CSRI SUMMER PROCEEDINGS 2008

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

Editors:

Denis Ridzal and S. Scott Collis
Sandia National Laboratories

December 11, 2008

SAND2008-8257P

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed-Martin Company, for the United States Department of Energy
under Contract DE-AC04-94AL85000.

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

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

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

Available to DOE and DOE contractors from

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

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

Available to the public from

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

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

Preface

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

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

Starting in 2006, CSRI has encouraged all summer participants and their mentors to contribute a technical article to the CSRI Summer Proceedings, of which this document is the third 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 2008 CSRI Proceedings, research articles have been organized into the following broad technical focus areas — *computational mathematics and algorithms, discrete mathematics and informatics, architectures and systems software, and 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 2008 as documented by the high quality articles in this proceedings. The success of CSRI hinged on the hard work of 20 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 Deanna Ceballos, Bernadette Watts, Mel Loran, Dee Cadena, and Vonda Coleman. The success of CSRI is, in large part, due to their dedication and care, which are much appreciated. We would also like to thank those who reviewed articles for this proceedings — their feedback is an important part of the research training process and has significantly improved the quality of the papers herein. We would like to thank David Womble for his advice, guidance and overall CSRI management. Finally, we want to acknowledge the ASC program for their continued support of the CSRI and its activities which have benefited both Sandia and the greater research community.

Denis Ridzal
S. Scott Collis

December 11, 2008

Table of Contents

Computational Mathematics and Algorithms

Articles in this section focus on fundamental numerical algorithms ranging from mesh adaptation, optimal quadrature, and model reduction to numerical linear algebra and multigrid algorithms that each have broad potential for application in a variety of computational disciplines.

Wilson and Carnes study the selection of optimal coarsening parameters for the mesh–adaptive solution of time–dependent PDEs. Their results indicate that the optimal amounts of pre and post coarsening depend heavily on the nature of the problem at hand. *Nong and Thornquist* present a survey of model order reduction techniques used for the solution of linear time–invariant (LTI) systems in descriptor form. The presentation and analysis of the techniques, which are applied to LTI systems arising in the simulation of integrated electrical circuits, are followed by useful recommendations based on the Hankel singular values. *Keegan et al.* investigate the use of sparse grids, typically employed in stochastic sampling, for the numerical integration in "physical space", with applications to the finite element solution of PDEs. They devise an adaptive algorithm for the construction of numerical integration rules that are well suited for exact integration of complete polynomial spaces of a given degree. *Karlin and Hu* describe a new matrix class in the linear algebra package Epetra that gives Trilinos applications access to the Optimized Sparse Kernel Interface (OSKI) package. They observe that OSKI has the potential to produce large speedups in sparse matrix operations and thus directly benefit a number of important Sandia applications. *Chen and Tuminaro* propose a multigrid algorithm for the solution of linear systems associated with saddle–point problems in which the constraint equations are of the "Dirichlet" or "Neumann" type. Such systems arise in the simulation of power grid networks. The proposed algorithm effectively converts the saddle–point system to a symmetric positive definite system to which standard multigrid techniques can be applied. *Berry et al.* study the application of classical multigrid ideas to the solution of linear systems associated with stochastic PDEs. They investigate conditions for positive definiteness of such linear systems and, based on their generic structure, develop an effective multigrid approach.

D. Ridzal
S.S. Collis

December 11, 2008

OPTIMAL COARSENING METHODS FOR ADAPTIVITY IN TRANSIENT PROBLEMS

BRIAN J. WILSON * AND BRIAN R. CARNES †

Abstract. In transient problems, the cost of adaptivity can be much more significant than in stationary problems. It is therefore desirable to carefully manage adaptivity cost within the overall simulation cost budget. An overall goal is to design an adaptivity system for transient problems that is both accurate and efficient, with a minimum of problem-specific tuning needed. In particular, coarsening parameters become much more important. By tuning coarsening parameters and eliminating adaptivity when the error falls within acceptable limits, computational cost can be significantly reduced. We present some optimal values for coarsening parameters. Furthermore, we illustrate adaptive timestepping with a simple one dimensional example and then consider its effect on coarsening parameters.

1. Introduction. In general, one can encounter two potential problems when solving a partial differential equation numerically. The first is that while a finite element solver runs quickly on a sufficiently coarse grid, the potential error in the computed solution may lie outside acceptable boundaries. On the other hand, if the error is kept within acceptable tolerances by using a sufficiently fine grid, the solver may run too slow to be of any practical use. An ideal mesh which strikes a balance between the two extremes should be coarse in regions with low error and fine in regions with high error. This is the essential idea behind adaptivity. We start by solving the PDE on a coarse mesh, which can be done cheaply. Using this crude solution, we obtain an estimate of the error on any given element using an a posteriori error estimator and refine elements in regions where the most error occurs. We repeat this process, adapting the mesh until the estimated error falls below a specified tolerance. Furthermore, it is often beneficial to coarsen previously refined elements on which the solution has low error. This allows more refinement of elements on which the solution has high error.

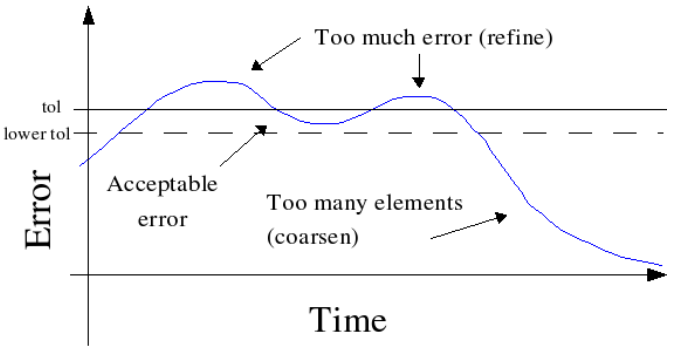
We can also apply adaptivity to transient problems. This consists of a two step process. First, the solution is advanced from one timestep to the next using a numerical integrator, such as the Implicit (Backward) or Semi-Implicit Euler method. Second, the spatial error at that timestep is reduced to an acceptable tolerance using the adaptive refinement method described above. We also consider the case when the error is already acceptable after advancing the solution to a new timestep, presumably because of refinement at previous timesteps. In this case we may proceed directly to the next timestep without performing any adaptive refinement. However, if the error is far below the tolerance, we may wish to coarsen some elements in regions of low error until the total error is somewhat larger, but still below the acceptable tolerance. This can reduce computational cost for the remaining timesteps. Figure 1.1 illustrates how adaptivity should behave with respect to the error as time progresses.

Finally, one might also wish to implement adaptivity in time. The basic idea is to adjust the timestep size based on the local truncation error (LTE), which measures the local error introduced by discretization in time. Since the exact error is not known, an estimate of the LTE is obtained as the difference of approximate solutions computed using two different numerical integration schemes. The timestep is then adjusted according to the proportion of the desired error tolerance to the estimated LTE.

In each of the situations above (spatial adaptivity in stationary problems, spatial adaptivity in transient problems, and spatial and temporal adaptivity in transient problems) several unresolved issues remain. What method should be used to determine which elements are refined or coarsened? To what degree does this method depend on problem-specific characteristics? In this paper, we use the percent of total error method to mark elements for

*Colorado State University, wilson@math.colostate.edu
†Sandia National Laboratories, bcarnes@sandia.gov

FIG. 1.1. *General Scheme for Spatial Adaptivity in Transient Problems*



refinement and coarsening. The percent of total error marker ranks the elements according to their percentwise contributions to the total error. It then marks the highest error elements for refinement (resp. lowest error elements for coarsening) up to specified percentages of the total error. Based on previous studies, we mark elements corresponding to 70% of the total error for refinement. We limit our studies to optimal percentages of error to coarsen.

We study three (transient) solutions of a typical parabolic problem (corresponding to different source terms), which exhibit different types of behavior: advection of a singularity out of a domain, circulation of a smooth function inside the domain, and growth and decay of a sine wave. Since the source terms are contrived so that we know the exact solutions, we can use the exact error in the percent of total error method to drive adaptivity. The percent of total error method is implemented in the SIERRA [2] codes ENCORE [1], which computes the interpolant of a known solution, and ARIA [4], which computes the finite element approximation to the solution of a partial differential equation. Due to space constraints, we present detailed results only for the advecting singularity, both with and without temporal adaptivity.

The paper is organized as follows. Section 2 examines coarsening parameters for spatial adaptivity applied to the transient problems without adaptive timestepping. In section 3, we present a one dimensional example illustrating the method of adaptive timestepping. Section 4 examines the effect of temporal adaptivity on optimal coarsening parameters for the advecting singularity solution. Finally, we present our conclusions in section 5.

2. Spatial Coarsening in Transient Problems. Coarsening is much more useful in transient problems than in stationary problems. This is because the singularities and other regions of the solution with high error can shift as time passes, leaving behind regions of the mesh where high levels of refinement are no longer required. If such regions are coarsened as local error decreases, significant time savings may result.

First we will study the H^1 error in the interpolant of explicitly defined functions using ENCORE, and afterwords we will examine the error in the solution of corresponding differential equations using ARIA. As mentioned above, we will use the percent of total error marker so we can precisely control the percentage of the total error we wish to reduce through refinement. Based on the results of previous studies for stationary problems, we refine elements with 70% of the total error, and limit our study to the optimal percentage of error to coarsen.

One problem which can occur in a transient problem is that the error not only meets the tolerance but then decreases too much as time progresses. This is a problem because if we can meet the error tolerance with a much coarser mesh, we can significantly reduce computational time. Hence if the error is below a certain percentage of the tolerance (85% worked well

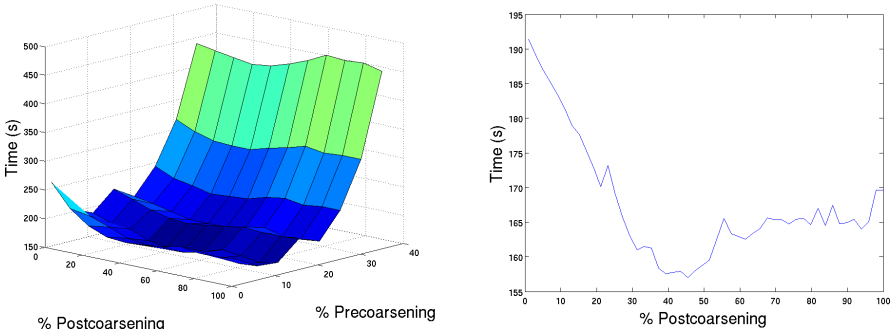
for our purposes) we perform one step of coarsening without doing any refinement. (If the error remains between the tolerance and 85% of the tolerance when the timestep is advanced, we proceed immediately to the next timestep without adapting the mesh.) We refer to this new type of coarsening (which decreases the number of elements and increases the error) as *postcoarsening* in order to distinguish it from the standard method of coarsening (which occurs simultaneously with refinement in order to decrease the error). We also refer to the standard method as *precoarsening*. This terminology is appropriate since in general the mesh is first refined down to the error tolerance and coarsened later, when the error remains below the tolerance after a timestep is completed.

