest error without period information was 33.6%, and with period information it fell to 5.5%.

Most of the time, neural net tests with random periods fared worse than tests with no period information. However, one test with random periods performed much better, although not as well as tests with the correct period information. This appears to have been caused by one of the random periods being nearly equal to twice the most important period.

7. Conclusions. Neural nets achieved lower error than decision trees in all tests. We suspect this is caused by two things: 1) the structure imposed by weight sharing cannot be implemented with decision trees, and 2) many important features are individually uncorrelated with the output, which decision trees do not learn from well.

Adding periodic features requires only a mild assumption and improves multiple machine learning algorithms significantly. This approach is likely to be a crucial part of behavioral modeling of hard disk drive performance.

8. Future work. The analysis needs to be extended to an entire hard disk drive. Other types of devices also need to be tested, such as SSDs and RAID arrays.

One large assumption made by this approach is that positioning time is a meaningful and useful concept. For other block storage devices, this may not be the case, and removing this assumption would be useful future work. Generalizing, we may predict the latency (not positioning time) of a request given k previous requests. The neural net is easily extended to predict $f(x_1, x_2, ..., x_k) = h(g(x_1), g(x_2), ..., g(x_k))$ in time linear to k. The difficulty lies in finding useful periods to feed the neural net. One possibility is to generalize the diagonal search. Previously, we searched $u = -v$. Renaming $u = \xi_1$ and $v = \xi_2$, this can be expressed as a line through the origin normal to $\sum_i c_i \xi_i = 0$ with $c_1 = 1$, $c_2 = -1$. An intuitive expansion is all lines through the origin normal to some hyperplane $\sum_i c_i \xi_i = 0$ where $c_i \in \{-1, 0, 1\}$. This yields 3^k lines, but this is still far better than searching an entire k-dimensional space.

8.1. Other ideas. Another way to approach the Fourier analysis is to use a search-based method, rather than scanning uniformly. Assuming the number of strong frequencies is small, this should be much faster. Unfortunately, the methods we found required first performing a convolution, and it's not clear if performing a convolution on sparse data is possible or meaningful.

REFERENCES

[1] J. S. BUCY, J. SCHINDLER, S. W. SCHLOSSER, G. R. GANGER, AND CONTRIBUTORS, *The DiskSim Simulation Environment Version 4.0 Reference Manual*, Carnegie Mellon University, Pittsburgh, PA, May 2008.

	Configuration	Error (ms)
Decision trees	no periods, without bagging	2.075 ± 0.014
	no periods, with bagging	2.067 ± 0.001
	6 random periods, without bagging	2.019 ± 0.013
	6 random periods, with bagging	2.015 ± 0.013
	6 periods, without bagging	1.649 ± 0.154
	6 periods, with bagging	1.123 ± 0.009
Neural nets	no periods, without subnets	2.014 ± 0.034
	no periods, with subnets	2.012 ± 0.019
	6 random periods, without subnets	1.924 ± 0.176
	6 random periods, with subnets	1.992 ± 0.059
	6 periods, without subnets	0.954 ± 0.052
	6 periods, without subnets, 52 hidden	0.926 ± 0.027
	6 periods, with subnets	0.830 ± 0.031

TABLE 8.1
Average RMS errors for positioning time predictions.

	Configuration	Error (%)
Decision trees	no periods, without bagging	33.11 ± 1.02
	no periods, with bagging	32.20 ± 0.64
	6 random periods, without bagging	34.33 ± 0.01
	6 random periods, with bagging	33.42 ± 0.95
	6 periods, without bagging	31.24 ± 0.70
	6 periods, with bagging	20.50 ± 1.04
Neural nets	no periods, without subnets	38.80 ± 1.91
	no periods, with subnets	33.60 ± 0.29
	6 random periods, without subnets	42.59 ± 0.55
	6 random periods, with subnets*	38.26 ± 14.80
	6 periods, without subnets	10.40 ± 2.69
	6 periods, with subnets	5.50 ± 0.75

TABLE 8.2
*Average scheduling error. *One run had an error of 11.80%, and the others averaged 44.88%.*

[2] C. DAI, G. LIU, L. ZHANG, AND E. CHEN, *Storage device performance prediction with hybrid regression models*, in PDCAT'12, Beijing, China, December 2012.

[3] D. ELIZONDO, G. HOOGENBOOM, AND R. MCCLENDON, *Development of a neural network model to predict daily solar radiation*, Agricultural and Forest Meteorology, 71 (1994), pp. 115–132.

[4] S. E. FAHLMAN AND C. LEBIERE, *The cascade-correlation learning architecture*, Tech. Rep. CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, February 1990.

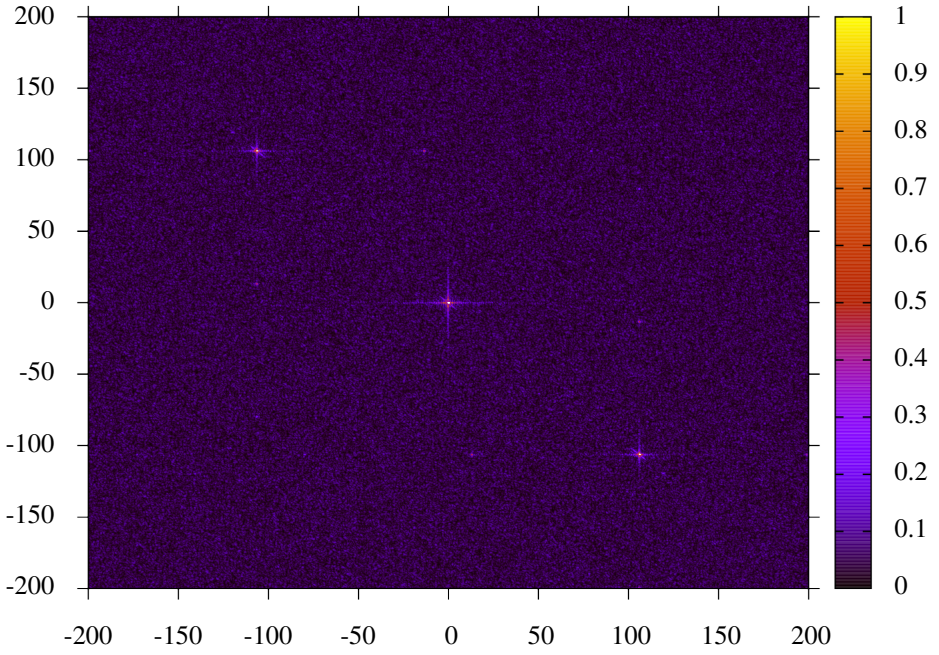


FIG. 8.1. *Full 2D Fourier spectrum for the first* 237,631 *sectors out to* 200×200*, which corresponds periods of at least 1188.155 sectors. Plot is clipped to magnitude 1 to show detail, but central spike goes up to 8.6, and other diagonal spikes go up to 3.9.*

[5] S. FERRARI AND R. F. STENGEL, *Smooth function approximation using neural networks*, Neural Networks, IEEE Transactions on, 16 (2005), pp. 24–38.

[6] J. GARCIA, L. PRADA, J. FERNANDEZ, A. NUNEZ, AND J. CARRETERO, *Using black-box modeling techniques for modern disk drives service time simulation*, in Simulation Symposium, 2008. ANSS 2008. 41st Annual, april 2008, pp. 139 –145.

[7] M. GARDNER AND S. DORLING, *Neural network modelling and prediction of hourly NO_x and NO_2 concentrations in urban air in London*, Atmospheric Environment, 33 (1999), pp. 709 – 719.

[8] J. GIM AND Y. WON, *Extract and infer quickly: Obtaining sector geometry of modern hard disk drives*, Trans. Storage, 6 (2010), pp. 6:1–6:26.

[9] A. GUEZ AND Z. AHMAD, *Solution to the inverse kinematics problem in robotics by neural networks*, in Neural Networks, 1988., IEEE International Conference on, IEEE, 1988, pp. 617–624.

[10] M. T. HAGAN AND M. B. MENHAJ, *Training feedforward networks with the Marquardt algorithm*, Neural Networks, IEEE Transactions on, 5 (1994), pp. 989–993.

[11] M. HALL, E. FRANK, G. HOLMES, B. PFAHRINGER, P. REUTEMANN, AND I. H. WITTEN, *The weka data mining software: an update*, SIGKDD Explor. Newsl., 11 (2009), pp. 10–18. `http://www.cs.waikato.ac.nz/~ml/weka/`.

[12] S. JAGANNATHAN AND F. L. LEWIS, *Multilayer discrete-time neural-net controller with guaranteed performance*, Neural Networks, IEEE Transactions on, 7 (1996), pp. 107–130.

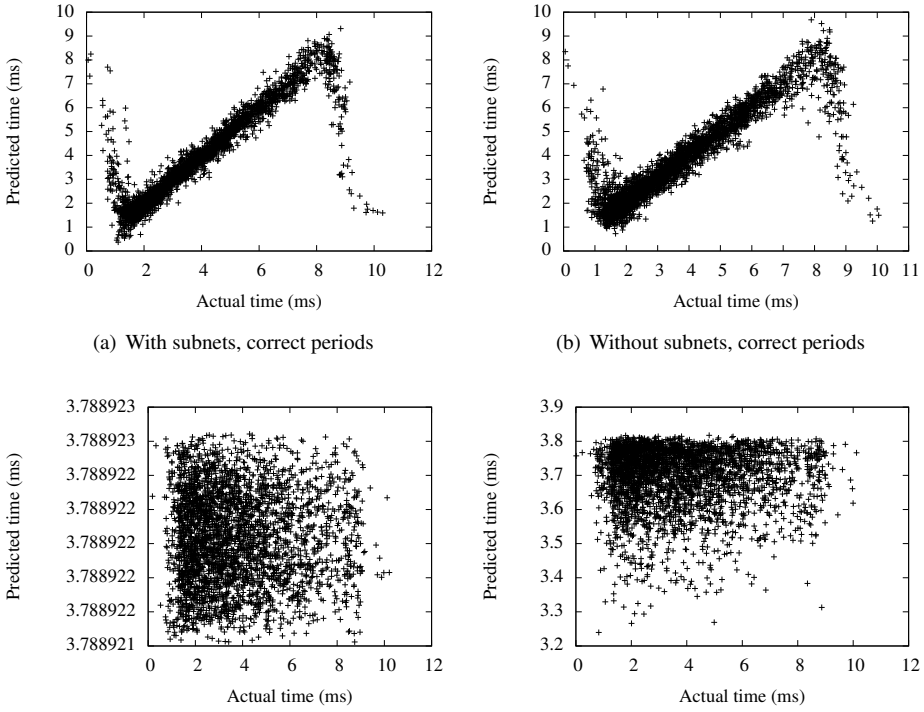
[13] P. KEGELMEYER, *Avatar tools: Home*. `http://www.ca.sandia.gov/avatar/`.

[14] T. KELLY, I. COHEN, M. GOLDSZMIDT, AND K. KEETON, *Inducing models of black-box storage arrays*, Technical Report HPL-2004-108, HP Laboratories, Palo Alto, CA, June 2004.

[15] L. KINDERMANN, A. LEWANDOWSKI, AND P. PROTZEL, *A framework for solving functional equations with neural networks*, in Proceedings of Neural Information Processing (ICONIP2001), vol. 2, Fudan University Press, Shanghai, 2001, pp. 1075–1078.

[16] K. KOSANOVICH, A. GURUMOORTHY, E. SINZINGER, AND M. PIOVOSO, *Improving the extrapolation capability of neural networks*, in Intelligent Control, 1996., Proceedings of the 1996 IEEE International Symposium on, IEEE, 1996, pp. 390–395.

[17] J. KWON, B. COIFMAN, AND P. BICKEL, *Day-to-day travel-time trends and travel-time prediction from loop-detector data*,



(a) With subnets, correct periods

(b) Without subnets, correct periods

(c) With subnets, no periods

(d) With subnets, random periods

FIG. 8.2. *Predicted versus actual times. Note that a diagonal line is ideal.*

Transportation Research Record: Journal of the Transportation Research Board, 1717 (2000), pp. 120–129.

[18] A. S. LEBRECHT, N. J. DINGLE, AND W. J. KNOTTENBELT, *A performance model of zoned disk drives with I/O request reordering*, in Proceedings of the 2009 Sixth International Conference on the Quantitative Evaluation of Systems, QEST '09, Washington, DC, USA, 2009, IEEE Computer Society, pp. 97–106.

[19] N. LIU, J. COPE, P. CARNS, C. CAROTHERS, R. ROSS, G. GRIDER, A. CRUME, AND C. MALTZAHN, *On the role of burst buffers in leadership-class storage systems*, in MSST/SNAPI 2012, Pacific Grove, CA, April 16 - 20 2012.

[20] M. P. MESNIER, M. WACHS, R. R. SAMBASIVAN, A. X. ZHENG, AND G. R. GANGER, *Modeling the relative fitness of storage*, in SIGMETRICS 2007, 2007.

[21] G. NOONE AND S. D. HOWARD, *Investigation of periodic time series using neural networks and adaptive error thresholds*, in Neural Networks, 1995. Proceedings., IEEE International Conference on, vol. 4, IEEE, 1995, pp. 1541–1545.

[22] B. E. ROSEN, *Ensemble learning using decorrelated neural networks*, Connection Science, 8 (1996), pp. 373–384.

[23] Y. SHANG AND B. W. WAH, *Global optimization for neural network training*, Computer, 29 (1996), pp. 45–54.

[24] J. M. SOPENA, E. ROMERO, AND R. ALQUEZAR, *Neural networks with periodic and monotonic activation functions: a comparative study in classification problems*, in Artificial Neural Networks, 1999. ICANN 99. Ninth International Conference on (Conf. Publ. No. 470), vol. 1, IET, 1999, pp. 323–328.

[25] D. F. SPECHT AND P. SHAPIRO, *Generalization accuracy of probabilistic neural networks compared with backpropagation networks*, in Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on, vol. i, 1991, pp. 887–892 vol.1.