We wish to find optimal precoarsening and postcoarsening parameters for each function. The general approach will be to first perform a 2D coarse parameter study, estimate either the minimum precoarsening or postcoarsening parameter, and then perform a more detailed 1D parameter study on the second parameter with the first parameter set at a minimum. This approach is feasible because the 2D parameter study results are fairly monotone. Then, we estimate the optimal parameters and plot the error and number of elements at each timestep using these parameters. Furthermore, we compare these results with corresponding plots for some non-optimal parameters to determine what makes the optimal parameters optimal. Finally, we include some sample meshes generated by the given function with the optimal parameters.

We examine the function

$$u(x,t) = \left(\sqrt{(x_1 - t)^2 + x_2^2}\right)^{.8}, \quad x = (x_1, x_2) \in [-1,1]^2, \quad t \in [0,2]$$

which has a singularity at $x = (t, 0)$. In fact this is simply the function $|x|^{.8}$ shifting to the right. Eventually, the singularity moves out of the domain. The two dimensional parameter study, using ENCORE to compute the interpolant, plots computational time as a function of pre and post coarsening percentages. It is shown in Figure 2.1(a). It is clear from this graph that 10% precoarsening is optimal no matter what the value of postcoarsening. Hence we fix precoarsening at 10% and perform a more detailed study of the optimal postcoarsening parameter, shown in Figure 2.1(b).

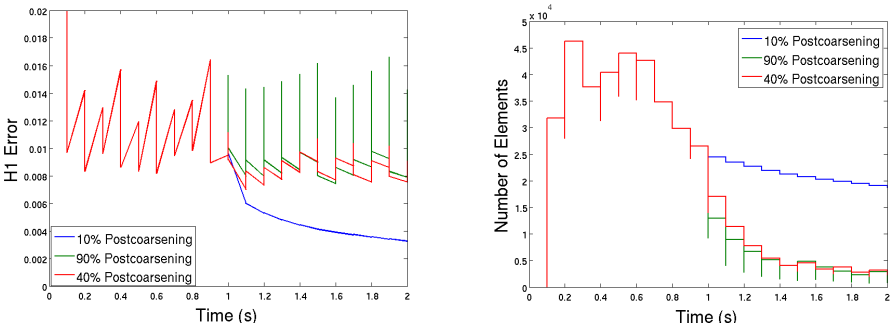


(a) Pre/Post Coarsening (2D)

(b) Postcoarsening (10% Precoarsening)

FIG. 2.1. *ENCORE Coarsening Parameter Studies*

It appears that the optimal percentage for postcoarsening is somewhere between 40% and 50%. To see why this is so, we compare number of elements and error at each timestep for three different postcoarsening percentages, 10%, 40%, and 90% in Figure 2.2 below.



(a) H1 Error (b) Number of Elements

FIG. 2.2. *Error and Number of Elements for Various Postcoarsening Percentages (10% Precoarsening)*

We can see that with 10% postcoarsening, the error falls well below the tolerance $\epsilon = .01$, but the number of elements is not significantly reduced, resulting in wasted time spent computing solutions on meshes which are too large. On the other hand, using 90% postcoarsening causes so much coarsening that the error grows too high, and more time is wasted re-refining the mesh. At about 40% postcoarsening, however, an optimal number of elements are refined so that the error stays just slightly beneath the tolerance, allowing an effective reduction in the number of elements in the mesh yielding substantive time savings. A few sample meshes for the solution with 10% precoarsening and 40% postcoarsening are displayed in Figure 2.3.

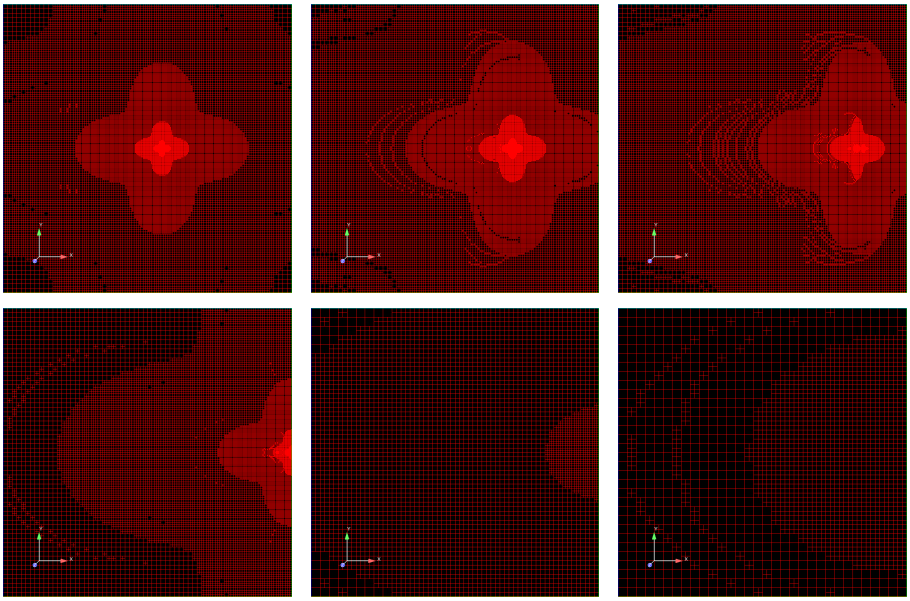
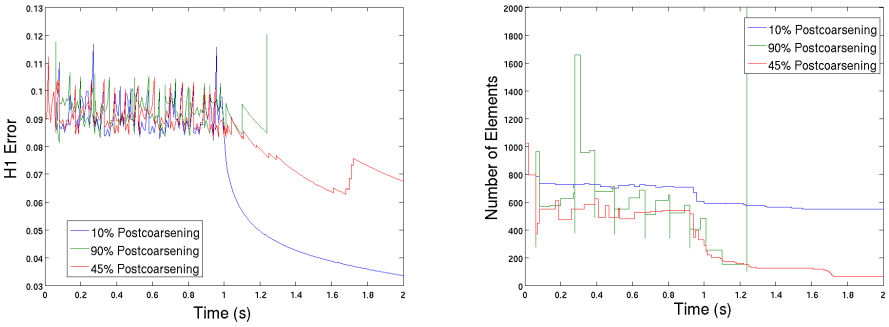


FIG. 2.3. *Sample Meshes (10% Precoarsening, 40% Postcoarsening)*

Since the interpolant is a fairly good approximation to the solution in the space of piecewise linear functions, we would expect these coarsening parameters to be fairly reliable estimates of the optimal parameters for computing the finite element solution in ARIA as well. However, there are a few differences. A full parameter study in ARIA is impractical because

it fails to converge for many parameter values, for example if precoarsening is too high. In this case the precoarsening cancels out the refinement to the point where the adaptivity does not reduce the error to an acceptable tolerance. We can, however, examine the finite element approximation in ARIA using the optimal parameters suggested by computing the interpolant in ENCORE. As before we plot error and number of elements with respect to time for 10%, 40%, and 90% precoarsening with no postcoarsening in Figure 2.4.



(a) H1 Error (b) Number of Elements

FIG. 2.4. *Error and Number of Elements for Various Postcoarsening Percentages (10% Precoarsening)*

Similar analyses were performed for the solutions

$$v(x,t) = \exp(-(x_1 - .5\cos(\pi t))^2 - (x_2 - .5\sin(\pi t))^2)$$

and

$$w(x,t) = 1 + \sin(4\pi x_1)\sin(5\pi x_2)\cdot t \cdot e^{-.41\pi^2 t}$$

for $x = (x_1, x_2) \in [-1,1]^2$ and $t \in [0,2]$, with source terms f computed appropriately. The first solution $v(x,t)$ is a bump function circulating around the domain. A 2D parameter study indicates that the fastest convergence times are achieved when the postcoarsening is set to 0%. Since the bump function stays inside the domain, the error never decreases to a point where postcoarsening is beneficial. The 1D parameter study for 0% postcoarsening indicates that about 20% precoarsening is optimal. This coarsens previously refined elements in regions that the bump has already passed by to allow new refinement in the areas currently occupied by the bump. The second solution $w(x,t)$ is essentially a decaying sine wave. In order to simplify the problem so that the initial condition is constant, the sine wave grows linearly from a constant before decaying exponentially, peaking at about $t = .2$. The 2D parameter study for this solution indicates just the opposite than for $v(x,t)$; the best convergence times are obtained with 100% postcoarsening! Since the exponential decay of the sine wave causes a rapid decrease in error, full postcoarsening quickly reduces the number of elements in the mesh. The 1D parameter study shows that $0 - 17\%$ precoarsening is optimal. Since $w(x,t)$ involves no advection, the regions of high error in the solution remain stationary. Precoarsening may have some small benefit in shrinking and expanding the refined regions as the sine wave grows and decays, respectively. Clearly, precoarsening beyond 17% merely works against refinement, increasing computational time.

3. Adaptive Timestepping Example. In transient problems, the difference between interpolation error and finite element approximation error becomes more significant. This is

because spatial interpolation at a given time does not incur any time error. In contrast, a fully discrete finite element scheme requires both spatial and temporal approximation. For example, using linear finite elements and Backward Euler time integration, the $L^\infty - H^s$ a priori error estimate for the finite element error takes the form

$$\max_{0\le n\le N} \|u(t_n) - U^n\|_{H^s(\Omega)} \le C_1\, k + C_2\, h^{2-s},$$

for $s = 0, 1$, where k is the timestep and h is the spatial mesh size. The difference between this and spatial interpolation error is the presence of the timestep size k, resulting from the numerical time integration. A proof is given in the book by Thomee [5].

Algorithms for timestep adaptivity also become important, since choosing a variable timestep size can be both more accurate and efficient than using a fixed timestep. In Aria, the adaptive timestep scheme is based on work by P.M. Gresho [3]. The time integration schemes in Aria are called θ-schemes, since they are parameterized by θ, which ranges from zero to one. Here we set $\theta = 1$, which corresponds to the Backward (Implicit) Euler method.