[26] M. WANG, K. AU, A. AILAMAKI, A. BROCKWELL, C. FALOUTSOS, AND G. R. GANGER, *Storage device performance prediction with CART models*, in MASCOTS 2004, 2004.

[27] K.-W. WONG, C.-S. LEUNG, AND S.-J. CHANG, *Use of periodic and monotonic activation functions in multilayer feedforward neural networks trained by extended Kalman filter algorithm*, in Vision, Image and Signal Processing, IEE Proceedings-, vol. 149, IET, 2002, pp. 217–224.

[28] Z. ZAINUDDIN AND O. PAULINE, *Function approximation using artificial neural networks*, WSEAS Transactions on Mathe-

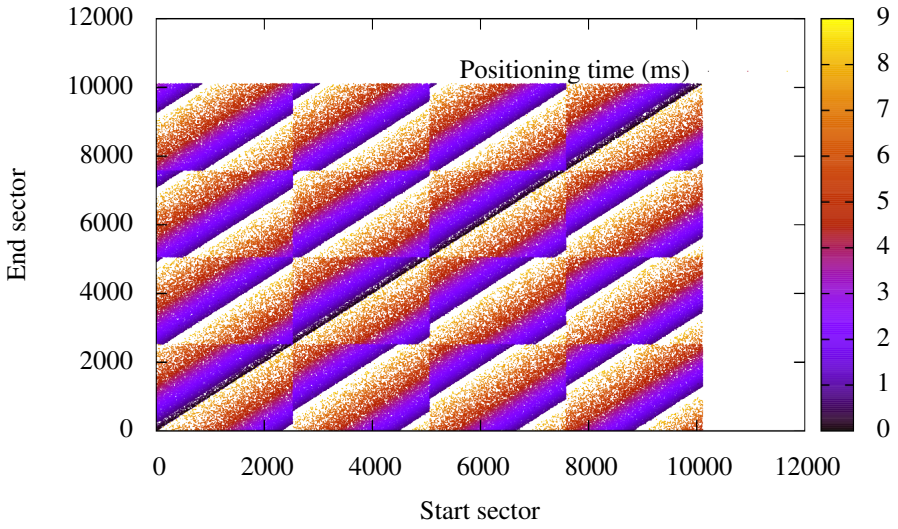


FIG. 8.3. *Positioning time over the first 4 tracks of a hard drive.*

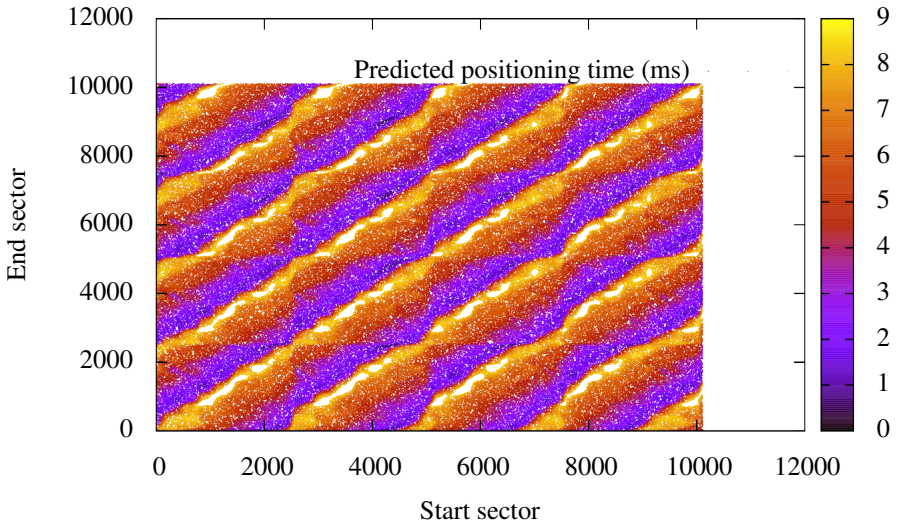


FIG. 8.4. *Predicted positioning time over the first 4 tracks of a hard drive.*

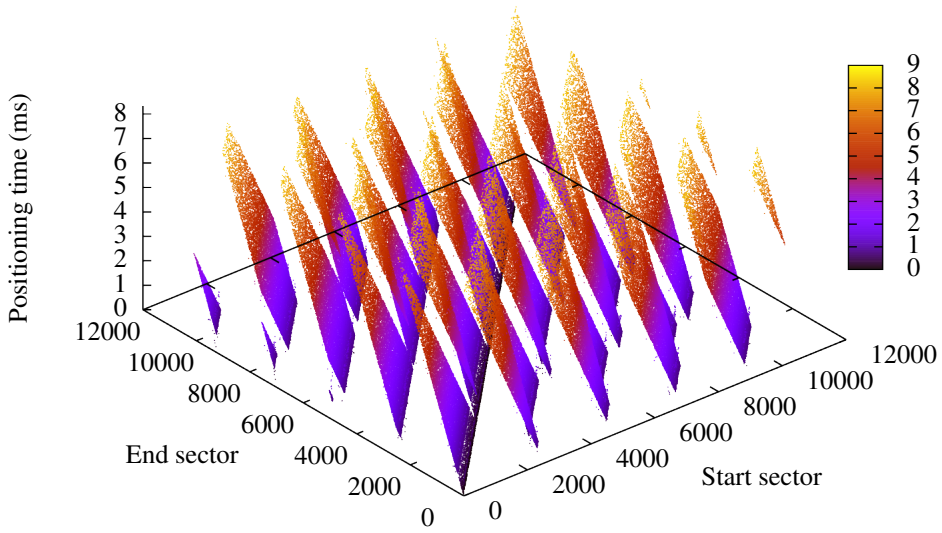


FIG. 8.5. *Positioning time over the first 4 tracks of a hard drive.*

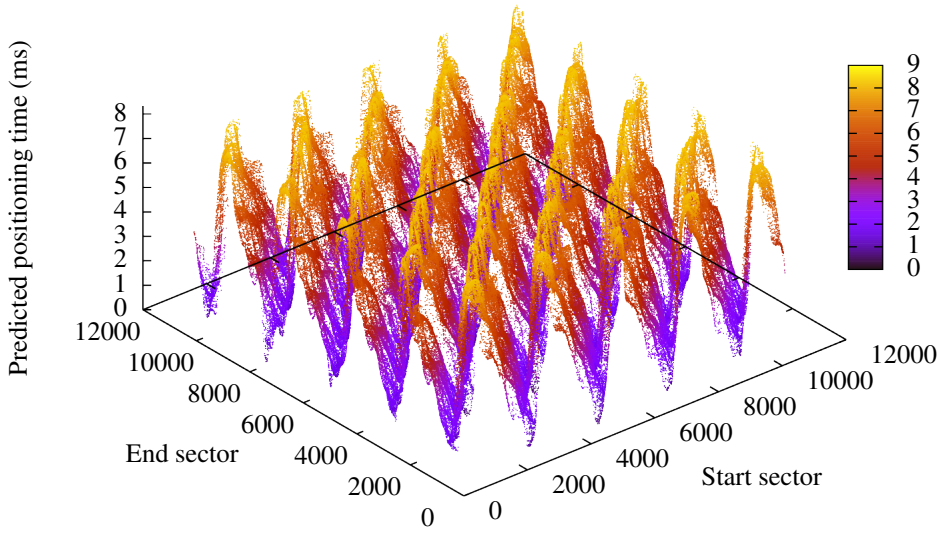


FIG. 8.6. *Predicted positioning time over the first 4 tracks of a hard drive.*

matics, 7 (2008), pp. 333–338.

Computational Applications

Articles in this section discuss the use of computational techniques in physical simulations. *Hanks and Robinson* investigate the ALEGRA shock hydrocode algorithms using an exact free surface jet flow solution. They use a detailed and challenging test to explore the strengths and weaknesses in ALEGRA and provide a summary of verification results. *Merrell and Robinson* review the theory behind ALEGRA's spatial variation capabilities and the contributions to ALEGRA software. They explain the significant changes made to the ALEGRA code to increase the usability and coherence of ALEGRA's spatial variation capability. *Wolf and Bettencourt* describe a Particle-In-Cell method based on a unconditionally stable wave-equation solver allowing much larger timesteps than previously allowed. They apply the method to 1D electrostatic test problems and show that the results agree with the theory.

S. Rajamanickam
M.L. Parks
S.S. Collis

July 22, 2014

INVESTIGATION OF ALEGRA SHOCK HYDROCODE ALGORITHMS USING AN EXACT FREE SURFACE JET FLOW SOLUTION

BRADLEY W. HANKS* AND ALLEN C. ROBINSON[1]

Abstract. Computational testing of the general purpose arbitrary Lagrangian-Eulerian shock physics code, ALEGRA, is presented using an exact solution that is very similar to a shaped charge jet flow. The solution is a steady, isentropic, subsonic free surface flow with significant compression and release and is provided as a steady state initial condition. Ideally, there should be no shock and no entropy production throughout the problem. The purpose of this test problem is to present a detailed and challenging computation which provides evidence for algorithmic strengths and weaknesses in ALEGRA. The work is stored in a permanent testing framework intended to be used to guide and enable future algorithmic improvements in the spirit of test-driven development processes.

1. Introduction. Shock hydrocodes can be used to model material interactions at very high pressures due to high velocity impact. Modeling methodologies may perform well under many conditions but fail catastrophically in other cases. This is especially true for highly complex general purpose modeling tools. Due to the importance and breadth of situations a code must handle, constant testing is required for improvement of the mathematical algorithms and their implementation. This type of test-driven development ensures continuous improvement and quality maintenance of a code. Verification and validation are two separate and distinct aspects of testing a simulation code. Verification compares the results of the code against known properties of the mathematical equations. Validation involves comparison of the results against experimental data for determination of the validity of the equations for the purposes intended. The results of such testing guide necessary additions or improvements of the code. The theory and evaluation of an exact free surface compressible jet flow solution which closely mimics that of a shaped charge jet is detailed in a 2002 SAND report [15]. This work discusses a summary of the verification results obtained using ALEGRA, an arbitrary Lagrangian-Eulerian shock code, with this exact solution similar to a shaped charge jet flow. A complete report of the verification work of ALEGRA is available in a 2014 SAND report entitled, *"Investigation of ALEGRA Shock Hydrocode Algorithms using an Exact Free Surface Jet Flow Solution"* [6].

ALEGRA is an arbitrary Lagrangian-Eulerian (ALE) shock physics hydrocode that has been developed at Sandia National Laboratories since 1990 [1]. The code is designed for modeling shock waves and has the ability to handle complex geometries with multiple materials. ALEGRA can simulate a wide variety of scenarios involving shocks at high pressures.

A conical shaped charge includes a cylinder packed with a high explosive. A cone shape is hollowed out of the explosive and replaced with a metal liner. Upon detonation of the explosive material, the conical liner is collapsed forming a jet of metal. A shaped charge jet may be idealized during the quasi-steady collapse phase with a steady compressible fluid model. Key features in this model include large velocity gradients in small spatial regions as well as very large strains in a steady subsonic isentropic free-surface flow. The features combine to generate computational difficulties with the shaped charge jet test problem. The shaped charge jet is very difficult to model correctly by either a Lagrangian finite element code or an Eulerian code. Lagrangian codes tend to experience severe deformation in the jet leading to a breakdown of the numerical method due to element distortion. The Eulerian codes may have difficulty with interfaces and excessive heating of jet material. This serves as an excellent test problem for ALEGRA executing in Eulerian mode where the nodes move based on the velocity of the material (Lagrangian) and then the nodes are moved back to their original positions and quantities are remapped back onto this mesh (Eulerian).

Expected numerical issues may include unrealistic temperature diffusion into the liner from the ex-

*Brigham Young University, hanksb@byu.edu
[1]Sandia National Laboratories, acrobin@sandia.gov

plosive products, unphysical numerical exchange of kinetic energy to internal energy and heating due to artificial viscosity terms in high compression rate shockless processes [8, 11, 17]. Variations in numerical algorithms can produce dramatic differences in estimates of internal energy and temperature. The numerical difficulties may be particularly acute when the flow to be computed is isentropic. If confidence is to be placed in calculations which purport to include advanced material models that are highly dependent on temperature, it is necessary to develop reliable numerical methods and practical calculational rules of thumb to deal with the shaped charge jet problem in the case of simple hydrodynamic material modeling. The proper application of advanced material modeling in shaped charge simulations depends upon proper energy partitioning in the numerical method. In particular it may be difficult for a numerical method to distinguish a rapid shockless transition from a true shock which is to be captured by the numerical method.

This report is concerned with an exact steady solution imported into ALEGRA to test its ability to correctly model shockless high-strain-rate isentropic subsonic flow. The steady subsonic isentropic flow is a very complex and challenging simulation for a general purpose shock code like ALEGRA. The desired results are that the entropy remains constant at all points while the temperature of any material point moves toward the free-stream value as it flows into the fully released free-stream jet. The principal strategy and purpose for this work in a test-driven development environment is to discover inconsistencies and anomalies that provide direction as to which classical algorithms or newly implemented and proposed advanced algorithms are proper candidates for acceptance as default algorithms or whether any of these algorithms need additional improvement and development.

2. Evaluation of the Exact Solution. The conical shaped charge jet has been reasonably modeled in a gross engineering sense for years by the assumption that the jet collapse process is approximately a steady state in the frame of reference of the collapse point and that free-surface jet theory can be applied [2].

Steady compressible subsonic plane and axis-symmetric free surface jet flows may be effectively calculated with specialized finite difference codes employing boundary fitting coordinate systems or by computing in the hodograph plane [5]. By restricting to two-dimensional plane flows in hodograph coordinates, significant progress can be made analytically. The hodograph plane uses velocity and flow angle, (q,θ), as independent variables. Conservation of mass implies the existence of a stream function, ψ, such that $d\psi = -\rho v dx + \rho u dy$ represents the mass flux across a differential line element from left to right. Assuming a one-to-one mapping between the physical plane (x,y) and the hodograph or velocity-angle space, (q,θ) with $(u,v) = (q\cos\theta, q\sin\theta)$, one obtains equations for the variation of the stream function and velocity potential in terms of q and θ. Thus

$$dz = dx + idy = \frac{e^{i\theta}}{q}\left(d\phi + \frac{i}{\rho}d\psi\right) . \tag{2.1}$$

Since dz is a perfect differential one obtains after considering that ϕ and ψ are functions of q and θ and equating the mixed derivatives of z with respect to q and θ, the equations

$$\frac{\partial\phi}{\partial\theta} = \frac{q}{\rho}\frac{\partial\psi}{\partial q} \qquad\qquad \frac{\partial\phi}{\partial q} = -\frac{(1-M^2)}{q\rho}\frac{\partial\psi}{\partial\theta} . \tag{2.2}$$

Elimination of ϕ leads to an equation for the stream function

$$q^2\frac{\partial^2\psi}{\partial q^2} + q(1+M^2)\frac{\partial\psi}{\partial q} + (1-M^2)\frac{\partial^2\psi}{\partial\theta^2} = 0 . \tag{2.3}$$

This is termed the Chaplygin equation for the stream function. It is a separable linear second order equation whose coefficients depend only on the speed q. This equation possesses separable solutions of the form

$\psi = \psi_n(q)e^{in\theta}$. In the case of the isentropic ideal gas relation, Chaplygin noted solutions of the form

$$\psi = \tau^{n/2} F_n(\tau) e^{in\theta} = \psi_n(\tau) e^{in\theta} \tag{2.4}$$

where

$$\tau = (q/q_{max})^2 = (\gamma - 1)q^2/2 \tag{2.5}$$

and the $F_n(\tau)$ are particular Gauss hypergeometric functions.

These solutions can be used to solve certain problems of a particular form that arise frequently in free surface flow theory. The original ideas and procedures are due to Chaplygin who solved the problem of a plane jet emerging from a slot in a wall [4]. The Chaplygin procedure takes a solution of the incompressible problem and provides a similar subsonic compressible solution. The incompressible wall jet solution for this problem can be determined by standard complex variable techniques [3, 7]. The incompressible complex potential, $W = \phi + i\psi$, is given by

$$\begin{aligned} W(\Omega) = (q_1/\pi) \Big\{ & \log(1 + \Omega e^{i\beta}) + \log(1 + \Omega e^{-i\beta}) \\ & -(1 - \cos\beta)\log(1 - \Omega) - (1 + \cos\beta)\log(1 + \Omega) \} \end{aligned} \tag{2.6}$$

where $\Omega = (q/q_1)e^{-i\theta}$ is the incompressible velocity in complex form. Another representation for this solution may be given by expanding each of the log functions in a Taylor series about $\Omega = 0$. The $n = 1$ terms in each series sum exactly to zero as a consequence of the required mass and momentum balance and thus do not appear. The Chaplygin procedure for writing a corresponding subsonic compressible solution from an incompressible solution is to make the correspondence

$$\left(\frac{q}{q_1}\right)^n \Rightarrow \frac{\psi_n(\tau)}{\psi_n(\tau_1)} \tag{2.7}$$

in the formula for the stream function ψ where τ_1 is the value of τ on the free streamlines. Thus the stream function for compressible flow is

$$\begin{aligned} \psi = ((\rho_1 q_1)/\pi) \Bigg\{ & \sum_{n=2}^{\infty} \frac{1}{n} \frac{\psi_n(\tau)}{\psi_n(\tau_1)} \sin n(\theta - \beta + \pi) \\ & + \sum_{n=2}^{\infty} \frac{1}{n} \frac{\psi_n(\tau)}{\psi_n(\tau_1)} \sin n(\theta + \beta - \pi) \\ & - (1 - \cos\beta) \sum_{n=2}^{\infty} \frac{1}{n} \frac{\psi_n(\tau)}{\psi_n(\tau_1)} \sin n\theta \\ & + (1 + \cos\beta) \sum_{n=2}^{\infty} \frac{1}{n} \frac{\psi_n(\tau)}{\psi_n(\tau_1)} \sin n(\theta - \pi) \Bigg\} . \end{aligned} \tag{2.8}$$

An extra factor of ρ_1 is applied in the above formula since the stream function in the compressible case represents a mass flux. The convergence theory for this series, called a Chaplygin series, has been described by Sedov [18] and a summary of the theory is given in a previous report [15].

Integration to obtain the physical plane may be accomplished in several ways since the physical plane is independent of integration path in the (q, θ) plane. The evaluation code actually implements two different approaches which utilize Equations 2.8, 2.1 and 2.2 to obtain a complex series for $\partial z/\partial q$ and $\partial z/\partial \theta$. In the first $\partial z/\partial q$ is evaluated for each point (q, θ) and then $z(q, \theta)$ is obtained by numerical integration with respect to q subject to $z(0, \theta) = 0$ using the trapezoidal rule. The second technique is to integrate $\partial z/\partial \theta$ with respect to θ analytically and sum the resultant series of integrated terms to obtain the position $z(q, \theta)$.

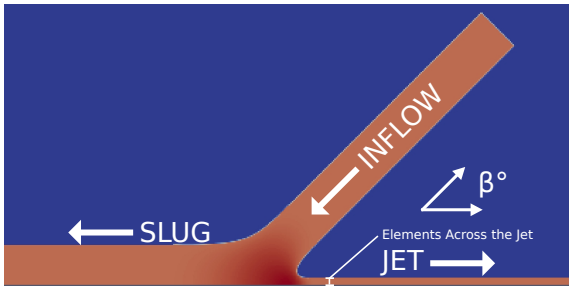


FIG. 3.1. *Initial time step of the shaped charge test problem in ALEGRA.*

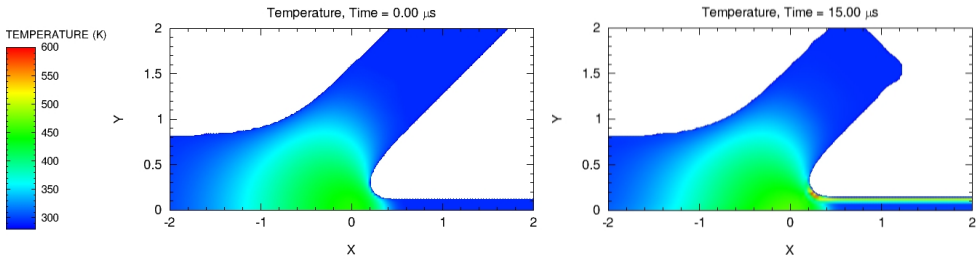
3. Verification of the Exact Solution in ALEGRA. ALEGRA has been upgraded to support a special Mie-Grüneisen equation of state in which the reference curve matches the exact solution on an isentrope. The exact solution is imported into ALEGRA as an initial condition for testing purposes. The test is implemented permanently in the ALEGRA test harness in the spirit of test-driven development and continuous improvement of the numerical algorithms. Since the exact solution is a steady solution, we wish to import this solution as an initial condition. We then expect this solution to be maintained for some time period until the effects of the jet at "infinity" possibly interacting with mesh boundaries become apparent. Once imported, a number of tests are run which are intended to highlight the status of the current numerical algorithms in ALEGRA. Some algorithms are relatively new and others have been in the code for a long time. The results of testing the default settings of ALEGRA, artificial viscosity options, midpoint time integration, DeBar energy advection, the extended finite element method (XFEM), and mixed material/void algorithms are summarized in the full report.

The complete verification report details multiple frames of reference and incident angles for the shaped charge jet solution [6]. This report focuses on the stagnation point frame of reference and a single incident angle, $\beta = 45^\circ$ with the inflow velocity set to .9 of the free stream sound speed. The stagnation point frame of reference provides a simple way of viewing the simulation. In addition to the stationary imported solution state, material in the free stream state is overlaid in the free stream jet regions to create a continuous jetting flow as shown in Figure 3.1.

The testing in ALEGRA focuses on the under-resolved case of the test problem. The under-resolved cases with larger errors provide great insight into algorithmic weaknesses. Mesh resolution is given by the number of elements across the jet as shown in Figure 3.1. When $\beta = 45^\circ$, the typical under-resolved case includes 5 elements across the jet. A simulation at high resolution includes between 9 and 17 elements across the jet.

The results of a "perfect" simulation would not change as time progresses because the analytic solution is for a steady state problem. The material would heat up as it compresses and then release and cool down. The entropy would be constant and uniform throughout the whole problem. It is expected that some initial small waves and oscillations will appear as the simulation begins due to finite resolution and imperfections in the initial state. These oscillations may be expected to diminish and damp out as time progresses. The oscillations may cause fluctuations in dependent variables such as temperature but should be small and reach a quasi-steady state. Once a quasi-steady state is achieved, analysis is performed to view any variations from the analytic solution.

Analysis of the results in the complete verification report is performed using multiple horizontal line-outs for multiple variables which are compared against the analytic solution [6]. This report focuses on a temperature line-out through the elements closest to the jet surface, where the majority of issues are visible.



(a) Time = 0.0 μs (b) Time = 15.0 μs

FIG. 4.1. *Verification testing of exact solution in ALEGRA.*

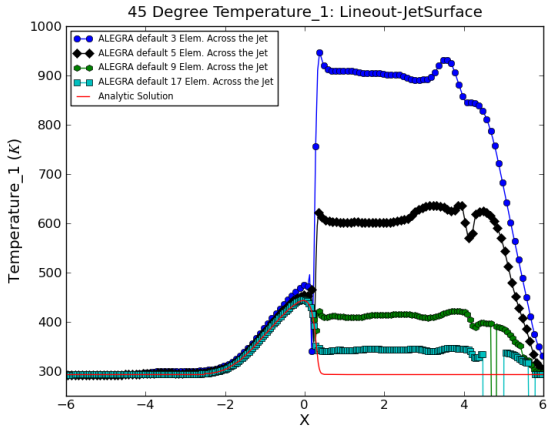


FIG. 4.2. β = 45°*: Temperature line-out on the jet surface for various mesh resolutions.*

The nature of the test problem allows for comparison of later time steps with the initial time step to provide evidence for the correct subsonic isentropic flow.

4. ALEGRA Solution Using Default Settings. As the simulation begins, small oscillations occur near the stagnation point and the corner where the jet is formed. These two areas are expected to be the most difficult because of the high compression and release, and the sharp change in velocity around the corner. These oscillations are not unexpected and are caused by small local interactions as the truncation errors in the numerical solution try to adjust to the initial conditions given by the exact solution. These oscillations are greatly reduced after a few ALEGRA time steps. At the final time step of 15.0 μs, the oscillations that occurred at the beginning have diminished and the effects have been pushed farther along in the jet as more material moves through stagnation point.

Figures 4.1(a) and 4.1(b) show the temperature for $\beta = 45^\circ$ at t = 0.0 μs and t = 15.0 μs respectively. From these plots a more significant rise in temperature is seen along the surface of the jet, including a particular increase in the corner where material flows into the jet. Important to note is the lack of temperature rise along the slug. In the jet, the material at the free surface must turn a sharp corner at constant pressure and the presence of numerical difficulties is perhaps not surprising.

Figure 4.2 plots the temperature line-out on the jet surface at t = 15.0 μs for various mesh resolutions.

A spike in temperature happens as the material leaves the stagnation point region and enters the jet. The nature of the analytic solution is such that after heating and compressing in the stagnation point region, the temperature should decrease to its initial value. At greater computational expense, as the mesh resolution increases, the temperature error decreases along the surface of the jet.

The oscillations from the initial time steps are seen clearly in Figure 4.2. As an example, with a mesh resolution of 5 elements across the jet, from the position $X = 1$ to $X = 2.5$, the temperature is constant at about 600 kelvins which is an error of about 300 kelvins. This region for $1 < X < 2.5$ represents a quasi-steady state that has been reached. For $X > 2.5$, the effects of the initial oscillations in the stagnation point are seen and are pushed out as the test is allowed to progress further in time. Figure 4.2 shows how the error decreases to approximately 50 kelvins with 17 elements across the jet. In continuation, several algorithms under current development in ALEGRA are tested for whether they provide improvements in temperature along the jet surface for the 5 element case.

5. Artificial Viscosity and Time Integration Methods. Artificial viscosity algorithms in ALEGRA were tested for improvement of their ability to correctly handle the shockless isentropic flow. With an inflow velocity just under the sound speed and an initial exact steady state profile which is fully subsonic, no shock waves should be produced. It is possible the high compression at low resolution may appear as a shock to the hydrocode and additional entropy could be added through the artificial viscosity. Advanced algorithms should be more proficient at differentiation between shocks and non-shock compression and control of nonphysical numerical dissipation.

Artificial viscosity works by adding additional dissipation to smooth out the solution near shock discontinuities [20]. The default artificial viscosity seen in the previous section is tested along with two other newer options called the limiter and hyperviscosity. The limiter works to monitor the artificial viscosity and reduce or turn off its effects when the artificial viscosity may be unnecessary. In this case, where no shocks should occur, the limiter should reduce the effects of the artificial viscosity adding entropy. Hyperviscosity is an option designed to be utilized together with the limiter. Hyperviscosity is not applied where the lower order artificial viscosity is applied. The use of hyperviscous dissipation helps to control small-scale oscillations [12].

There are two time integration methods currently available in ALEGRA. The current default is the central-difference time integrator that is second order accurate in space but has only a first order accurate energy equation update in time and exhibits instability in expansion [9, 10]. The alternative predictor-corrector or midpoint method is approximately twice as expensive as the central-difference algorithm in the ALEGRA Lagrangian step but is fully second-order accurate in time and stable. Stability analysis of the midpoint time integrator is discussed in [9] The midpoint method is intended as the future default.

With only 5 elements across the jet, Figure 5.1 shows that the limhyp artificial viscosity algorithm has decreased the temperature along the surface of jet significantly. While it has reduced the excess heating, the temperature remains high in relation to the analytic solution. Though results vary slightly between the two time integration methods, overall the options do not have a strong effect on this test problem.

6. Debar Energy Advection. DeBar Energy Advection is an optional algorithm implemented in ALEGRA which conserves total energy [1, 16]. The DeBar method remaps the kinetic energy, then adds the difference between the remapped kinetic energy and the kinetic energy computed after the momentum remap to the internal energy. The advection option does have the possibility of creating anomalously cold regions or hot regions due to truncation errors but has shown excellent results with some shock simulations. The DeBar method includes an option to limit its full application.