As an example to illustrate adaptive timestepping, we consider the solution of the simple ordinary differential equation

$$y' = \alpha y$$
$$y(0) = y_0$$

Of course this has the exact solution

$$y(t) = y_0 e^{\alpha t}$$

We partition the interval of time $[0, T]$ as $0 = t_0 < t_1 < ... < t_N = T$, where N is the number of discrete timesteps. We follow the convention that $y(t)$ represents the exact solution and Y the discrete approximation, with Y^n being the value of Y at time t_n. The approximation Y^n is easily computed using the Backward Euler method. This consists of replacing y' with a difference quotient and evaluating the right hand side of the equation at at time t_n to obtain

$$Y^n = (1 - \alpha k_n)^{-1} Y^{n-1}$$

where $k_n = t_n - t_{n-1}$ is the timestep. If the timestep is constant, i.e. $k = k_n$, we can further solve for Y^n in terms of the initial condition $Y^0 = y_0$ to obtain

$$Y^n = (1 - \alpha k)^{-n} y_0 = \left(\frac{1}{1 - \alpha k}\right)^{\frac{t_n}{k}} y_0 = y_0 e^{\left[\frac{1}{k} \ln\left(\frac{1}{1-\alpha k}\right)\right] t_n}$$

In the limit as $k \to 0$, the approximation converges to the exact solution since $\lim_{k\to 0} \frac{1}{k} \ln\left(\frac{1}{1-\alpha k}\right) = \alpha$.

In order to estimate the error in the computed approximation (also known as the corrector), we use a second approximation computed using a different method of numerical integration. The predictor Y_p^n is computed using the Forward Euler method, in which the right hand side of the equation is evaluated at time t_{n-1} instead of t_n. It is given by

$$Y_p^n = (1 + \alpha k_n) Y^{n-1},$$

and again in the case of a constant timestep $k = k_n$, we substitute for Y^{n-1} to obtain the solution

$$Y_p^n = (1 + \alpha k)(1 - \alpha k)^{1-n} y_0 = \frac{1 - \alpha^2 k^2}{(1 - \alpha k)^n} y_0$$

We can now compute the difference between the predictor and the corrector as

$$|Y^n - Y_p^n| = \left[(1-\alpha k_n)^{-1} - (1+\alpha k_n)\right] Y^{n-1} = \frac{\alpha^2 k_n^2}{1-\alpha k_n} Y^{n-1} \approx \mathcal{O}(k^2)$$

In the computation of the predictor Y_p, the right hand side of the equation is evaluated at time t_{n-1}, underestimating the difference quotient on the left hand side, which causes Y_p to undershoot the exact solution y. Similarly, Y will overshoot the exact solution y, so that $|Y - Y_p|$ is very roughly twice the actual error. Hence it is reasonable to use $\frac{1}{2}|Y - Y_p|$ as an estimate for local truncation error (LTE). In general, $\frac{1}{2}\|Y - Y_p\|$ is a good estimate of the LTE for problems in any number of dimensions assuming the timestep size is sufficiently small. We note that this estimate is second order in time, which is true in general for the Backward Euler method.

The adaptive timestepping strategy is based on this *a priori* knowledge of the LTE. Let the current timestep by given by k_n. To achieve a target error of ϵ, we use the ratio of the errors and the order of accuracy to compute a candidate timestep size k_n^c:

$$\frac{\epsilon}{1/2\,\|Y^n - Y_p^n\|} \approx \left(\frac{k_n^c}{k_n}\right)^2$$

Solving this equation for k_n^c yields the candidate timestep formula

$$k_n^c \equiv k_n \left(\frac{2\,\epsilon}{\|Y^n - Y_p^n\|}\right)^{1/2}$$

To verify the accuracy of this timestep, one could re-solve for Y^n and recompute the error estimator. However, in codes like Aria the candidate timestep is used to compute the next timestep, so that $k_{n+1} \equiv k_n^c$. In our case we can calculate the exact LTE since we know the analytic solution. Using the Backward Euler method applied to the exact solution $y(t_{n-1})$ we obtain a modified corrector given by

$$\tilde{Y}^n = (1-\alpha k_n)^{-1} y(t_{n-1}) = \frac{y_0 e^{\alpha t_{n-1}}}{1-\alpha k_n}$$

which yields the exact LTE:

$$|\tilde{Y}^n - y(t_n)| = \left|\frac{y_0 e^{\alpha t_{n-1}}}{1-\alpha k_n} - y_0 e^{\alpha t_n}\right| = |y_0| e^{\alpha t_n} \left|\frac{e^{\alpha k_n}}{1-\alpha k_n} - 1\right|$$

where we have used the fact that $t_n = t_{n-1} + k_n$.

We plot the LTE estimate $\frac{1}{2}|Y - Y_p|$ and actual LTE $|\tilde{Y}^n - y(t_n)|$ with constant timestep $k = .01$, $\alpha = 1$, and inital condition $y_0 = 1$ in Figure 3.1(a). The estimate is remarkably accurate. It should be noted that these quantities measure the *local* error since Y_p^n and Y^n are both computed from Y^{n-1}, the value of the corrector at the previous timestep, and the modified corrector $\tilde{Y}^n$ is computed from the solution itself at the previous timestep. If we wish to obtain global error estimates instead, we can compute a modified predictor $\hat{Y}_p^n$ from the previous (also modified) *predictor* value $\hat{Y}_p^{n-1}$:

$$\hat{Y}_p^n = (1+\alpha k_n)\hat{Y}_p^{n-1}$$

and compute Y^n from Y^{n-1} as before. With constant timestep $k = k_n$, we solve to obtain

$$\hat{Y}_p^n = (1+\alpha k_n)^n y_0$$

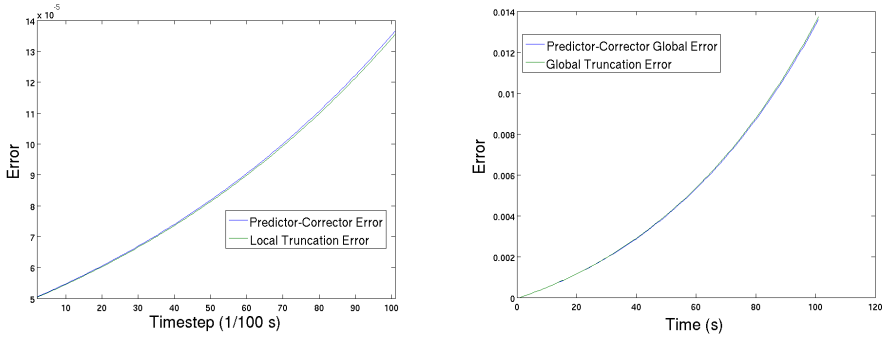
Since $\hat{Y}_p^n$ and Y^n are not computed from the same quantity at the previous timestep as before, these calculations include the effect of the accumulation of error over all previous timesteps. We plot the global error estimate $\frac{1}{2}|Y - \hat{Y}_p|$ and exact global error $|Y - y|$ in Figure 3.1(b).

It should be noted that these quantities measure the *local* error since Y_p^n and Y^n are both computed from Y^{n-1}, the value of the corrector at the previous timestep, and the modified corrector $\tilde{Y}^n$ is computed from the solution itself at the previous timestep. If we wish to obtain global error estimates instead, we can compute a modified predictor $\hat{Y}_p^n$ from the previous (also modified) *predictor* value $\hat{Y}_p^{n-1}$:

$$\hat{Y}_p^n = (1 + \alpha k_n)\hat{Y}_p^{n-1}$$

and compute Y^n from Y^{n-1} as before. With constant timestep $k = k_n$, we solve to obtain

$$\hat{Y}_p^n = (1 + \alpha k_n)^n y_0$$



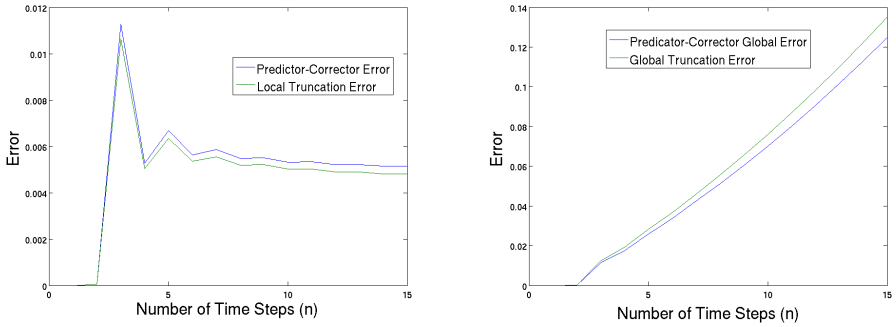
(a) Local Error (b) Global Error

FIG. 3.1. *Comparison of Local and Global Truncation Errors vs. Predictor-Corrector Estimates*

We can now implement adaptive timestepping by using the LTE $\frac{1}{2}|Y^n - Y_p^n|$ to compute the the next timestep k_{n+1} as outlined above. The global error estimate cannot be used because of the Forward Euler timestepping used to run the problem forward in time; after error has accumulated over multiple timesteps there is no cost effective way to go back in time to reduce it. If we were to try using the global error to compute the next timestep, the timestep would be adapted infinitely smaller, preventing the solution from ever reaching the final time. Thus the best one can do is to control the LTE at any given timestep, in which case the global error is bounded by the LTE tolerance multiplied by the number of timesteps. For example, the adaptive timestep for our example problem is

$$k_{n+1} = \frac{1}{\alpha}\sqrt{\frac{2\epsilon}{y_0}}(1 - \alpha k_n)^n$$

The local and global error estimates are presented in Figure 3.2; however in both cases the estimate for the LTE is used to adapt the timestep. Note that the adaptive algorithm was implemented with $\epsilon = .01$. Figure 3.2 clearly shows that the LTE is controlled so that it remains below .01. However, the adaptive timestepping uses only 15 timesteps as compared to the 100 timesteps used with fixed timestep $k = .01$. The global error should be bounded by $15\epsilon = .15$. Indeed, as we can see in Figure 3.2, the error at the final timestep is just below .15.