The challenge for the DeBar type of algorithm is to provide for high quality shock simulations while still retaining thermodynamic robustness. The DeBar method is implemented when the code finds a shock

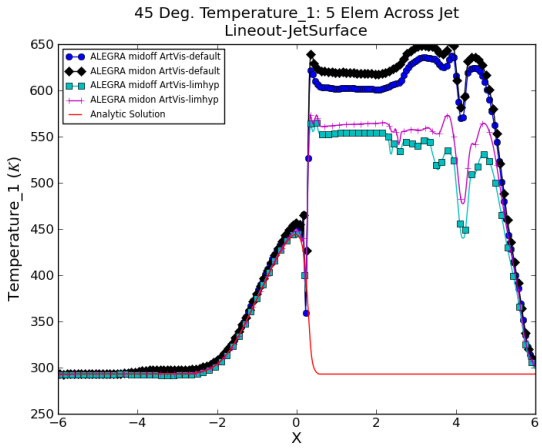


FIG. 5.1. $\beta = 45°$*: Jet surface line-out for temperature showing the effects of midpoint and limited-hyperviscosity artificial viscosity algorithms.*

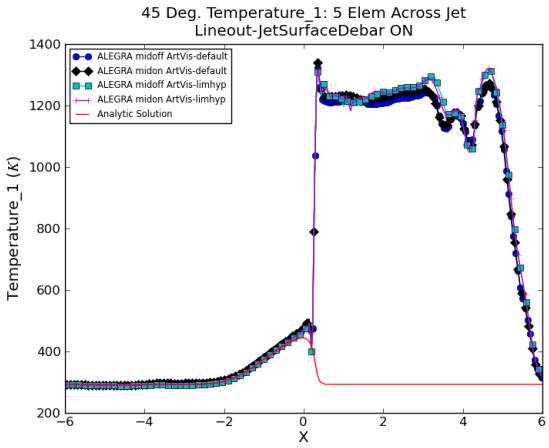


FIG. 6.1. $\beta = 45°$*: Jet Surface line-out for temperature with 5 elements across the jet showing results from DeBar energy advection* $(Q/p = 0)$.

with a Q/p value greater than the value specified in the input deck, where Q represents the artificial viscosity and p represents the pressure. If no value is specified, a default value of 0.0 is utilized.

Large errors appear with the DeBar option where $Q/p = 0.0$ including cooling anomalies along the inflow and heating errors along the jet. Errors became so large for the limhyp option with central-difference time integration that for some mesh resolutions the simulation failed. Figure 6.1 shows the line-out for temperature along the jet surface. Values of $Q/p \approx .1$ slightly improve the results and then continue to improve until the DeBar method is completely off and the results are the similar to the default ALEGRA code shown in Section 4. The plot also shows that none of the various options seem to make much difference with the DeBar method implemented.

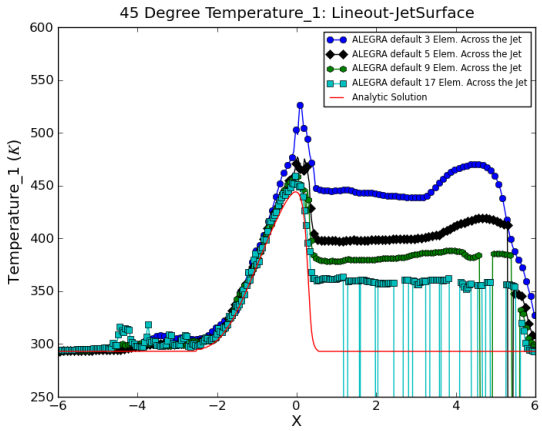


FIG. 7.1. $\beta = 45°$*: Temperature line-out on the jet surface for the XFEM in ALEGRA. Compare to Figure 4.2.*

The spurious effect that this algorithm causes at the free surface on this relatively benign quasi-isentropic flow may be indicative of an implementation error of some sort for mixed cells. A thorough review of this algorithm is required.

7. Extended Finite Element Method (XFEM). The extended finite element method (XFEM) in ALEGRA was developed in order to achieve better sub-cell resolution for multi-material elements. It is conceivable that the issues that occur in multi-material elements in general and specifically on a material void interface in this jet problem may be totally sidestepped using this algorithm. The standard finite element method (FEM) basically binds two materials together and issues may occur when two materials experience contact, slide past one another, or must release from each other. This is caused by the continuous velocity field of the FEM [19]. One may speculate that the anomalous heating issues visible in the previous tests are caused by the interface algorithms between a material and void. In the shaped charge test problem this interface lies between the void background and copper jet where the significant anomalous heating of the jet surface material occurs. Originally XFEM was used to model crack propagation and has since been implemented in ALEGRA as a new algorithm for dealing with multi-material elements [14, 21]. The key advances of this algorithm are additional kinematic degrees of freedom in a single mixed material element and it seems appropriate to test this emerging ALEGRA capability on this test problem.

The XFEM has a significant effect on the shaped charge test problem at the jet surface for low mesh resolution. Figure 7.1 shows the results and may be compared with Figure 4.2. When comparing the highest resolution mesh, the default ALEGRA matches the analytic solution better than the XFEM. It is the low resolution plots with only 3 and 5 elements across the jet where the XFEM in ALEGRA greatly reduces the heating along the jet surface.

A subtle instability is a new problematic issue seen in the XFEM simulations. The issue is represented by the development of a coarseness or waves seen in the plots. The default settings of ALEGRA are quite stable and smooth while the XFEM plot exhibits instabilities that seem to worsen with increasing resolution. These waves are clearly visible in Figure 7.1 when compared to Figure 4.2. The vertical lines present in the XFEM plot represent small break-ups of the copper jet along its surface caused by these instabilities. After further analysis, these instabilities appear to be related in some way to the relatively new second order remap code associated with the XFEM algorithm. Reducing the remap to first order removes the waves and

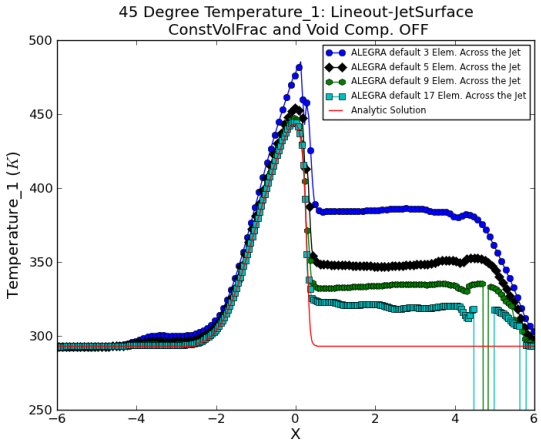


FIG. 8.1. $\beta = 45°$*: Temperature line-out on the jet surface for the CVFA without void compression. Compare with Figure 4.2.*

instabilities yet the second order remap is essential for reducing the anomalous heating along the jet surface.

8. Multi-Material Algorithms and Void Compression. The XFEM simulation from Section 7 appears to be the best for reducing the heat along the surface of the jet. This suggests that an algorithm required by XFEM may be related to this effect of reducing the anomalous heating. The constant volume fraction algorithm (CVFA) and void compression algorithm are possible causes for the heat reduction seen in XFEM. The CVFA was replaced by the isentropic multi-material algorithm (IMMA) as the default in ALEGRA [13]. The CVFA may be used with or without the void compression algorithm while IMMA forces the use of void compression. These algorithms alter the way void interacts with material in ALEGRA. This chapter provides further information relative to algorithmic issues that may exist on the material/void interface by reviewing the effects of the CVFA and IMMA with and without the void compression algorithm.

Figure 8.1 shows results for the CVFA with void compression off. All resolutions have reduced surface heating relative to default ALEGRA. This plot reflects similar results seen in Figure 7.1, particularly those of 3 and 5 elements across the jet. Special code modifications in ALEGRA allowed for the IMMA to be used without void compression. The results are essentially identical to the CVFA without void compression for all resolutions. The CVFA and IMMA with void compression show increased temperatures along the jet surface. These results reflect similar trends seen with the default ALEGRA simulations. For results of the CVFA and IMMA with void compression see the complete verification report [6]. The behavior of the plots with void compression off are similar to those shown in Chapter 7 which contains the ALEGRA XFEM results. These plots show that a major factor in the heating of the jet surface is perhaps the void compression algorithm. The CVFA and IMMA without void compression provide smooth solutions, reduced heating along the jet surface, and do not exhibit the subtle oscillations and interface instabilities seen in the XFEM simulations.

9. Conclusions. We have found that importing the exact solution into ALEGRA as an initial condition allows for setting up this quasi-steady state subsonic flow problem in a way which we believe permits for a significant test of the shock capturing and advection algorithms found in ALEGRA on a complex isentropic shockless problem. We found that most of the default algorithms gave reasonable results; however, there are still improvements to be made in the default as well as other proposed algorithms. The shaped charge jet test

problem creates heating anomalies on the jet surface which may be associated with similar thermodynamic issues which appear in other very complex ALEGRA simulations. We thus have provided a concrete test problem for a difficult yet controlled simulation environment. Continued analysis of the simulation results will allow for evidence based improvement of the ALEGRA algorithms.

The entropy should remain constant through the entire simulation because it is a near steady state subsonic isentropic flow. Temperature should increase as the material passes the stagnation point then return to its initial value. As shown, the heating along the jet surface can be very severe. Evidence shows that this is primarily related to the mixed cell algorithms active in cells with partial void. The mixed cell constant volume fraction algorithm and the isentropic multi-material algorithm singled out void compression as the most important algorithmic variant which could be examined further to provide insight. This suggests that heating issues may be unrelated to algorithms such as artificial viscosity and time integration due to their minimal effects on the test problem.

New artificial viscosity algorithms were tested to see if they improved anomalous heating along the edge of the jet. Specific parameters using the limiter and hyperviscosity were selected for the simulation. The new parameters slightly reduced the heating along the jet surface yet results of the limiter and hyperviscosity varied for other line-outs in the jet [6].

In general, time integration did not have a major effect on the accuracy of the simulation. In many cases there was no visible difference between the central difference and midpoint time integrators.

DeBar energy advection was specifically tested for reducing heating on the edge of the jet. In each case, DeBar has the adverse effect of increasing the heating along the jet. For this test problem DeBar also caused heating issues in the slug and some negative temperatures along the inflow. This test case indicates a need for more detailed analysis and improvements of this algorithm as a general purpose algorithm.

The XFEM algorithms appear to robustly run this test problem. However, some sort of anomalous wave structure is created. Heating characteristics along the jet surface are excellent and are similar to results with void compression off.

At lower resolutions a significant factor is the void compression algorithm. In current default settings of ALEGRA, a user is forced to use void compression with the IMMA. Switching to the CVFA without void compression or minor code modifications which allow IMMA to be used without void compression greatly reduce the heating along the jet surface. At higher resolutions these algorithms do not have significant effect.

The default settings in ALEGRA with a highly resolved mesh appear to be the most accurate and consistent results up to this point. All major issues in the simulation occur along the edge of the jet and can be significantly reduced by increasing the mesh resolution. It has been determined that the cooling and heating anomalies in the simulations are primarily related to mixed cells. Cells that include material and void appear to be causing the anomalies and this work recommends a detailed look at current algorithms that interact with mixed material/void elements.

The recommendations given herein are specifically for the version of the ALEGRA repository head code as of August 2013. The test problem presented here is integrated into the ALEGRA test suite. ALEGRA tools are utilized to compute the norm of the difference between the ALEGRA simulation and the exact solution line-outs in the jet. This problem represents a clear example of test driven development principles in which developers have an institutionalized running test case suite against which to improve algorithms. Given success in algorithmic improvements, the test suite tolerances can be immediately reduced and thus provide for a continually protected and improving metric of code quality.

Acknowledgement. The authors which to thank John Niederhaus for providing a helpful review.

REFERENCES

[1] 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.
[2] G. BIRKHOFF, D. P. MACDOUGALL, E. M. PUGH, AND G. I. TAYLOR, *Explosives with lined cavities*, J. Appl. Phys., 19 (1948), pp. 563–582.
[3] G. BIRKHOFF AND E. H. ZARANTONELLO, *Jets, wakes and cavities*, in Applied Mathematics and Mechanics, vol. II, Academic Press, New York, 1957.
[4] S. A. CHAPLYGIN, *On gas jets*, tech. rep., Moscow University, 1902. In Russian, Translated in NACA Technical Memorandum No. 1063, 1944.
[5] L. P. COOK, E. NEWMAN, S. RIMBEY, AND G. SCHLEINIGER, *Sonic and subsonic axisymmetric nozzle flows*, SIAM J. Appl. Math., 59 (1999), pp. 1812–1824.
[6] B. W. HANKS AND A. C. ROBINSON, *Investigation of ALEGRA shock hydrocode algorithms using an exact free surface jet flow solution*, Tech. Rep. SAND2014-0479, Sandia National Laboratories, 2014.
[7] R. R. KARPP, *An exact partial solution to the compressible flow problems of jet formation and penetration in plane, steady flow*, Quart. Appl. Math, (1984), pp. 15–29.
[8] R. R. KARPP AND S. GOLDSTEIN, *Modifications to the hydrodynamic computer code HELP*, Tech. Rep. M-4-1596, Los Alamos National Laboratory, December 1977. Work for U. S. Army Ballistic Research Lab. under Army Project Order N721-77.
[9] E. LOVE, W. J. RIDER, AND G. SCOVAZZI, *Stability analysis of a predictor/multi-corrector method for staggered-grid Lagrangian shock hydrodynamics*, Journal of Computational Physics, 228 (2009), pp. 7543–7564.
[10] E. LOVE AND M. K. WONG, *Lagrangian continuum dynamics in ALEGRA*, Tech. Rep. SAND2007-8104, Sandia National Laboratories, Albuquerque, NM, 2007.
[11] W. F. NOH, *Errors for calculation of strong shocks using an artificial viscosity and an artificial heat flux*, J. Comp. Phys., 72 (1978), pp. 78–120.
[12] W. J. RIDER, E. LOVE, G. SCOVAZZI, AND V. G. WEIRS, *A high resolution Lagrangian method using nonlinear hybridization and hyperviscosity*, Computers & Fluids, 83 (2013), pp. 25–32.
[13] W. J. RIDER, E. LOVE, M. K. WONG, O. E. STRACK, S. V. PETNEY, AND D. A. LABRECHE, *Adaptive methods for multi-material ALE hydrodynamics*, International Journal for Numerical Methods in Fluids, 65 (2011), pp. 1325–1337.
[14] J. ROBBINS AND T. VOTH, *An extended finite element method formulation for modeling the response of polycrystalline materials to dynamic loading*, in AIP Conference Proceedings, vol. 955, December 2007, p. p259.
[15] A. C. ROBINSON, *Evaluation techniques and properties of an exact solution to a subsonic free surface jet flow*, Tech. Rep. SAND2002-1015, Sandia National Laboratories, 2002.
[16] A. C. ROBINSON, J. H. J. NIEDERHAUS, V. G. WEIRS, AND E. LOVE, *Arbitrary Lagrangian-Eulerian 3D ideal MHD algorithms*, International Journal for Numerical Methods in Fluids, 65 (2011), pp. 1438–1450.
[17] J. A. SCHMITT, *Truncation error terms in the kinetic energy calculation in the HELP algorithm and their consequences*, J. Comp. Phys., 35 (1980), pp. 206–228.
[18] L. I. SEDOV, *Two-dimensional Problems in Hydrodynamics and Aerodynamics*, Interscience Publishers, New York, 1965.
[19] E. VITALI AND D. J. BENSON, *An extended finite element formulation for contact in multi-material arbitrary Lagrangian–Eulerian calculations*, International Journal for Numerical Methods in Engineering, 67 (2006), pp. 1420–1444.
[20] J. VONNEUMANN AND R. D. RICHTMYER, *A method for the numerical calculation of hydrodynamic shocks*, Journal of Applied Physics, 21 (1950), pp. 232–237.
[21] T. VOTH, S. J. MOSSO, AND R. KRAMER, *XFEM for multi-material Eulerian solid/hydrodynamics*. SAND2013-7316C. Presented at Multimat 2013 Conference, San Francisco, CA, September 2013.

RANDOM FIELD SUPPORT IN ALEGRA

DAVID P. MERRELL* AND ALLEN C. ROBINSON[1]

Abstract. Spatial variation is easily observed in many engineering materials. This heterogeneity is important for engineering design and modeling. It is desirable that simulation models be capable of representing spatial variation in material parameters or initial thermodynamic state. This paper reviews the theory behind ALEGRA's spatial variation capabilities and the authors' contributions to this aspect of the software are described. Additionally, further desirable enhancements to ALEGRA's spatial variation capabilities are outlined.

1. Introduction. ALEGRA is an arbitrary Lagrangian-Eulerian finite element code [1]. It is capable of simulating physical systems involving several materials, which may move between solid, liquid, gas, and plasma phases. ALEGRA simulations may also incorporate the effects of electromagnetic fields. Furthermore, it is well-suited for modeling physical systems undergoing shock propagation and large distortions. It is frequently tested against analytical solutions and experimental data and strives to provide increasingly realistic and relevant modeling.

It is desirable in ALEGRA to move beyond idealized representations of materials. In reality, materials are heterogeneous, with model parameter values that are dependent on position within the material. It is also desirable to be able to specify more than uniform values for initial temperatures and/or densities; in some circumstances, the initial thermodynamic state of a system could vary randomly throughout its volume, and hence require a statistical description rather than a nominal description. Such an initial state could also be used to seed instabilities for example.

The variability requirements concerning material parameters and initial thermodynamic states will be referred to as "spatial variation" issues. It is desirable to model spatial variation in material properties and in initial densities and temperatures. ALEGRA is equipped with methods to incorporate spatial variation into its simulations. In section 2, we will discuss the mathematical ideas underlying the methods. In section 3, we will describe some improvements that have been made to the methods over the course of the summer of 2013. Lastly, section 4 will outline some desirable future improvements.

2. Theory. Currently, ALEGRA is equipped with two methods for the creation of spatial variation. The first is known as the Space-Filling Curve (SFC) method. The second is known as the Karhunen-Loève Random Field Expansion (KL) method. Each method has its own strengths and rationale for use.

The basic idea behind the SFC Method is to create a unique ordering for the elements in the domain, and then perturb the value of interest (material parameter or initial state) according to some statistical distribution, making it vary spatially. This unique ordering of the elements is important; ALEGRA is intended for parallel computation, and the spatial variation must be created consistently independently of the number of processors used for the analysis. This consistent ordering is ensured in a rather simple (though effective) way; each processor computes the unique ordering independently. In the sense that each processor is doing the same computation independently, this algorithm is not scalable. However, even for large domains with many elements, the computation is very fast and doesn't add to the computational cost in any significant way, allowing it to still be a useful method.

As indicated by its name, the Space-Filling Curve Method uses a space-filling curve to compute this unique ordering. A space-filling curve is a map from $[0,1]$ onto a higher dimensional region such as $[0,1]^2$ or $[0,1]^3$ such that $|\mathbf{x}(s)-\mathbf{x}(t)| < c|s-t|$ where $0 \leq s,t \leq 1$ and c is a constant. In ALEGRA the SFC algorithm creates a box containing the domain under consideration. The algorithm then fills this box with a Hilbert

*Brigham Young University, davidpm111@gmail.com
[1]Sandia National Laboratories, acrobin@sandia.gov

space-filling curve. The unique ordering of the elements is determined from the element positions using an inverse Hilbert curve mapping function and subdivided to give an approximate "aggregate" size. Once this ordering has been created, the algorithm steps through the elements in that order, perturbing the value of interest as it goes. The perturbations are made so that the value of interest will conform to some statistical distribution. At this point, the perturbation can correspond to either a uniform distribution or a Weibull distribution. The uniform distribution is a simple way to perturb parameters and state variables when little is known about their actual distribution in the real system. The uniform perturbations are made according to the following formula:

$$X_{perturbed} = \bar{X} \cdot (1 + \varepsilon(2 \cdot \texttt{random()} - 1)), \tag{2.1}$$

where $\bar{X}$ is the mean value, ε is given by the user, and `random()` is a random number between zero and one.

The Weibull distribution is commonly used in material science. For many materials, the measured strengths of a large sample set are distributed in a Weibull fashion. ALEGRA uses a modified form of the Weibull probability distribution function (PDF) that accounts for the effects of element volume. The difficulty of using this distribution is that several parameters are required to determine the distribution's shape, which would ideally be determined from experiment. These include the Weibull Modulus m, the Modulus Scale Factor m_{sf}, a Reference Volume V_{ref}, and a Volume Exponent k. The Weibull perturbations are made according to this formula:

$$X_{perturbed} = \bar{X} \cdot \left(\frac{V_{ref}}{V}\right)^{\frac{k}{m}} \left(\frac{\ln(\texttt{random()})}{\ln\left(\frac{1}{2}\right)}\right)^{\frac{1}{m \cdot m_{sf}}}, \tag{2.2}$$

where $\bar{X}$ is the median value, V is the volume of the element and `random()` is a random number between zero and one.

Figure 2.1 shows various shapes for the Weibull distribution. Figure 2.2 shows a cylinder that has had a material parameter perturbed by the SFC method. The Weibull method seeks to determine effective material parameters so that probabilistic property variations at scales smaller than the finite mesh size are still effectively modeled. This is why the model depends on volume. For example, such modeling can be used to characterize damage in ceramics [6]. The Weibull theory is part of a large literature related to extreme value statistics [5]. The Weibull methodology is appropriate when the correlation scale of the material heterogeneities are smaller than the finite element size.

When the correlation length of the material heterogeneity is larger than the finite element size another methodology is needed. A random field is a domain $D \subset \mathbb{R}^n$, with a random variable defined at each point within D. There is a rich mathematical theory established for random fields which provide a desirable foundation for describing spatial variation [3, 7].

The KL method in ALEGRA utilizes the Karhunen-Loève (KL) expansion of a random field. The random field $X(\vec{r})$ centered at zero has an associated covariance function $K(\vec{x},\vec{y}) = E(X(\vec{x}),X(\vec{y}))$. The covariance function expresses the correlation between each pair of random variables in the random field. The KL expansion for $X(\vec{r})$ is its eigenfunction series expansion. By "eigenfunctions", we mean $\phi_i(\vec{x})$ satisfying

$$\int_D K(\vec{x},\vec{y})\phi_i(\vec{x})\mathrm{d}\vec{x} = \lambda_i \phi_i(\vec{y}) \tag{2.3}$$

The eigenfunction expansion of $X(\vec{r})$ is then given by the Karhunen-Loève Theorem:

$$X(\vec{r}) = \sum_i^{\infty} \xi_i \sqrt{\lambda_i}\phi_i(\vec{r}) \tag{2.4}$$

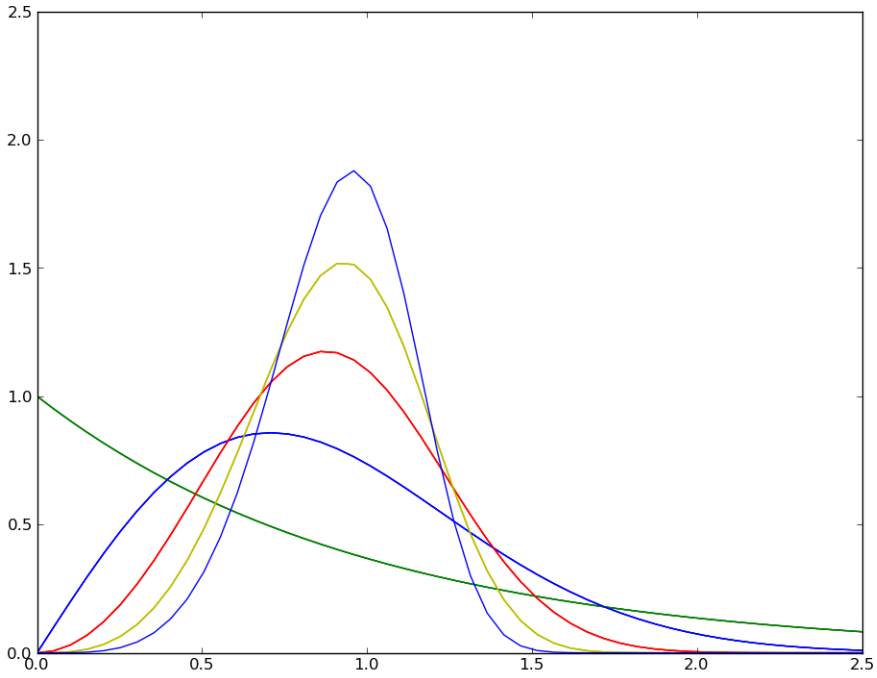


FIG. 2.1. *The Weibull distribution is commonly used in material science as a description for the distribution of strength in a set of material samples. The shape of the Weibull distribution depends strongly on the Weibull Modulus* m.

where the ξ_i are stochastically uncorrelated random variables.

The KL method in ALEGRA is based on this theorem. The user specifies the domain and covariance function. ALEGRA then solves the eigenproblem 2.3 to produce the desired number of eigenfunctions and eigenvalues. However, ALEGRA uses a slightly more general form for the series expansion:

$$X(\vec{r}) = \bar{X} + \sigma \sum_{i}^{N} \xi_i \sqrt{\lambda_i} \phi_i(\vec{r}). \tag{2.5}$$

$\bar{X}$ is the mean value and σ is provided to allow control over the scale of the spatial variation. Users should use ξ_i that are independently normally distributed with zero mean and unit variance. The user also chooses to limit the sum to a finite number of terms N, which we refer to as the KL dimension. An illustration of a random field's KL expansion is provided in Figure 2.3.

3. Summer 2013 Code Enhancements. The spatial variation cababilities in ALEGRA were improved over the course of the summer. These aspects included the organization of the spatial variation code, the logic of the spatial variation code, and the input syntax available to users.

At the beginning of the summer, the code in ALEGRA pertaining to spatial variation was scattered across several unrelated classes. Several of these classes had nearly identical code within them. At the

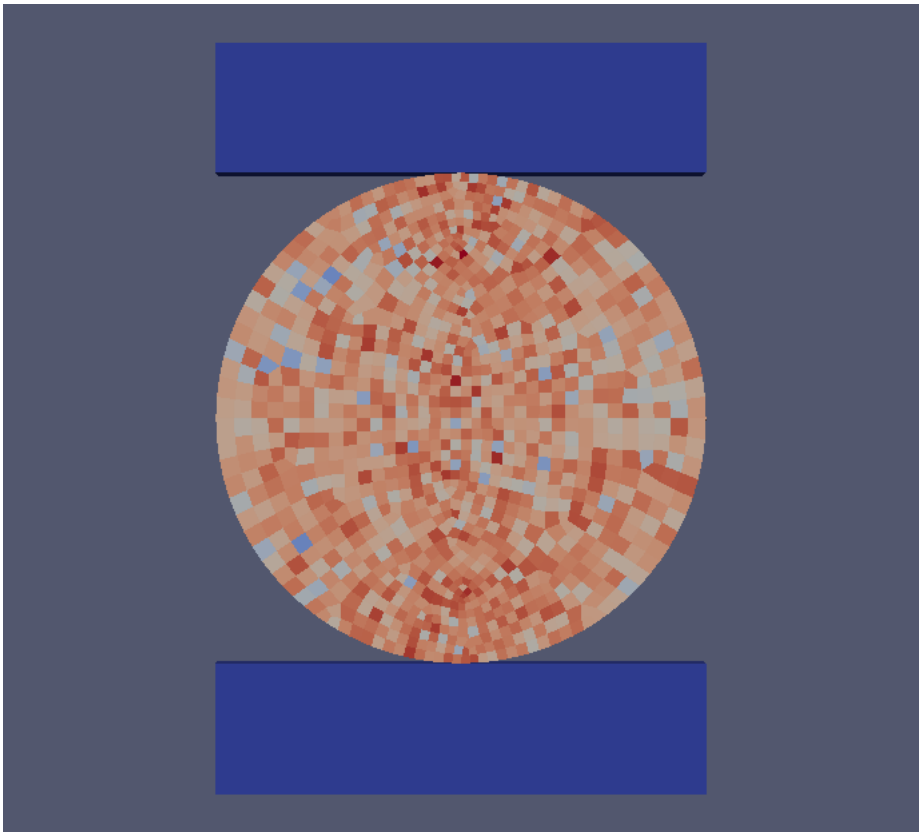


FIG. 2.2. *The SFC spatial variation method has been applied to the yield stress of this cylinder, with perturbations of the Weibull variety.*

end of the summer, the code has been unified and focused on one class whose only purpose is to support spatial variation. The code redundancies have been eliminated as well. Now spatial variability algorithms and modeling can be applied much more easily and generally. The algorithms can be applied to any material or any blocks (subdomains), or the intersection of the two. It is hoped that this more generalized and unified user interface will encourage users to begin thinking about how they might productively consider the effects of material variability in their modeling efforts. The end result of the work is a unified, sensible framework that can be easily built upon as new insights are obtained or additional approaches are needed relative to modeling spatial variation.