(a) Local Error (b) Global Error

FIG. 3.2. *Comparison of Local and Global Truncation Errors vs. Predictor-Corrector Estimates With Adaptive Timestepping*

4. Adaptivity in Space and Time. We now examine the effects of adaptivity in time on optimal coarsening parameters. As in the previous section, we compute the timestep according to the formula

$$k_{n+1} \equiv k_n \left(\frac{2\,\epsilon}{||Y^n - Y_p^n||} \right)^{1/2},$$

where ϵ is the desired error tolerance for the LTE at the current timestep, and Y_p^n and Y^n are the predictor and corrector computed using the Explicit and Implicit Euler methods, respectively. Thus at times when the solution varies slowly with respect to time, the LTE estimate will be smaller than the specified error tolerance, causing the timestep to increase. At times when the solution varies rapidly with respect to time, the predicted error will be larger than the specified tolerance, causing the timestep to decrease. Adapting the timestep in this way allows us to use the largest possible timesteps which limit the increase the error to the specified tolerance ϵ. As our results indicate, this can significantly reduce computational time. We apply adaptive timestepping to the solution of the parabolic problem

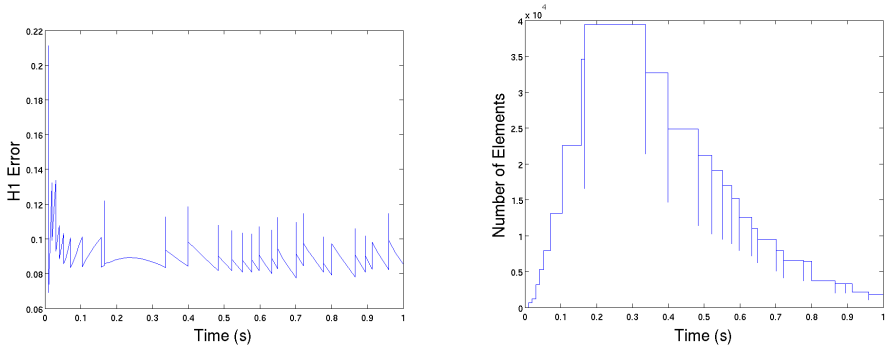
$$\epsilon u_t - \Delta u = f$$

where the source f is computed from the exact solution

$$u(x,t) = 1 + \sin(4\pi x_1)\sin(5\pi x_2) \cdot t \cdot e^{-.41\pi^2 t}$$

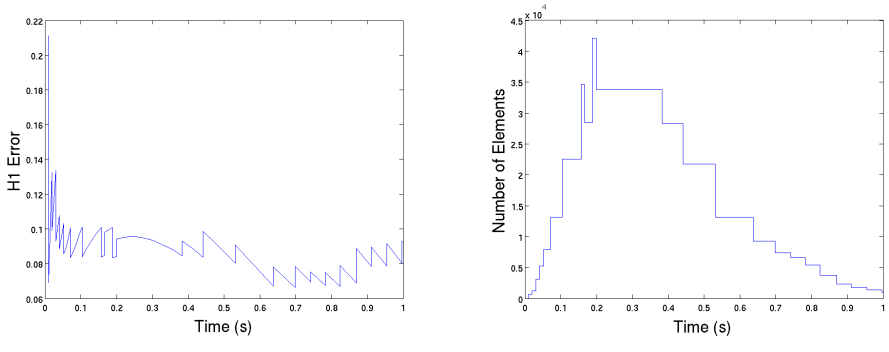
which is a sine wave undergoing linear growth and exponential decay. Computing the solution with ARIA, we plot the error and number of elements with respect to time using both 70% and 50% postcoarsening in Figures 4.1- 4.2. Refinement is set at 70% and precoarsening at 15%, which are appropriate parameters given the previous analysis.

Two observations should be noted from these results. First, we can see that with 70% postcoarsening the mesh is overcoarsened when the error gets small enough to trigger postcoarsening, and must be re-refined down to the error tolerance. If we reduce the postcoarsening to 50% this problem disappears. We hypothesize that adaptive timestepping somewhat reduces the need for postcoarsening since decay is handled not only by spatial coarsening but also by increasing the timestep size. Second, although it is not shown in Figures 4.1-4.2, adaptive timestepping somewhat reduces computational time, in the first case (70% postcoarsening) from 787 s to 579 s and in the second (50% postcoarsening) from 700 s to 440 s.



(a) H1 Error (b) Number of Elements

FIG. 4.1. *Error and Number of Elements with Adaptive Timestepping (70% Postcoarsening)*



(a) H1 Error (b) Number of Elements

FIG. 4.2. *Error and Number of Elements with Adaptive Timestepping (50% Postcoarsening)*

A comparison of the size of the timestep in the fixed vs. the adaptive cases, plotted in Figure 4.3, reveals the reason for these savings: the adaptive method uses less timesteps to reach the final time $t = 1$, resulting in less computation overall.

5. Conclusions. Our results indicate that the optimal amounts of pre and post coarsening depend on the behavior of the problem at hand. Although we considered three different problems exhibiting different types of behavior, due to space constraints we presented detailed results only for the first problem, which has a solution involving a singularity advecting out of a square domain. In this case, only about 10% precoarsening but approximately 40% postcoarsening is optimal. While much adaptivity is needed near the singularity, refined elements must be rapidly coarsened after the singularity leaves the domain in order to speed convergence by reducing the number of elements. The second example consisted of a bump function circulating around the domain. Since the bump function never leaves the domain, no postcoarsening can be performed without increasing the error above acceptable limits. About 20% precoarsening is needed to coarsen elements in areas where previous refinement has occurred due to the steep gradients of the bump function, but where refinement is no longer needed since the bump has moved out of the area. Finally, in the case of a sine wave growing linearly and then decaying exponentially, full postcoarsening is desirable due to the exponential reduction of error as the wave decays. Since this function involves no advection, little to no precoarsening is needed, although it is acceptable up to about 17% of the total error.

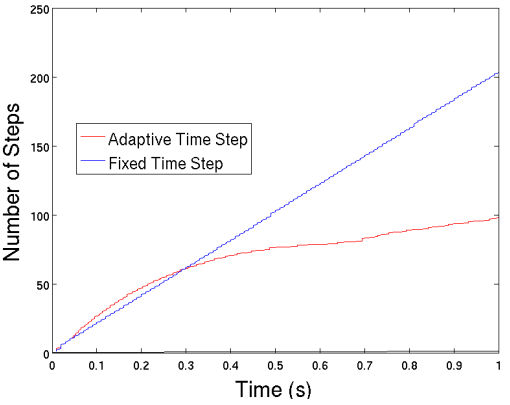


FIG. 4.3. *Time vs. Timestep*

Beyond that, precoarsening merely increases computational time.

Thus, we recommend that parameter values for coarsening be determined by the nature of the problem being studied. Solutions where the error tends to shift from place to place benefit most from precoarsening, which coarsens old areas of high error so that new areas may be refined. However, solutions which have diminishing error, for example due to parabolic smoothing or decay, benefit more from postcoarsening, which trades an acceptable increase in error for a mesh with fewer elements, reducing computational cost.

REFERENCES

[1] B. R. CARNES, K. D. COPPS, AND D. R. GASTON., *Encore user/theory manual*, tech. rep., Sandia National Laboratories, 2008. forthcoming.

[2] H. C. EDWARDS, *Sierra framework for massively parallel adaptive multiphysics applications.*, tech. rep., Sandia National Laboratories, 2004. SAND2004-6277C.

[3] P. M. GRESHO AND R. L. LEE, *On the time-dependent solution of the incompressible navier-stokes equations in two and three dimensions*, in Recent Advances in Numerical Methods in Fluids, C. Taylor and K. Morgan, eds., vol. 1, Pineridge Press Limited, 1980, ch. 2.

[4] P. NOTZ, S. R. SUBIA, M. M. HOPKINS, H. K. MOFFAT, AND D. R. NOBLE, *Aria 1.5 : User manual*, tech. rep., Sandia National Laboratories, 2007. SAND2007-2734.

[5] V. THOMEE, *Galerkin Finite Element Methods for Parabolic Problems*, Springer-Verlag, 1997.

A SURVEY OF MODEL ORDER REDUCTION METHODS FOR LTI SYSTEMS IN DESCRIPTOR FORM

RYAN NONG* AND HEIDI THORNQUIST†

Abstract. A survey of different model order reduction methods that are suitable for linear time invariant systems in descriptor form is presented. These techniques are applied to a number of typical interconnect circuits to construct corresponding reduced models. A comparison of the resulting models is documented. The reduced models are then re-integrated back into larger circuits, and a transient simulation is performed to compare the original and approximate time-domain responses and simulation time.

1. Introduction. While advances in manufacturing enable the fabrication of integrated circuits containing tens-to-hundreds of millions of devices, the time-sensitive modeling and simulation necessary to design these circuits pose a significant computational challenge. When the integrated circuit has millions of devices, performing a full system simulation can be infeasible. The principal reason for this is the time required for the nonlinear solver to compute the solutions of large linearized systems during the simulation of these circuits.

Model order reduction (MOR) techniques attempt to produce low-dimensional systems that capture the same response characteristics as the original systems while enabling substantial speedups in simulation time and resulting in much less storage requirements. While model-order reduction is an active area of research, the techniques see limited use in commercial Electrical Design Automation (EDA) tools.

In this paper, we present a survey of different model order reduction (MOR) methods applied to linear circuits. The methods we are studying are suitable for linear time invariant (LTI) systems in descriptor form, which are often the result of using the Modified Nodal Analysis (MNA) formulation. The algorithms for these methods are implemented and then applied to a number of typical interconnect circuits to construct corresponding reduced models. An assessment of how and in what situation each of the techniques is efficient in model order reduction is performed. In addition, we also re-integrate the reduced systems back into larger circuits and observe similar time-domain responses as the original circuit with significant speedups in simulation times. The paper is organized as follows: In Section 2, four different techniques (PRIMA [7], RKS [10], GSHSR [6] and IRKA [2]) from three different schemes (moment matching, balanced truncation and optimal $\mathcal{H}_2$) are briefly presented. Numerical analysis and results follow in Section 3. The paper concludes with a few final observations and future work. Additional details about the algorithms and results are presented in a number of appendices at the end of the paper.

In this paper, except when specified otherwise, upper case bold letters ($\mathbf{A}$, $\mathbf{B}$, etc.) denote matrices, lower case bold letters ($\mathbf{x}$, $\mathbf{y}$, etc.) vectors, and non-bold or Greek letters scalars. Script letters ($\mathcal{A}$, $\mathcal{E}$, etc.) denote special or structured matrices. Conjugate transpose is denoted by $\mathbf{A}^*$ and transpose by $\mathbf{A}^T$.

2. Methods. In this paper, we consider three different model order reduction schemes: Moment matching, balanced truncation and optimal $\mathcal{H}_2$ reductions. In all of these approaches, one essentially starts with the state space realization of the original LTI system $\mathbf{\Sigma} \equiv (\mathbf{C}, \mathbf{G}, \mathbf{B}, \mathbf{L})$ as in

$$\begin{aligned} \mathbf{C}\tfrac{d\mathbf{x}}{dt} &= -\mathbf{G}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \\ \mathbf{y}(t) &= \mathbf{L}^T\mathbf{x}(t), \end{aligned} \tag{2.1}$$

*CAAM Department, Rice University, ryannong@caam.rice.edu
†Electrical and Microsystems Modeling Dept., Sandia National Laboratories, hkthorn@sandia.gov

where $\mathbf{C} \in \mathbb{R}^{n\times n}$, $\mathbf{G} \in \mathbb{R}^{n\times n}$, $\mathbf{B} \in \mathbb{R}^{n\times p}$ and $\mathbf{L} \in \mathbb{R}^{n\times p}$ and $\mathbf{x}(t)$ is the state, $\mathbf{u}(t)$ the input and $\mathbf{y}(t)$ the output of the system. Also, n is the size of the original system and p the number of inputs (outputs). If $p = 1$, then (2.1) is referred to as a single-input-single-output (SISO) system and, if $p > 1$, it is a multiple-input-multiple-output (MIMO) system.