Although a basic code base for KL based random field descriptions was in place at the beginning of the summer of 2013 due to prior efforts in this direction, the implementation did not carefully match up with the mathematical discription discussed in section 2. The ξ_i in equation 2.5 were uniformly distributed and generated internally by ALEGRA . Furthermore, the algorithm written to determine a unique sign for the eigenfunctions was flawed, and would frequently cause failures.

These issues were corrected and the KL random field expansion code has been made consistent with its mathematical basis. The ξ_i are now provided externally to the model. These random variables are thus now easily available via the embedded DAKOTA-ALEGRA interface and the user can easily control and provide

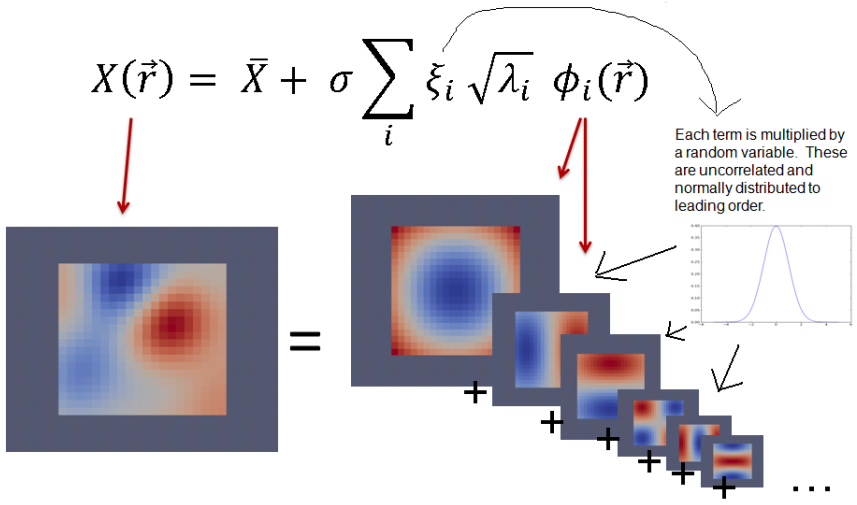


FIG. 2.3. *An illustration of the KL expansion of a random field.*

probabilistic assessment of quantities of interest associated with spatial variability via the algorithms found in DAKOTA. The unique sign algorithm was also made robust. The unique sign algorithm is important because one wishes to have a processor independent solution to the eigenvalue problem. The only time this algorithm is known to fail to give the same result independent of number of processors used is if the problem has a geometric symmetry which causes a repeated eigenvalue. This leads to uniqueness for the associated eigenvectors only up to a basis which spans the associated invariant subspace. In this case, one can either break the symmetry by slight modifications to the domain or restrict the stochastic analysis to a constant number of processors and a single machine. Results from different machines or processors numbers will vary in particulars. However, we expect the results to be stochastically equivalent.

At the beginning of the summer, the spatial variation input syntax available to users was in a somewhat incoherent state. Spatial variation in material parameters and spatial variation in initial state had very different input syntax. The fundamental differences between the SFC and KL methods were not enforced by the input syntax since a user could input keywords used the SFC method in the same line as keywords meant for a KL method. At the end of the summer, the spatial variation in material parameters and initial state both have identical input syntax and the fundamental differences between the SFC and KL methods are now respected by the input syntax, which enforces a much more intuitive and self-consistent user experience.

4. Prospects for Future Improvements. The work completed this summer provides a foundation for future progress. The clean, unified spatial variation framework currently used by ALEGRA now provides fertile ground for enhancements. Some enhancements to ALEGRA 's spatial variation capabilities which could be implemented in the future include empirically determined covariance functions and a robust methodolgy for verification of random field capability.

The KL expansion of a random field is critically dependent on the random field's covariance function. A correct covariance function is essential to producing an accurate representation of spatial variation. However, ALEGRA users only have two covariance functions to choose from at the present moment. They have the

following form in 3D

$$K_1(\vec{x},\vec{y}) = \exp\left(-(|x_1-y_1|+|x_2-y_2|+|x_3-y_3|)/L_c\right) \tag{4.1}$$

$$K_2(\vec{x},\vec{y}) = \exp\left(-((x_1-y_1)^2+(x_2-y_2)^2+(x_3-y_3)^2)/L_c^2\right) \tag{4.2}$$

and a similar form in 2D where L_c is user specified scalar correlation length. The covariance function is then used to construct an $n \times n$ covariance matrix, where n is the number of elements in the domain which approximates Equation 2.3. The KL expansion code then performs an eigensolve on this covariance matrix to produce the λ_i and $\phi_i(\vec{x})$. The covariance functions K_1 and K_2 capture the behavior we expect; points of the field close to each other are highly correlated, while points of the field far from each other have little correlation. However, they will certainly fail to capture the statistics of the true spatial variability for real materials in many instances. Thus, an option that could benefit ALEGRA users would be to employ empirical data to construct a customized covariance matrix. A user could create a set of sample data fields from empirical measurements and then feed the matrix into ALEGRA , which would compute a sample covariance matrix from the data. The KL expansion code would then perform an eigensolve on this sample covariance matrix. The result would be a KL expansion which is highly customized to the particular material or initial condition being simulated, and hence a much more accurate representation of the spatial variation. Details of how this might work are given in [4].

With the capability to create spatial variation, there is a need to verify that capability. It turns out that much is known relative to the structure of Gaussian random fields. For example, formulas for the expected value of the Euler characteristic of the excursion sets of a Gaussian random field are available in the literature [2,3]. This analysis could be used to verify whether random fields are being produced correctly. It is thus important to provide robust support for verification capabilities in the ALEGRA development and test environment. Currently, verification of ALEGRA solutions is performed by means of comparisons with single forward executions or spatial convergence studies. However, problems involving spatial variation must be verified through a statistical analysis of an ensemble of executions. The tools and/or examples necessary to show how to do this effectively must still be developed in a form that is readily usable. This is especially important to demonstrate in the case of KL expansions with high geometric symmetry as discussed above. In this case distributional metrics should be able to be verifiable as long as each set of ensemble runs fixes the machine and number of processors and the stochastic dimensions are properly converged.

5. Conclusions. In the summer of 2013, significant strides were made in terms of increasing the usability and coherence of ALEGRA 's spatial variation methodology. The groundwork laid provides a foundation for future innovation and capability development. Many challenges and opportunities are in sight for the development of spatial variation in ALEGRA and for providing more meaningful simulation analysis of real systems.

Acknowledgement. The authors acknowledge the prior work of Dale Koenig and Brian Carnes in building the initial code base for the KL expansion capability in ALEGRA and Emily Fischer for additional contribution to the random field coding. John Niederhaus provided a helpful review.

REFERENCES

[1] 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.

[2] R. J. ADLER, *Some new random field tools for spatial analysis*, Stochastic Environmental Research and Risk Assessment, 22 (2008), pp. 809–822.

[3] R. J. ADLER AND J. E. TAYLOR, *Random Fields and Geometry*, Springer, 2007.
[4] R. GHANEM AND A. DOOSTAN, *On the construction and analysis of stochastic models: Characterization and propagation of the errors associated with limited data*, Journal of Computational Physics, 217 (2006), pp. 63–81.
[5] E. J. GUMBEL, *Statistics of Extremes*, Columbia University Press, 1958.
[6] R. B. LEAVY, R. M. BRANNON, AND O. E. STRACK, *The use of sphere indentation experiments to characterize ceramic damage models*, International Journal of Applied Ceramic Technology, 7 (2010), pp. 606–615.
[7] E. VANMARCKE, *Random Fields*, World Scientific, 2010.

A PARTICLE-IN-CELL (PIC) METHOD FOR THE SIMULATION OF PLASMAS

ERIC M. WOLF* AND MATTHEW M. BETTENCOURT[1]

Abstract. We propose a new particle-in-cell (PIC) method for the simulation of plasmas based on a recently developed, unconditionally stable solver for the wave equation. We describe the 1D implementation, and present the results of several standard test problems, showing good agreement with linear theory.

1. Introduction. Collisionless plasmas - systems of charged particles interacting through electromagnetic fields - are modelled by the Vlasov-Maxwell system of partial differential equations (PDEs), which couple Maxwell's equations, describing the evolution of the electric and magnetic fields **E** and **B**, to Vlasov equations, a type of hyperbolic PDE describing the evolution of the phase-space probability density functions (PDFs) f_s of the various species s of charged particles:

$$\frac{\partial f_s}{\partial t} + \mathbf{v}\cdot\nabla_{\mathbf{x}} f_s + \frac{q_s}{m_s}(\mathbf{E}+\mathbf{v}\times\mathbf{B})\cdot\nabla_{\mathbf{v}} f_s = 0 \tag{1.1}$$

$$\frac{1}{c^2}\frac{\partial \mathbf{E}}{\partial t} = \nabla\times\mathbf{B} - \mu_0\mathbf{J} \qquad \frac{\partial \mathbf{B}}{\partial t} = -\nabla\times\mathbf{E} \tag{1.2}$$

$$\nabla\cdot\mathbf{E} = \rho/\varepsilon_0 \qquad \nabla\cdot\mathbf{B} = 0 \tag{1.3}$$

$$\rho(\mathbf{x},t) = \sum_s q_s \int_{\mathbf{v}} f_s(\mathbf{x},\mathbf{v},t)d\mathbf{v}, \quad \mathbf{J} = \sum_s q_s \int_{\mathbf{v}} \mathbf{v} f_s(\mathbf{x},\mathbf{v},t)d\mathbf{v} \tag{1.4}$$

In these equations, **x** and **v** represent position and velocity, **J** is the current density, ρ is the charge density, q_s and m_s are particle charge and mass of species s, ε_0 and μ_0 are the electric permittivity and magnetic permeability of the vacuum, and c is the speed of light in vacuum.

Particle-in-cell (PIC) methods [1], in development and use since the 1960s, combine an Eulerian description of the fields with a Lagrangian description of the PDFs; that is, fields are evolved on a mesh, while PDFs are represented by moving particles whose trajectories are characteristics in phase-space of the corresponding Vlasov equation. Thus, PIC methods require a method to compute the fields on a mesh and a method to compute particle trajectories, as well as interpolation tools to provide for their coupling. This work focuses on a new method for the computation of the fields.

Under the Lorenz gauge condition, $\nabla\cdot\mathbf{A} + \frac{1}{c}\phi_t = 0$ (where the subscript t stands for partial differentiation), Maxwell's equations can be reformulated in terms of wave equations for the scalar potential ϕ and the vector potential **A**:

$$\phi_{tt} - \frac{1}{c^2}\nabla^2\phi = \rho/\varepsilon_0 \tag{1.5}$$

$$\mathbf{A}_{tt} - \frac{1}{c^2}\nabla^2\mathbf{A} = \mu_0\mathbf{J} \tag{1.6}$$

The electric and magnetic fields, **E** and **B**, are then given by

$$\mathbf{E} = -\nabla\phi - \mathbf{A}_t \tag{1.7}$$

$$\mathbf{B} = \nabla\times\mathbf{A} \tag{1.8}$$

*Michigan State University, wolferi1@msu.edu
[1]Sandia National Laboratories, mbetten@sandia.gov

Recently, a novel method for the solution of the wave equation [2,3], based on the Method of Lines Transpose (MOLT) and a fast boundary integral solver, has been proposed that is unconditionally stable - that is, the Courant-Friedrichs-Lewy (CFL) condition limiting the ratio of the time step size to the spatial step size, typical of explicit methods, need not be obeyed to maintain numerical stability. Our goal is to apply this wave solver to (1.5) and (1.6), in conjunction with an appropriate description of particles, to develop a PIC method.

In the non-relativistic, zero-magnetic field limit, it is typical to make the electrostatic approximation, $\mathbf{E} = -\nabla\phi$, $-\nabla^2\phi = \rho/\varepsilon_0$, simplifying the Vlasov-Maxwell system to the Vlasov-Poisson system. Correspondingly, we consider the same non-relavistic, zero-magnetic field limit, and drop $\mathbf{A}$ and the wave equation (1.6) from our model. In the limit of particle velocities small compared to the speed of light, this will agree approximately with the electrostatic case.

2. Description of Method.

2.1. Wave Solver. We summarize the basics of a recently proposed wave equation solver that is unconditionally stable, but refer the reader to the recent work [3] for the details. The solver is based on the Method of Lines Transpose (MOLT), in which a semidiscrete equation is obtained by discretizing the time derivative. While many time discretizations are possible, each leading to a different variant of the solver with different properties, we presently use a backward difference formula (BDF):

$$\phi_{tt}^{n+1} = \frac{2\phi^{n+1} - 5\phi^n + 4\phi^{n-1} - \phi^{n-2}}{\Delta t^2} - \frac{11\Delta t^2}{12}\phi_{tttt}(\eta). \tag{2.1}$$

Substituting into the wave equation (1.5) and rearranging gives the elliptic equation:

$$\left(\nabla^2 - \frac{2}{(c\Delta t)^2}\right)\phi^{n+1} = -\frac{5\phi^n - 4\phi^{n-1} + \phi^{n-2}}{(c\Delta t)^2} - \rho(x, t_{n+1})/\varepsilon_0 \tag{2.2}$$

In one dimension, this elliptic equation may be solved by a boundary integral method. A naive implementation of this solution would cost $O(N^2)$ operations for N grid points; however, a fast convolution algorithm was developed in [2, 3] that costs only $O(N)$ operations, giving a method that is competitive with standard finite difference methods for the wave equation (and, by the Lorenz gauge formulation, for Maxwell's equations). In higher dimensions, alternating dimension implicit (ADI) dimensional splitting combined with the fast 1D solver provides for a fast solution (with, however, an associated splitting error). We consider only the 1D case at present.

The Green's function of the one dimensional elliptic operator $\left(\frac{1}{\alpha^2}\frac{\partial^2}{\partial x^2} - 1\right)$ is $G(x,y) = \frac{\alpha}{2}e^{-\alpha|x-y|}$, so that the solution to (2.2) in 1D is given by [2]

$$\begin{aligned}\phi^{n+1}(x) = \alpha \int_a^b e^{-\alpha|x-y|}\left\{\frac{5\phi^n - 4\phi^{n-1} + \phi^{n-2}}{4} + \frac{\rho(y, t^{n+1})}{2\alpha^2\varepsilon_0}\right\}dy \\ + Ae^{-\alpha(x-a)} + Be^{-\alpha(b-x)}\end{aligned} \tag{2.3}$$

where $\alpha = \sqrt{2}/(c\Delta t)$, and where we assume that the support of $\rho(\cdot, t^{n+1})$ is contained in the interval $[a,b]$, and the constants A and B are chosen to impose boundary conditions at $x = a$ and $x = b$. A number of boundary conditions have been implemented, including Dirichlet, Neumann, periodic and outflow boundary conditions. A domain decomposition method has also been developed and tested. The exponential decay of the Green's function allows for a fast recursive numerical computation of (2.3) on a mesh of points in the interval $[a,b]$, while providing for unconditional stability.

2.2. Exact Integration of Particles. In PIC methods, particles are typically represented by shape functions, so that $\rho(x,t^n) = \sum_i q_i S(x-x_i^n)$, where the sum is over particles and $S(x)$ is the shape function. For typical, piecewise polynomial shape functions, the contribution of $\alpha \int_a^b e^{-\alpha|x-y|}\rho(y,t^n)dy = \sum_i q_i \alpha \int_a^b e^{-\alpha|x-y|}S(y-x_i^n)dy$ to (2.3) may be computed exactly. This step replaces the charge-accumulation step of a typical electrostatic PIC algorithm.

In the special case of $S(x) = \delta(x)$ (point particles), we have

$$\alpha \int_a^b e^{-\alpha|x-y|}S(y)dy = \alpha e^{-\alpha|x|} \tag{2.4}$$

More typical, however, are piecewise linear particles (sometimes called cloud-in-cell (CIC) interpolation). In this case,

$$S(x) = \begin{cases} 1-|x| & |x|<1 \\ 0 & |x|\geq 1 \end{cases} \tag{2.5}$$

where we have normalized the particle width to 2 for simplicity, giving

$$\alpha \int_a^b e^{-\alpha|x-y|}S(y)dy = \begin{cases} \frac{2e^{-\alpha|x|}}{\alpha}(\cosh\alpha - 1) & |x|\geq 1 \\ 2\left[1-|x|-\frac{e^{-\alpha|x|}}{\alpha}+\frac{e^{-\alpha}}{\alpha}\cosh(\alpha x)\right] & |x|<1 \end{cases} \tag{2.6}$$

We can again use fast exponential recursion to evaluate these integrals; if there are P particles and N grid points, we can evalute the integrals in $O(P+N)$ operations (naive direct evaluation would take $O(PN)$ operations, an unacceptable cost). For any integrable function $u(x)$, we have

$$I[u](x) = \alpha \int_a^b e^{-\alpha|x-y|}u(y)dy = I^L[u](x) + I^R[u](x) \tag{2.7}$$

where

$$I^L[u](x) = \alpha \int_a^x e^{-\alpha(x-y)}u(y)dy, \quad I^R[u](x) = \alpha \int_x^b e^{-\alpha(y-x)}u(y)dy. \tag{2.8}$$

These satisfy

$$I^L[u](x+\Delta x) = e^{-\alpha\Delta x}I^L[u](x) + \alpha \int_x^{x+\Delta x} e^{-\alpha(x+\Delta x-y)}u(y)dy \tag{2.9}$$

$$I^R[u](x) = e^{-\alpha\Delta x}I^R[u](x+\Delta x) + \alpha \int_x^{x+\Delta x} e^{-\alpha(y-x)}u(y)dy \tag{2.10}$$

These relations provide the basis for the fast evaluation of the particle integrals. Consider a partition of the interval $x_1, x_2, ..., x_N$ of the interval $[a,b]$. For simplicity, assume the partition is uniform (though this is not necessary), so that $x_{j+1} = x_j + \Delta x$ for $j = 1, ...N-1$. We will initially compute

$$I_j^L[u] = \alpha \int_{x_{j-1}}^{x_j} e^{-\alpha(x_j-y)}u(y)dy, \quad j = 2, ..., N$$

$$I_j^R[u] = \alpha \int_{x_j}^{x_{j+1}} e^{-\alpha(y-x_j)}u(y)dy, \quad j = 1, ..., N-1$$

and define $I_1^L[u] = 0$ and $I_N^R[u] = 0$. Then we perform a recursive update:

for $j = 1$ to $N-1$ **do**
$I^L_{j+1}[u] = e^{-\alpha\Delta x} I^L_j[u] + I^L_{j+1}[u]$
$I^R_{N-j}[u] = e^{-\alpha\Delta x} I^R_{N-j+1}[u] + I^R_{N-j}[u]$
end for

Finally, we get $I[u] = I^L[u] + I^R[u]$.

Specializing to the particle integrals, we initialize I^L and I^R to zero and accumulate the contributions of each particle:

for each particle i **do**
for each cell $[x_j, x_{j+1}]$ overlapping with particle shape **do**

$$I^L_{j+1} = I^L_{j+1} + \alpha \int_{x_j}^{x_{j+1}} e^{-\alpha(x_{j+1}-y)} q_i S_i(y) dy$$
$$I^R_j = I^R_j + \alpha \int_{x_j}^{x_{j+1}} e^{-\alpha(y-x_j)} q_i S_i(y) dy$$

end for
end for

where $S_i(x)$ is the shape function of particle i. The integrals are evaluated analytically. On a uniform mesh with a linear particle shape of width $2\Delta x$, the particle shape overlaps with at most three cells, and one evaluation of an exponential function is required per particle. Some care must be taken for particles located in the first or last cell of the domain; in a periodic domain, the shape function should be periodically extended.

Then, we perform the recursive update to I^L and I^R, and finally obtain $I = I^L + I^R$.

Further, we note that the contribution to (2.3) of a uniform background charge distribution may also be computed exactly by evaluating

$$\alpha \int_a^b e^{-\alpha|x-y|} dy = 2 - e^{-\alpha(x-a)} - e^{-\alpha(b-x)} \tag{2.11}$$

on the mesh and multiplying by the uniform background charge density ρ_{back}.

2.3. Particle Mover and Computational Cycle. The motion of charged particles is governed by the Newton-Lorentz force law:

$$\frac{d\mathbf{x}_i}{dt} = \mathbf{v}_i \tag{2.12}$$
$$\frac{d\mathbf{v}_i}{dt} = \frac{q_i}{m_i}(\mathbf{E} + \mathbf{v}_i \times \mathbf{B}) \tag{2.13}$$

where i is the index of the particle, $\mathbf{x}_i$ is the position, $\mathbf{v_i}$ is the velocity, q_i is the charge and m_i is the mass of particle i. Various time stepping schemes have been developed for (2.12) and (2.13). In PIC methods, fields are interpolated to particle locations to provide the accelerations of the particles, and particle locations and motions are interpolated to the mesh to provide the charge density ρ and current $\mathbf{J}$ for the field equations.

There is a great deal of freedom in the choice of numerical scheme used to integrate (2.12) and (2.13) (the particle mover), as it is largely independent of the field solver. Presently we consider the explicit leapfrog scheme, as it is one of the simplest and most widely used choices; other particle movers can be

easily incorporated. The explicit leapfrog scheme in 1D is given by

$$v_i^{n+1/2} = v_i^{n-1/2} + \Delta t a_i^n \tag{2.14}$$

$$x_i^{n+1} = x_i^n + \Delta t v_i^{n+1/2} \tag{2.15}$$

The acceleration in our case is given by

$$a_i^n = \frac{q_i}{m_i} E(x_i^n) = \frac{q_i}{m_i} \sum_{j=1}^{N} E_j^n S(x_j - x_i^n) \tag{2.16}$$

where the sum in j is over grid points x_j, and the electric field is given by the centered difference approximation:

$$E_j = \frac{\phi_{j+1} - \phi_{j-1}}{2\Delta x} \tag{2.17}$$

Thus, one time step of the overall algorithm can be summarized as follows and in Figure 2.3. Assume we have the particle positions x_i^n and velocities $v_i^{n-1/2}$, the previous time steps for the potential $\phi^{n-2}, \phi^{n-1}, \phi^n$, and the electric field at the particle positions $E^n(x_i^n)$.

1. Update the particle velocities: $v_i^{n+1/2} = v_i^{n-1/2} + \Delta t E^n(x_i^n)$
2. Update the particle positions: $x_i^{n+1} = x_i^n + \Delta t v_i^{n+1/2}$
3. Evaluate exact particle integrals from particle positions x_i^{n+1}, perform numerical quadrature and BC updates to compute ϕ^{n+1}.
4. Compute E^{n+1} on the grid from finite differences applied to ϕ^{n+1}, interpolate to particle positions to obtain $E^{n+1}(x_i^{n+1})$.

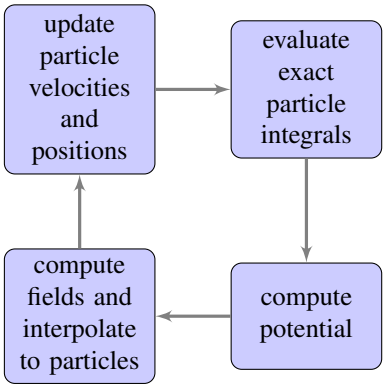


FIG. 2.1. *Cycle of one time step*

2.4. Start Up. It is necessary to provide the wave solver with three initial time steps as an initial condition. In the numerical examples below, this is done by computing the electrostatic potential from the Poisson equation $-\nabla^2\phi = \rho/\varepsilon_0$ through a standard finite difference method.

Alternatively, we could begin the time stepping with zero initial potential everywhere, and step the wave solver to steady state (keeping the particles stationary), then use this steady state as the initial condition. This

is possible because the BDF time discretization introduces numerical dissipation. While many time steps may be necessary to reach steady state, it will not be overly expensive, as the expense of the particle mover and exact particle integration is avoided. This approach avoids the construction and inversion of the Poisson matrix, which is particularly helpful in higher dimensions with complex geometry.

3. Numerical Examples. We apply the proposed method to a number of test problems. First, we show that, while the self-force of a single particle is not identically zero, it is not so large to render the method useless. Second, we show that the method can capture the behavior of a two-body oscillatory system. Finally, we apply the method to a number of standard electrostatic test problems, and show that the results agree with linear theory.

3.1. Self-Force Test. Standard PIC algorithms on uniform meshes have the property of zero-self force - a single particle placed in, say, a periodic domain with a uniform neutralizing background charge, will not experience any acceleration, up to round off error. The key to this property is the use of the same interpolation scheme in interpolating charge to the mesh points, and fields to the particle positions. Due to the nature of the boundary integral solver, our proposed method does not have the same zero-force property, as the Green's function modifies the effective weighting of charge to the grid. In this section, we see that, while a self-force error is present, it does not result in a catastrophic acceleration of the particle.

In this example, we consider a single particle in a periodic domain with uniform neutralizing background. We take the length of the domain to be 1, the mass of the particle to be 1, the charge of the particle to be -1, and also set $\varepsilon_0 = 1$. We vary the parameters Δx, Δt, and the speed of light c and investigate the resulting self-force error, considering $c\Delta t/\Delta x = 100$ or 1000. We plot the momentum of the particle, which is the time integral of the self-force error, over $10,000$ time steps in Figure 3.1.

3.2. Two Particle Oscillation. In this section, we apply the proposed method to a two-body problem. A light, negatively-charged particle (electron) is displaced by 5 cells from a heavy, positively-charged particle (ion), and the system oscillates. Because the electron is much lighter than the ion, the motion of the electron is much larger; however, both particles are advanced according to (2.14) and (2.15). In this example, the length of the domain is $L_x = 1$, the mesh size is $\Delta x = 0.01$, the speed of light is $c = 1$, and the time step size is $\Delta t = 10$; the electron charge to mass ratio is -1, the electron charge is -4×10^{-8}, the ion charge to mass ratio is $1/2000$, and the ion charge is 4×10^{-8}. The oscillation of the electron over many periods is displayed in Figure 3.2; the oscillation is well-resolved, with approximately 100 time steps per period of oscillation, and does not degrade over many time steps and periods.

3.3. Cold Plasma Langmuir Wave. In the following three test problems, electrons are loaded from an initial distribution of the form

$$f_e(x,v,t=0) = f_e(v)\left(1 + x_p \sin\left(\frac{2\pi x}{L_x}\right)\right) \tag{3.1}$$

where L_x is the length of the domain, x_p is the amplitude of perturbation, and $f_e(v)$ is the initial velocity distribution. We take charge to mass ratio for electrons to be 1, and normalize time quantities to the plasma frequency ω_p and spatial quantities to the Debye length λ_D. We will consider a periodic domain with a uniform neutralizing background charge, and further we set the speed of light $c = 1000$ and $\varepsilon_0 = 1$.