For model order reduction, one constructs two projection matrices $\mathbf{W} \in \mathbb{R}^{n\times k}$ and $\mathbf{V} \in \mathbb{R}^{n\times k}$ such that $\mathbf{W}^T\mathbf{V} = \mathbf{I}_k$, where k is the desired size of the reduced system ($k \ll n$). The reduced system is now $\hat{\mathbf{\Sigma}} \equiv (\hat{\mathbf{C}}, \hat{\mathbf{G}}, \hat{\mathbf{B}}, \hat{\mathbf{L}})$ governed by the following set of first-order LTI differential equations

$$\begin{aligned} \hat{\mathbf{C}}\tfrac{d\hat{\mathbf{x}}}{dt} &= -\hat{\mathbf{G}}\hat{\mathbf{x}}(t) + \hat{\mathbf{B}}\mathbf{u}(t) \\ \hat{\mathbf{y}}(t) &= \hat{\mathbf{L}}^T\hat{\mathbf{x}}(t), \end{aligned} \tag{2.2}$$

where $\hat{\mathbf{C}} = \mathbf{W}^T\mathbf{C}\mathbf{V}$, $\hat{\mathbf{G}} = \mathbf{W}^T\mathbf{G}\mathbf{V}$, $\hat{\mathbf{B}} = \mathbf{W}^T\mathbf{B}$, $\hat{\mathbf{L}}^T = \mathbf{L}^T\mathbf{V}$. For more details, we refer the readers to [1], [2] and the references therein.

The frequency input-output relationships of the original (2.1) and reduced (2.2) systems are determined by their corresponding transfer functions:

$$\begin{aligned} \mathbf{H}(s) &= \mathbf{L}^T(s\mathbf{C} + \mathbf{G})^{-1}\mathbf{B}, \\ \hat{\mathbf{H}}(s) &= \hat{\mathbf{L}}^T(s\hat{\mathbf{C}} + \hat{\mathbf{G}})^{-1}\hat{\mathbf{B}}. \end{aligned} \tag{2.3}$$

2.1. Integrating Reduced Models. Integrating the reduced model (2.2) into a larger circuit requires an admittance matrix to be generated from the original circuit (2.1). The admittance, or y-parameter, matrix provides the relationship between the input voltage and output current at any *port*, input or output node, of the original circuit. It is obtained by attaching a voltage source to every port of the original circuit so that every port becomes both an input and output node. The state space realization $(\mathbf{C}, \mathbf{G}, \mathbf{B}, \mathbf{L})$ of this modified circuit is then used, instead of the original, to generate the reduced model. As a result of this requirement for integration, even if the original circuit is SISO, the modified circuit used to generate the reduced model is always MIMO. This constraint will be taken into account during the presentation of the model order reduction schemes. For more details on reduced model integration, we refer the readers to [7] and the references therein.

2.2. Moment Matching Scheme. Consider the Laurent expansions of the transfer functions (2.3) of the original and reduced systems about a given point $s_0 \in \mathbb{C}$ as follows:

$$\begin{aligned} \mathbf{H}(s_0 + \sigma) &= \eta_0 + \eta_1\sigma + \eta_2\sigma + \ldots, \\ \hat{\mathbf{H}}(s_0 + \sigma) &= \hat{\eta}_0 + \hat{\eta}_1\sigma + \hat{\eta}_2\sigma + \ldots, \end{aligned}$$

where η_i and $\hat{\eta}_i$ are the moments of $\mathbf{\Sigma}$ and $\hat{\mathbf{\Sigma}}$ at s_0 respectively. The moment matching based methods aim to compute a reduced system $\hat{\mathbf{\Sigma}}$, with a certain number of moments matching those of the original system $\mathbf{\Sigma}$, i.e.,

$$\eta_i = \hat{\eta}_i, \quad i = 1, \ldots, l,$$

for some $l \ll n$. Note that l does not necessarily equal k/p. In this paper, we consider the following two moment matching techniques: PRIMA and RKS. Since these techniques can be implemented iteratively, they are quite numerically efficient. However, global error bounds are not available.

2.2.1. PRIMA. The Passive Reduced-Order Interconnect Macromodeling Algorithm (PRIMA) is proposed by Odabasioglu et al. [7] in 1998. The pseudocode for this algorithm can be found as Algorithm 1 in Appendix A. The algorithm utilizes the block Arnoldi

procedure. Note that for PRIMA, $\mathbf{W} = \mathbf{V}$. The resulting reduced system $\hat{\mathbf{\Sigma}}$ is proven to be passive and hence, stable. The number of matched moments is equal to the desired size of the reduced system $\hat{\mathbf{\Sigma}}$ divided by the number of inputs, i.e., $l = k/p$. While no global error bound is available, Heres [4] provides some heuristic considerations for error control of PRIMA in his Ph.D. thesis. A short summary of the stopping criteria can be found in Appendix B. However, based on our numerical experiments, the quality of the error estimation depends very highly on the interpolation points and hence, it is local.

2.2.2. RKS. The Rational Krylov Subspace (RKS) method for model-order reduction is proposed by Skoogh [10] and is based on the rational Krylov algorithm by Ruhe [9]. The pseudocode for this algorithm can be found as Algorithm 2 in Appendix A. The rational Krylov algorithm is a generalization of the standard Arnoldi and Lanczos methods. The advantage of the rational Krylov algorithm is that it provides the flexibility of choosing a set of m different interpolation points ($m \leq k/p$). The reduced system $\hat{\mathbf{\Sigma}}$ matches $l = k/p$ moments of the original system $\mathbf{\Sigma}$ at these interpolation points. Reduced models resulting from RKS are not guaranteed to be passive and stable, and also no global error bound is available.

2.3. Balanced Truncation Reduction. The balanced truncation reduction is classified as an SVD-based scheme. The scheme constructs the reduced model $\hat{\mathbf{\Sigma}}$ based on the Hankel singular values of the original system $\mathbf{\Sigma}$. For the LTI system $\mathbf{\Sigma}$ as in (2.1), the Hankel singular values can be computed by solving the following two generalized Lyapunov equations for the system Grammians $\mathcal{P}$ and Q:

$$\begin{aligned} \mathbf{G}\mathcal{P}\mathbf{C}^T + \mathbf{C}\mathcal{P}\mathbf{G}^T &= \mathbf{B}\mathbf{B}^T \\ \mathbf{G}^T Q\mathbf{C} + \mathbf{C}^T Q\mathbf{G} &= \mathbf{L}\mathbf{L}^T. \end{aligned} \tag{2.4}$$

Then the Hankel singular values of $\mathbf{\Sigma}$ are $\sigma_i(\mathbf{\Sigma}) = \sqrt{\lambda_i(\mathcal{P}Q)}, i = 1, \ldots, n$, the square roots of the eigenvalues of the product of the system Grammians. The projection matrices can be computed using the system Grammians and the reduced system $\hat{\mathbf{\Sigma}}$ results balanced. In addition, the reduced system $\hat{\mathbf{\Sigma}}$ has the following guaranteed properties: (a) stability is preserved, and (b) global error bounds exist in Hankel-norm approximation and they can be computed as follows:

$$\sigma_{k+1} \leq \|\mathbf{\Sigma} - \hat{\mathbf{\Sigma}}\|_\infty \leq 2(\sigma_{k+1} + \cdots + \sigma_n),$$

where k is the desired size of the reduced system $\hat{\mathbf{\Sigma}}$.

Despite the advantageous properties, MOR via balanced truncation is not very attractive due to its computational requirements. The reason is in directly solving the two Lyapunov equations (2.4); as n gets large, the complexity in computation and storage required is prohibitive. A number of efforts have been made to solve the Lyapunov equations iteratively, which we will not examine in this survey. For this scheme, we consider the following technique: GSHSR. This technique is a generalization of balanced truncation model-order reduction to descriptor systems.

2.3.1. GSHSR. The Generalized Schur-Hammarling Square Root (GSHSR) method is proposed by Mehrmann and Stykel [6]. A summary of the algorithm can be found as Algorithm 3 in Appendix A. The essence of the algorithm is to decouple the descriptor system (2.1) into its proper and improper portions and then reduce each of the portions separately. In addition to truncating the states that are difficult to control and/or to observe, GSHSR also removes those that are uncontrollable and/or unobservable. The algorithm utilizes a collection of solvers for (generalized) Lyapunov and (generalized) Sylvester matrix equations.

Specifically, as suggested in [6], to solve the generalized Sylvester equations (A.1), we use the generalized Schur method by Kågström and Westin [5]. The upper triangular Cholesky factors $\mathbf{R}_f$, $\mathbf{P}_f^T$, $\mathbf{R}_\infty$ and $\mathbf{P}_\infty^T$ of the solutions of the generalized Lyapunov equations (A.2) can be determined without computing the solutions themselves by using the generalized Hammarling method by Hammarling [3] and Penzl [8].

As mentioned above, GSHSR is a balanced truncation method. Therefore, there exist global error bounds for the approximation $\hat{\mathbf{\Sigma}}$. Since the error bounds are in terms of the Hankel singular values of $\mathbf{\Sigma}$, the quality of the approximation depends on the decay of the Hankel singular values. In other words, GSHSR is effective in model order reduction if the Hankel singular values of $\mathbf{\Sigma}$ decay rapidly. In fact, this is a common feature of all of the balanced truncation techniques.

2.4. Optimal $\mathcal{H}_2$. The optimal $\mathcal{H}_2$ is classified as an SVD-based scheme in [1]. The reason is that it solves the following model order reduction problem: Given a stable system $\mathbf{\Sigma}$, an approximation $\hat{\mathbf{\Sigma}}$ is sought to satisfy the following conditions:

$$\sigma_{k+1}(\mathbf{\Sigma}) \leq \|\mathbf{\Sigma} - \hat{\mathbf{\Sigma}}\|_{\mathcal{H}_2} \leq \epsilon < \sigma_k(\mathbf{\Sigma}).$$

Therefore, the construction of the reduced system $\hat{\mathbf{\Sigma}}$ in this framework can mimic the procedure as presented in Section 2.3. However, in 2008, Gugercin et al. [2] observe the equivalence of the local optimality conditions for the model order reduction problem in the two different frameworks: interpolation-based and Lyapunov-based. This result gives birth to a new direction using the interpolation properties to construct an approximation $\hat{\mathbf{\Sigma}}$ without solving the two Lyapunov equations. In this paper, we consider IRKA, a technique that takes advantage of this result.

2.4.1. IRKA. The Iterative Rational Krylov Algorithm (IRKA) is proposed by Gugercin et al. [2]. The original algorithm as shown in [2] is proposed to work with non-descriptor systems. The pseudocode for a generalization of the original algorithm to descriptor systems (2.1) can be found as Algorithm 4 in Appendix A. As mentioned above, by using the interpolation properties, constructing an approximation $\hat{\mathbf{\Sigma}}$ can be done iteratively and without solving the two Lyapunov equations.

Despite these computational advantages, IRKA does not guarantee stability for the approximation $\hat{\mathbf{\Sigma}}$. The success of the iteration depends on the convergence of shifts, which is unpredictable and depends on initial guesses. In addition, for MIMO descriptor systems where ill-conditioned generalized eigenvalue problems have to be solved at each iteration, shifts at infinity have to be taken care of. Essentially, one needs to remove the subspace corresponding to the shifts at infinity from the computation. Our current approach is to identify the unwanted subspace at each iteration and then replace it by some random subspace of the same dimension. This subspace replacement approach retains the size of the successive reduced systems. However, the collection of shifts keeps getting polluted by the replacements, which makes it very hard to achieve any convergence in shifts.

One resolution may be to remove the unwanted subspace without replacement. A drawback of this subspace removal approach is that the sizes of the successive reduced systems get smaller and smaller, reducing the system to an unacceptable size before any convergence of shifts can be observed. Another potential approach to resolving the issue of shifts at infinity may be to decouple the proper and improper portions of the descriptor system (2.1) similar to the approach in GSHSR and then reduce each of the portions separately.

With the current implementation with the subspace replacement approach, the IRKA algorithm for MIMO descriptor systems exhibits very unpredictable behavior with regards to the convergence of shifts, stability and formation of reduced systems.

3. Numerical Analysis and Results. In this section, we consider three different interconnect networks: an RC ladder, an RLC ladder, and an RLC mesh. The techniques presented in Section 2 are applied to these systems to construct reduced models. We compare the frequency responses of the original and reduced models and also demonstrate that by integrating the reduced models instead of the original models into larger circuits, we obtain similar transient simulation responses in much less time.

3.1. Test Models. We study three different RC/RLC networks, which are typically used to model the interconnects between devices on electronic chips. The descriptions of the models are as follows:

RC Ladder Circuit

The SISO RC ladder circuit shown in Figure 3.1 is excited by a voltage source V_s and has 100 blocks of RC cells, i.e., $q = 100$. The modified nodal analysis (MNA) formulation results in a SISO LTI system of size $n = 102$. The numerical values of the resistor and capacitor are $R_i = 10\Omega$ and $C_i = 1pF$, for $i = 1, \ldots, q$.

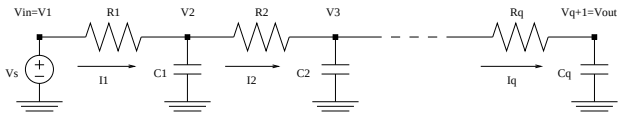


FIG. 3.1. *SISO RC ladder circuit.*

The MIMO RC ladder circuit model is obtained by replacing the last capacitor C_q in the SISO RC ladder circuit (Figure 3.1) by another voltage source V_s. Per the discussion in Section 2, this is the modified version of the original SISO RC ladder circuit that will be used for integration. While symmetry does not make it necessary to distinguish the two ports, for consistency and clarification we call the input port (V_{in}) of the original RC ladder 1 and the output port (V_{out}) 2. The MNA formulation of this circuit results in a MIMO LTI system of size $n = 103$. The numerical values of the resistor and capacitor are the same as for the SISO RC ladder circuit.

RLC Ladder Circuit

The SISO RLC ladder circuit model shown in Figure 3.2 is excited by a voltage source V_s and has 100 blocks of RLC cells, i.e., $q = 100$. The MNA formulation of this circuit results in a SISO LTI system of size $n = 302$. The numerical values for the resistor, inductor and capacitor are $R_i = 0.2\Omega$, $L_i = 1nH$ and $C_i = 0.5pF$, for $i = 1, \ldots, q$.

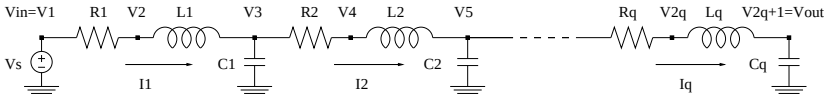


FIG. 3.2. *SISO RLC ladder circuit.*

The MIMO RLC ladder circuit model is obtained by replacing the last capacitor C_q in the SISO RLC ladder circuit (Figure 3.2) by another voltage source V_s. Per the discussion in Section 2, this is the modified version of the original SISO RLC ladder circuit that will be used for integration. We call the input port (V_{in}) 1 and the output port (V_{out}) 2. The MNA

formulation of this circuit results in a MIMO LTI system of size $n = 303$. The numerical values of the resistor, inductor and capacitor are the same as for the SISO RLC ladder circuit.

RLC Ladder Mesh

The MIMO RLC ladder mesh has two input ports and two output ports as depicted in Figure 3.3. Each of the RLC blocks consists of 30 RLC cells similar to the ladder in Figure 3.2. The MNA formulation of this circuit results in a MIMO LTI system of size $n = 1083$. The numerical values of all the resistors, inductors and capacitors are $R = 0.1\Omega$, $L = 0.15nH$ and $C = 0.5pF$, respectively.

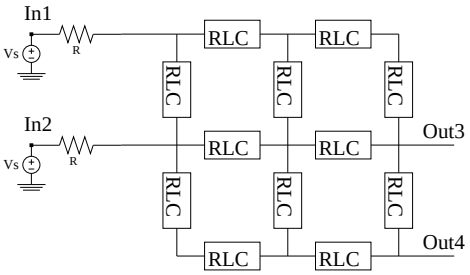


FIG. 3.3. *MIMO RLC ladder grid.*

3.2. Numerical Results. We first present a frequency response comparison of the reduced systems computed using the four previously discussed techniques. This is followed by a comparison of the time-domain responses of a larger circuit that includes either the original interconnect model or a reduced model of the interconnect.

Frequency Responses of SISO RC Ladder Circuit

With the original system's size of $n = 102$, we construct a reduced system of size $k = 10$ using each of the four presented techniques. The frequency responses between the input at the voltage source and the output at the final node of the original and reduced systems are shown in Figure 3.4. From these plots we see that the frequency responses of the reduced systems computed by IRKA and GSHSR are the closest to those of the original. Figure 3.4(c) presents the rapid decay of the Hankel singular values of the system, which illustrates why the reduced system resulting from GSHSR approximates the original system very well. Figure 3.4(d) shows the convergence history of IRKA shifts. Recall that out of the four techniques, PRIMA and GSHSR are the only two that guarantee stable reduced systems. For this circuit, the IRKA shifts converge and it is observed that the resulting reduced system is stable.

Frequency Responses of SISO RLC Ladder Circuit

With the original system's size of $n = 302$, we construct a reduced system of size $k = 60$ using each of the four techniques. The frequency responses between the input at the voltage source and the output at the final node of the original and reduced systems are shown in Figure 3.5. We see that the frequency responses of the resulting reduced systems do not match the original very well in the high range of frequency, especially in magnitude. The slow decay of the Hankel singular values illustrated in Figure 3.5(c) explains why GSHSR does not perform well on this RLC ladder circuit. There is still a lot of information about the original system left out when we construct a reduced system using only the 60 largest Hankel singular values. For a thorough study on how the slow decay in the Hankel singular values of the system

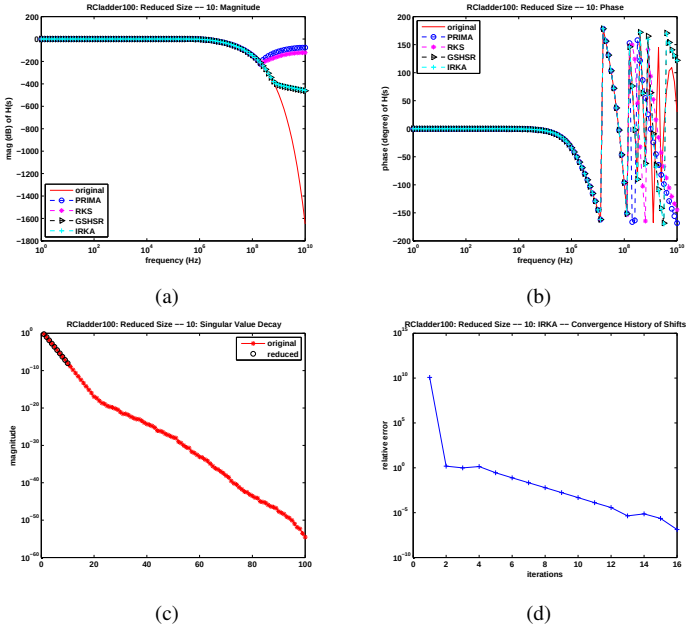


FIG. 3.4. *SISO RC ladder circuit: Magnitudes and phases of the frequency responses of the original ($n = 102$) and reduced ($k = 10$) systems are compared in (a) and (b), respectively. Part (c) shows the Hankel singular value decay and (d) shows the convergence history of IRKA shifts.*

affects the quality of the reduced system computed by GSHSR, see Appendix C.

Frequency Responses of MIMO RC Ladder Circuit

For this circuit, where the original system size is $n = 103$, we construct a reduced system of size $k = 20$ using each of the four presented techniques. Figures 3.6(a) and (b) show the frequency responses in magnitude and phase of the original and reduced systems between Input 1 and Output 2. Due to the rapid decay in the Hankel singular values observed in Figure 3.6(c), the reduced system computed using GSHSR again appears to be the best approximation. As mentioned in Section 2, IRKA does not work well for ill-conditioned MIMO systems and Figure 3.6(d) illustrates that no convergence in shifts is obtained for this system.