We consider a cold plasma Langmuir wave [6] with $f_e(v) = \delta(v)$. We use $N_p = 10000$ particles, and take $x_p = 0.01$, $\Delta t = 0.1$, and $L_x = 1$, with $N_x = 500$ cells in the domain. The potential energy is plotted in Figure 3.3; we see that the plasma frequency is accurately reproduced.

3.4. Two Stream Instability. We consider the two stream instability with $f_e(v) = \delta(v - v_{\text{beam}}) + \delta(v + v_{\text{beam}})$. According to the dispersion relation for the two instability from linear theory, we have

$$\omega^4 - 2\omega^2(\omega_p^2 + k^2 v_{\text{beam}}^2) + k^2 v_{\text{beam}}^2(k^2 v_{\text{beam}}^2 - 2\omega_p^2) = 0 \tag{3.2}$$

which gives the greatest growth rate, $\gamma \approx 0.3535$, for $k \approx 3.06$. We therefore scale the domain to this value of k, and take $L_x = 2\pi/3.06$, $\Delta t = 0.1$, $N_x = 100$, $v_{\text{beam}} = 0.2$, $N_p = 30000$, and $x_p = 0.001$. The growth of the $k = 3.06$ mode of the electric field is shown in Figure 3.4, and agrees with the rate from linear theory.

3.5. Landau Damping. Finally, we consider Landau damping of Langmuir waves in a warm plasma, with $f_e(v)$ taken to be Maxwellian. The dispersion relation from linear theory in this case gives a decay rate of $\gamma \approx 0.154$ for the $k = 0.5$ mode. We take $L_x = 4\pi$, $\Delta t = 0.1$, $N_x = 100$, $v_{\text{therm}} = 1$, $N_p = 300000$, and $x_p = 0.1$. The decay of the $k = 0.5$ mode of the electric field is shown in Figure 3.5, and agrees with the rate from linear theory.

4. Conclusions. We have proposed a PIC method based on a novel wave equation solver applied to the scalar and vector potentials in the Lorenz gauge. The wave solver is unconditionally stable, permitting the use of time steps much larger than allowed by the usual CFL condition. Using explicit leapfrog as the particle mover, we have applied this method to several standard 1D electrostatic test problems and found results agreeing with linear theory. The presence of non-physical self-force is a drawback to the proposed method, but one that does not seem to be fatal to the usefulness of the method. Future work will include the incorporation of more advanced particle movers, with the ultimate goal of generating an implicit PIC method that can take time steps much larger than the plasma period [4, 5], the extension of the method to two and three dimensions, and the extension to the fully electromagnetic case.

REFERENCES

[1] C. BIRDSALL AND L. A.B., *Plasma physics via computer simulation*, Taylor and Francis, New York, 2005.
[2] M. CAUSLEY, A. CHRISTLIEB, B. ONG, AND L. VAN GRONINGEN, *Method of Lines Transpose: An Implicit Solution to the Wave Equation*, Mathematics of Computation, to appear (2013).
[3] M. CAUSLEY, Y. GÜÇLÜ, E. WOLF, AND A. CHRISTLIEB, *A Fast, Unconditionally stable solver for the wave equation based on the Method of Lines Transpose*, Mathematics of Computation, to appear (2013).
[4] G. CHEN, L. CHACÓN, AND D. BARNES, *An energy- and charge-conserving, implicit, electrostatic particle-in-cell algorithm*, J. Comput. Phys., 230 (18) (2011).
[5] A. LANGDON, *Analysis of the time integration in plasma simulations*, J. Comput. Phys., 30 (1979), pp. 202–221.
[6] T. STIX, *Waves in Plasmas*, Amer Inst of Physics, New York, 1992.

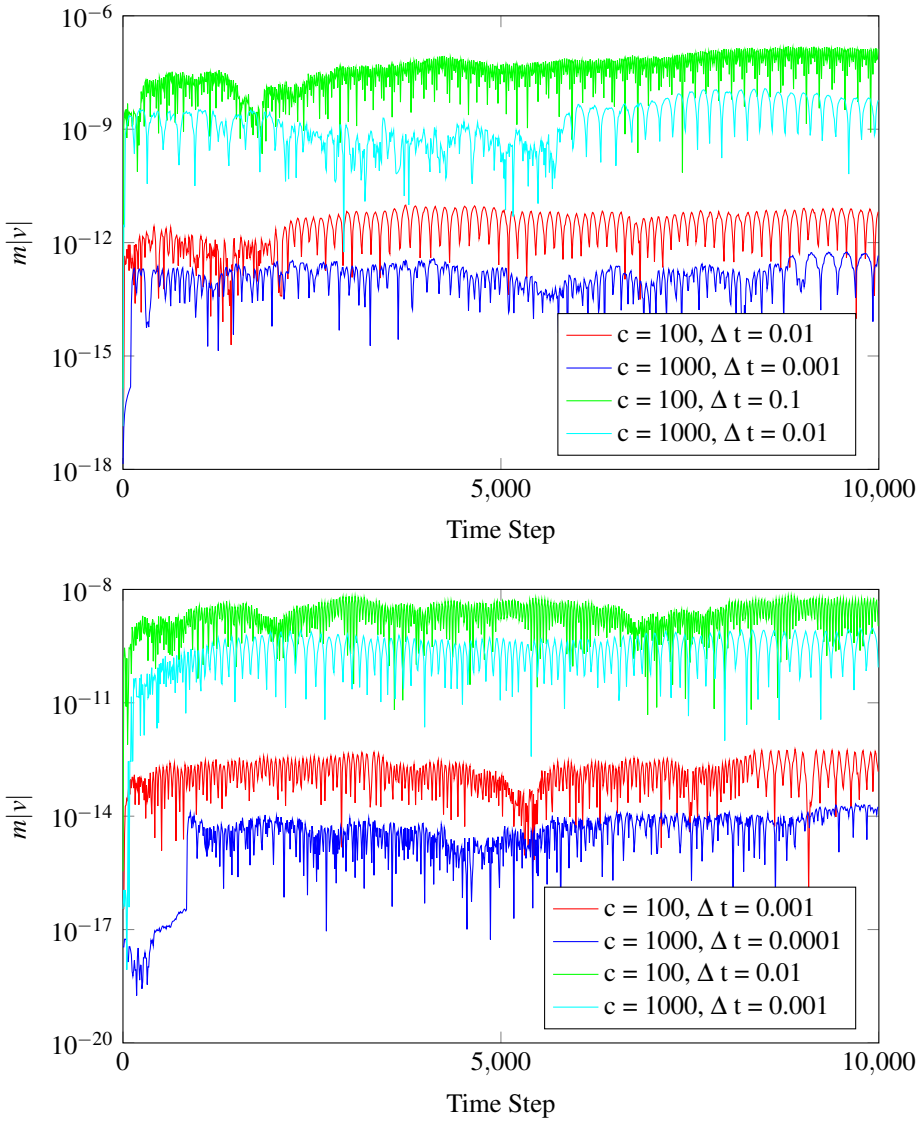


FIG. 3.1. *Momentum due to self-force error with* $\Delta x = 0.01$ *(top) and* $\Delta x = 0.001$ *(bottom).*

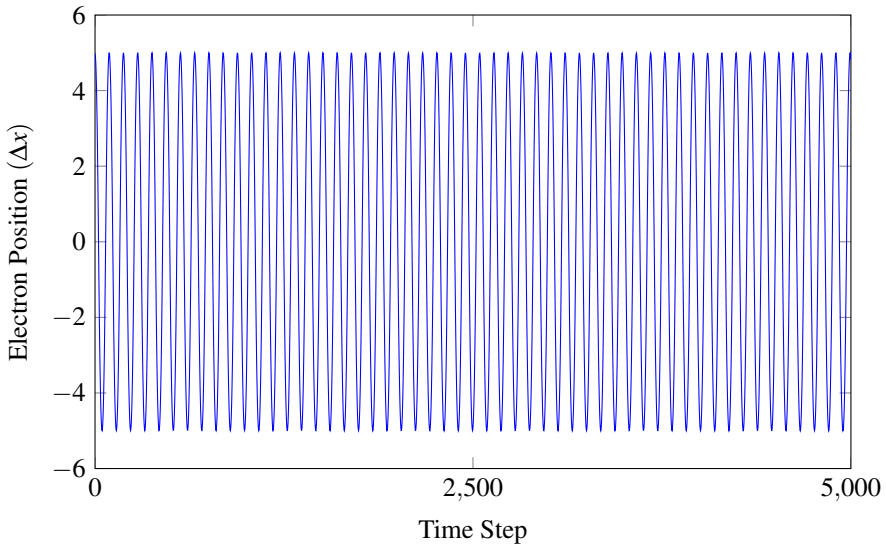


FIG. 3.2. *Position of electron oscillating about an ion. The well-resolved oscillation (approx. 100 time steps per period) does not degrade over many time steps and periods of oscillations.*

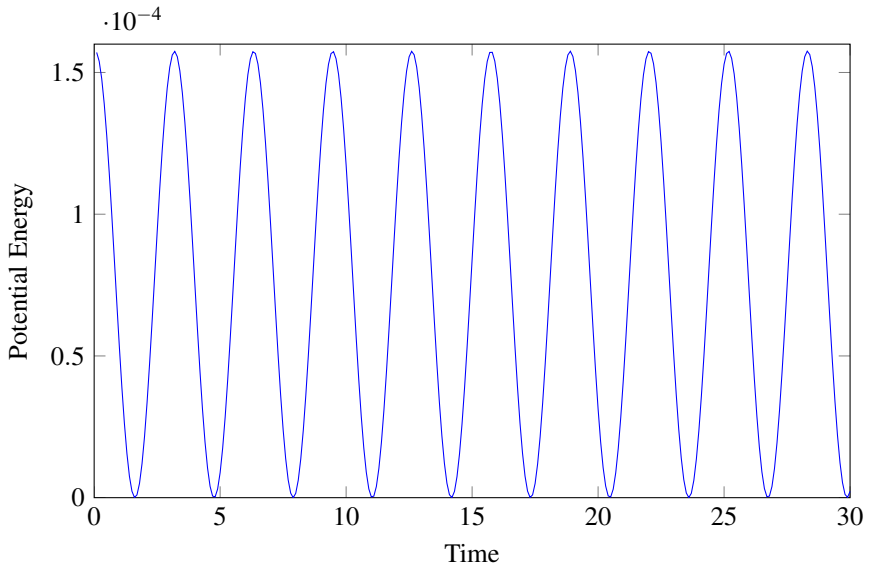


FIG. 3.3. *Cold plasma Langmuir wave. The plasma frequency* $\omega_p = 1$ *is accurately reproduced.*

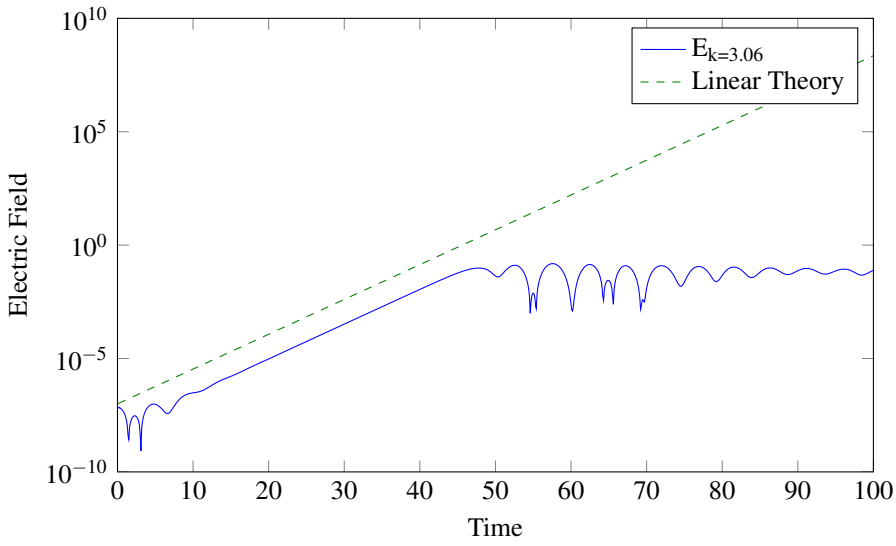


FIG. 3.4. *Two stream instability. The rate of growth in the* $k = 3.06$ *mode of the electric field is accurately reproduced.*

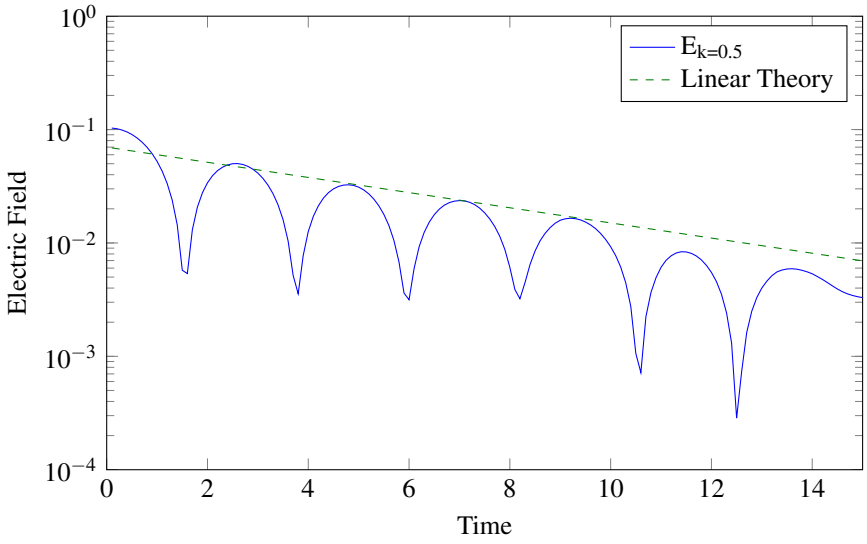


FIG. 3.5. *Landau damping. The rate of decay of the* $k = 0.5$ *mode of the electric field is accurately reproduced.*