Frequency Responses of MIMO RLC Ladder Circuit

For this circuit, where the original system size is $n = 303$, we construct a reduced system of size $k = 80$ using each of the four presented techniques. Figures 3.7(a) and (b) show the frequency responses in magnitude and phase of the original and reduced systems between Input 2 and Output 1. In this case, the reduced systems computed by PRIMA and RKS appear to be good approximations to the original system. The slow decay in the Hankel singular values observed in Figure 3.7(c) prevents GSHSR from approximating the original system well for this small of a reduced model. A similar study on how the slow decay in the Hankel singular values of the system affects the quality of the reduced system computed by GSHSR can be found in Appendix C. Again, IRKA does not work well for ill-conditioned MIMO systems and Figure 3.7(d) illustrates that no convergence in shifts is obtained for this system.

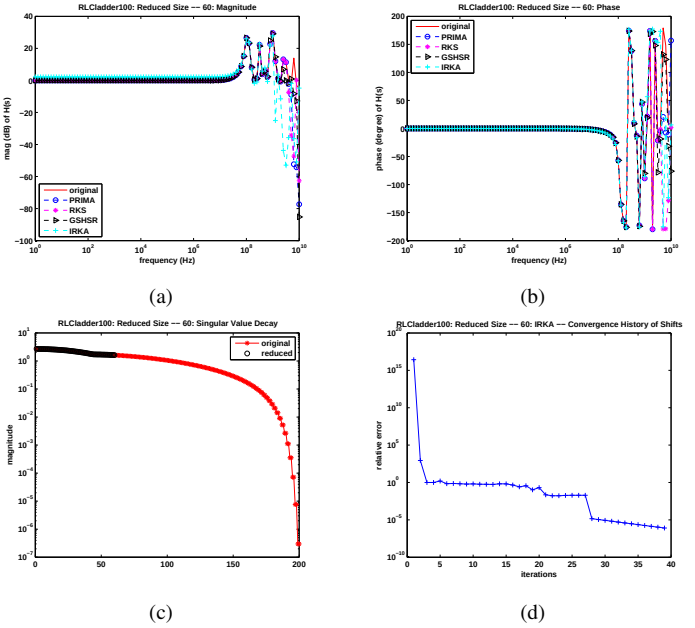


FIG. 3.5. *SISO RLC ladder circuit: Magnitudes and phases of the frequency responses of the original ($n = 302$) and reduced ($k = 60$) systems are compared in (a) and (b), respectively. Part (c) shows the Hankel singular value decay and (d) shows the convergence history of IRKA shifts.*

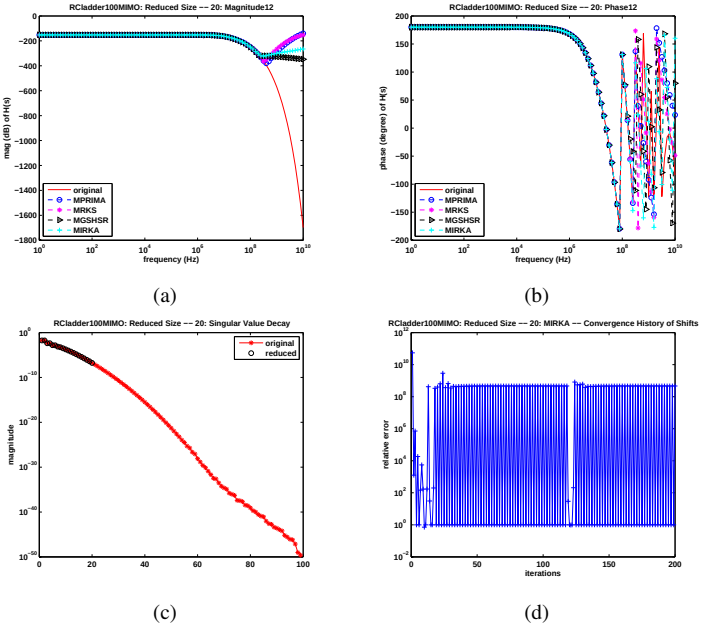


FIG. 3.6. *MIMO RC ladder circuit: The (1,2)-magnitudes and phases of the frequency responses of the original ($n = 103$) and reduced ($k = 20$) systems are compared in (a) and (b), respectively. Part (c) shows the fast decay of the Hankel singular values and (d) shows the lack of convergence in the MIRKA shifts.*

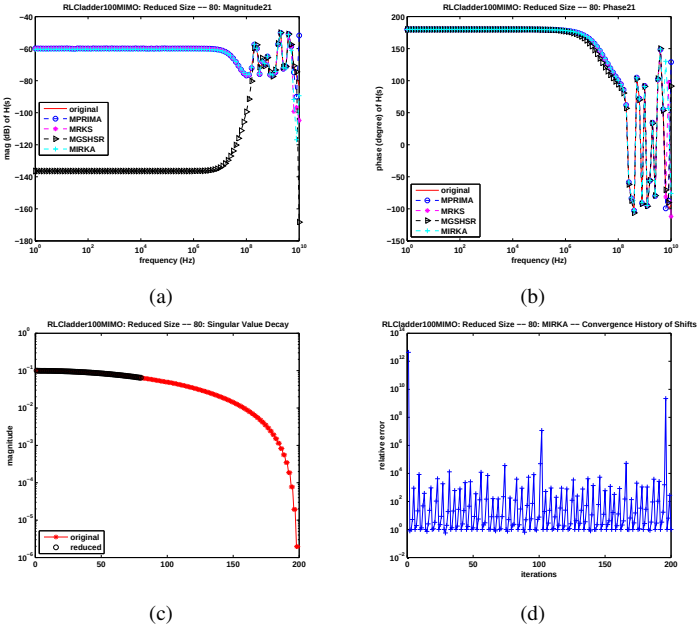


FIG. 3.7. *MIMO RLC ladder circuit: The (2,1)-magnitudes and phases of the frequency responses of the original ($n = 303$) and reduced ($k = 80$) systems are compared in (a) and (b), respectively. Part (c) shows the slow decay of the Hankel singular values and (d) shows the lack of convergence in the MIRKA shifts.*

Frequency Responses of MIMO RLC Ladder Mesh

For this circuit, where the original system size is $n = 1083$, we construct a reduced system of size $k = 60$. Since IRKA is very unstable for this system, we only present the results of the remaining three techniques. Figures 3.8(a) and (b) show the frequency responses in magnitude and phase of the original and reduced systems between Input 1 and Output 3. In this case, the reduced systems computed using PRIMA and RKS appear to be very good approximations to the original system despite their relatively small size. Similar to the SISO and MIMO RLC ladder circuits, the slow decay in the Hankel singular values prevents GSHSR from resulting in a reduced system of size $k = 60$ that well approximates the original system. For a study on how the slow decay in the Hankel singular values of the system affects the quality of the reduced system computed by GSHSR for this system, see Appendix C.

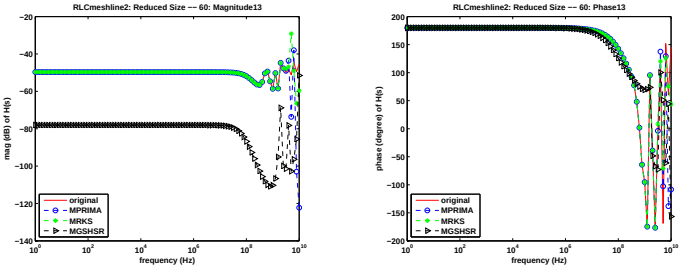


FIG. 3.8. *MIMO RLC ladder mesh: The (1,3)-magnitudes and phases of the frequency responses of the original ($n = 1083$) and reduced ($k = 60$) systems are compared in (a) and (b), respectively.*

Transient Simulations

In the following, we present a comparison of the time-domain responses of a larger circuit that includes an inverter and either the original lossy transmission line or a reduced model of the interconnect. The circuit can be seen in Figure 3.9.

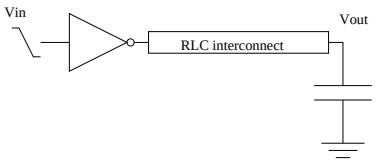
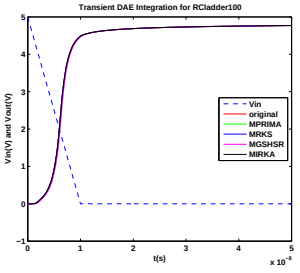


FIG. 3.9. *A simple interconnect circuit with an inverter.*

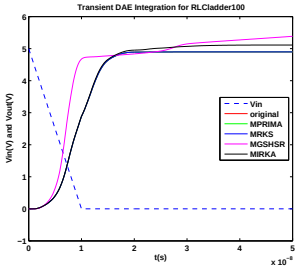
The resulting transient DAE simulations are shown in Figures 3.10 and 3.11 for the circuit in Figure 3.9 with the RC ladder and RLC ladder interconnect, respectively. The time window for the simulations is from 0 to 50 ns. For the circuit with the RC ladder interconnect, the approximations are quite indistinguishable from the exact response. (This agrees well with the results in the frequency responses in Figure 3.6.) The transient simulation time for the inverter circuit with a reduced interconnect model is reduced by at least one order of magnitude from the original. For the circuit with the RLC ladder interconnect, the approximations associated with PRIMA and RKS are indistinguishable from the exact response, while those associated with GSHSR and IRKA do not quite capture the behavior of the response of the original model. (This also agrees with the results in the frequency responses in Figure 3.7.) Similarly, the transient simulation time for the inverter circuit with a reduced interconnect model is reduced by at least one order of magnitude from the original.

For the RLC ladder mesh in Figure 3.3, the setup is similar to that in Figure 3.9. The inputs are connected to CMOS inverters. One of the drivers is switching while the other is quiet. We present the resulting transient DAE simulation observed at Output 3 in Figure 3.12. The simulation observed at Output 4 is the same. The time window for the simulation is from 0 to 100 ns. The approximations associated with PRIMA and RKS are indistinguishable from the exact response, while that associated with GSHSR do not capture the behavior of the response of the original model. The transient simulation time for the inverter circuit with a reduced interconnect model is reduced significantly by at least 30 times from the original.



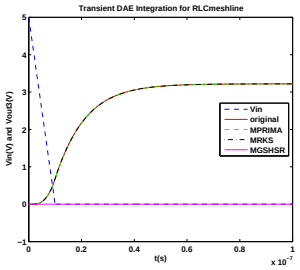
Simulation Time	
Model	*Time(s)*
Original	303.725
PRIMA	23.455
RKS	25.923
GSHSR	19.915
IRKA	31.374

FIG. 3.10. *Inverter + RC ladder circuit: Comparison of transient simulation output and time for the circuit in Figure 3.9. The time window for the simulation is from* 0 *to* 50 *ns.*



Simulation Time	
Model	*Time(s)*
Original	441.537
PRIMA	44.941
RKS	32.122
GSHSR	31.902
IRKA	42.473

FIG. 3.11. *Inverter + RLC ladder circuit: Comparison of transient simulation output and time for the circuit in Figure 3.9. The time window for the simulation is from* 0 *to* 50 *ns.*



Simulation Time	
Model	*Time(s)*
Original	1055.82
PRIMA	31.60
RKS	20.84
GSHSR	28.19

FIG. 3.12. *Inverter + RLC ladder mesh: Comparison of transient simulation output and time for the circuit in Figure 3.9. The time window for the simulation is from* 0 *to* 100 *ns.*

4. Conclusions. With the analysis and numerical experiments, the following observations and suggestions are made. Given an LTI system, its Hankel singular values should be computed and studied. If a rapid decay in the Hankel singular values is observed, balanced truncation reduction techniques (e.g., GSHSR) should be the methods of choice to construct reduced systems. The reasons are in the guarantee of stability for the reduced systems and the availability of a priori global error bounds. As mentioned before, a number of efforts have been made in solving the associated (generalized) Lyapunov and (generalized) Sylvester equations iteratively. Thus, in the case of large-scale systems, one should take advantage of these iterative algorithms for the feasibility of solving these equations.

Optimal $\mathcal{H}_2$ reduction is locally equivalent in the interpolation-based and Lyapunov-based frameworks, and hence, IRKA by using interpolation conditions bypasses the problem of solving the associated Lyapunov equations. The technique is then efficient for large-scale systems by taking advantage of Krylov subspace methods. However, IRKA's unpredictability of shift convergence and issue of shifts at infinity for ill-conditioned descriptor systems remain open questions. In addition, since the equivalence of the optimality conditions for the reduction problem in the two frameworks is local, IRKA does not necessarily inherent the availability of a priori global error bounds which may result from the properties in the Lyapunov-based framework.

If no rapid decay in the Hankel singular values is observed, then balanced truncation techniques are not good choices when significantly-small-size reduced models are desired. Even though the resulting reduced systems are guaranteed to be stable, they do not approximate the original system well. In this case, techniques from other schemes should be considered. PRIMA would be a good choice as it guarantees passivity and hence, stability for the reduced

systems. Note that these guaranteed properties are only for systems resulting from the MNA formulation. One drawback of PRIMA is that it does not have a priori global error bounds. However, as mentioned above, local error estimation can be achieved during the construction of reduced systems.

REFERENCES

[1] A. C. Antoulas, D. C. Sorensen, and S. Gugercin, *A survey of model reduction methods for large-scale systems*, Structured Matrices in Mathematics, Computer Science and Engineering, Vol. I. Comtemporaty Mathematics Series, 280 (2001), pp. 193–219.

[2] S. Gugercin, A. C. Antoulas, and C. Beattie, *$\mathcal{H}_2$ model reduction for large-scale linear dynamical systems*, SIAM Journals on Matrix Analysis and Applications, 30 (2008), pp. 609–638.

[3] S. J. Hammarling, *Numerical solution of the stable, non-negative definite Lyapunov equation*, IMA Journal of Numerical Analysis, 2 (1982), pp. 303–323.

[4] P. J. Heres, *Robust and Efficient Krylov Subspace Methods for Model Order Reduction*, PhD thesis, Eindhoven University of Technology, 2005.

[5] B. Kågström and L. Westin, *Generalized Schur methods with condition estimators for solving the generalized Sylvester equation*, IEEE Transactions on Automatic Control, 34 (1989), pp. 745–751.

[6] V. Mehrmann and T. Stykel, *Balanced truncation model reduction for large-scale systems in descriptor form*, Dimension Reduction of Large-Scale Systems, Lect. Notes Comput. Sci. Eng., 45 (2005), pp. 83–115.

[7] A. Odabasioglu, M. Celik, and L. T. Pileggi, *PRIMA: Passive reduced-order interconnect macromodeling algorithm*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 17 (1998), pp. 645–654.

[8] T. Penzl, *Numerical solution of generalized Lyapunov equations*, Advances in Computational Mathematics, 8 (1998), pp. 33–48.

[9] A. Ruhe, *Rational Krylov algorithms for nonsymmetric eigenvalue problems. ii. matrix pairs*, Linear Algebra and Its Applications, 197, 198 (1994), pp. 283–295.

[10] D. Skoogh, *A rational Krylov method for model order reduction*, Blåserien, 47 (1998).

Appendix

A. Model Order Reduction Algorithms. Algorithm 1. *PRIMA Method*

1. *Obtain an expansion point* s_0 *and the state space realization of the original system* $\mathbf{\Sigma} \equiv (\mathbf{C}, \mathbf{G}, \mathbf{B}, \mathbf{L})$.
2. *Formally set* $\mathbf{A} = -(\mathbf{G} + s_0\mathbf{C})^{-1}\mathbf{C}$ *and* $\mathbf{R} = (\mathbf{G} + s_0\mathbf{C})^{-1}\mathbf{B}$.
3. *Use a block Krylov subspace method to construct a unitary projection matrix* $\mathbf{V}$ *such that* $\mathbf{V} = [\mathbf{v}_1\ \mathbf{v}_2\ \ldots\ \mathbf{v}_k]$, *where span*$(\mathbf{V}) = \mathcal{K}_k(\mathbf{A}, \mathbf{R})$.
4. *Compute* $\hat{\mathbf{C}} = \mathbf{V}^T\mathbf{C}\mathbf{V}$, $\hat{\mathbf{G}} = \mathbf{V}^T\mathbf{G}\mathbf{V}$, $\hat{\mathbf{B}} = \mathbf{V}^T\mathbf{B}$ *and* $\hat{\mathbf{L}}^T = \mathbf{L}^T\mathbf{V}$ *to obtain the state space realization of the reduced system* $\hat{\mathbf{\Sigma}} \equiv (\hat{\mathbf{C}}, \hat{\mathbf{G}}, \hat{\mathbf{B}}, \hat{\mathbf{L}})$.

Algorithm 2. *RKS Method*

1. *Select a set of m expansion points* s_j *whose frequencies are* f_j *such that* $p\sum_{j=1}^{m} f_j = k$, *where p and k are the number of inputs of* $\mathbf{\Sigma}$ *and the desired size of the reduced system* $\hat{\mathbf{\Sigma}}$, *respectively. In addition, obtain the state space realization of the original system* $\mathbf{\Sigma} \equiv (\mathbf{C}, \mathbf{G}, \mathbf{B}, \mathbf{L})$.
2. *Compute* $[\mathbf{Q}, \mathbf{R}] = (\mathbf{G} + s_1\mathbf{C})^{-1}\mathbf{B}$, *the QR factorization.*
3. *With* $\mathbf{Q}$ *being the initial block, use the block rational Krylov algorithm to construct the three matrices* $\mathbf{V} \in \mathbb{R}^{n\times(k+p)}$, $\mathbf{H} \in \mathbb{R}^{(k+p)\times(k+p)}$ *and* $\mathbf{K} \in \mathbb{R}^{(k+p)\times(k+p)}$ *such that* $\mathbf{GVH} = \mathbf{CVK}$.
4. *Let* $\hat{\mathbf{C}} = \mathbf{H}(1:k, 1:k)$ *and* $\hat{\mathbf{G}} = \mathbf{K}(1:k, 1:k)$ *and compute*

$$\hat{\mathbf{L}}^T = \mathbf{L}^T\mathbf{V}(:, 1:k)[\mathbf{K}(1:k, 1:k) + s_1\mathbf{H}(1:k, 1:k)]$$
$$\text{and} \quad \hat{\mathbf{B}} = \mathbf{E}_1\mathbf{R},$$

where $\mathbf{E}_1 \in \mathbb{R}^{n\times p}$ *with an identity matrix in its upper* $p \times p$ *block and zeros everywhere else. The resulting reduced system is* $\hat{\mathbf{\Sigma}} \equiv (\hat{\mathbf{C}}, \hat{\mathbf{G}}, \hat{\mathbf{B}}, \hat{\mathbf{L}})$.

Algorithm 3. *Generalized Schur-Hammarling Square Root Method*

1. *Obtain the state space realization of the original system* $\mathbf{\Sigma} \equiv (\mathbf{C}, \mathbf{G}, \mathbf{B}, \mathbf{L})$.
2. *Compute the generalized Schur form*

$$\mathbf{C} = \mathbf{V}\begin{bmatrix} \mathbf{C}_f & \mathbf{C}_u \\ \mathbf{0} & \mathbf{C}_\infty \end{bmatrix}\mathbf{U}^T \quad \text{and} \quad \mathbf{G} = \mathbf{V}\begin{bmatrix} \mathbf{G}_f & \mathbf{G}_u \\ \mathbf{0} & \mathbf{G}_\infty \end{bmatrix}\mathbf{U}^T,$$

where $\mathbf{U}$ *and* $\mathbf{V}$ *are orthogonal,* $\mathbf{C}_f$ *upper triangular nonsingular,* $\mathbf{C}_\infty$ *upper triangular nilpotent,* $\mathbf{G}_f$ *upper quasi-triangular and* $\mathbf{G}_\infty$ *upper triangular nonsingular.*

3. *Compute the matrices and then partition them conformally with respect to C and G as follows*

$$\mathbf{V}^T\mathbf{B} = \begin{bmatrix} \mathbf{B}_u \\ \mathbf{B}_\infty \end{bmatrix} \quad \text{and} \quad \mathbf{L}^T\mathbf{U} = \begin{bmatrix} \mathbf{L}_f^T & \mathbf{L}_u^T \end{bmatrix}.$$

4. *Solve the systems of generalized Sylvester equations for* $\mathbf{Y}$ *and* $\mathbf{Z}$

$$\begin{aligned} \mathbf{C}_f\mathbf{Y} - \mathbf{Z}\mathbf{C}_\infty &= -\mathbf{C}_u, \\ \mathbf{G}_f\mathbf{Y} - \mathbf{Z}\mathbf{G}_\infty &= -\mathbf{G}_u. \end{aligned} \tag{A.1}$$

5. *Compute the Cholesky factors* $\mathbf{R}_f$, $\mathbf{P}_f$, $\mathbf{R}_\infty$ *and* $\mathbf{P}_\infty$ *of the solutions* $\mathbf{X}_{pc} = \mathbf{R}_f\mathbf{R}_f^T$,

$\mathbf{X}_{po} = \mathbf{P}_f\mathbf{P}_f^T$, $\mathbf{X}_{ic} = \mathbf{R}_\infty\mathbf{R}_\infty^T$ *and* $\mathbf{X}_{io} = \mathbf{P}_\infty\mathbf{P}_\infty^T$ *of the generalized Lyapunov equations*

$$\begin{aligned}
\mathbf{C}_f\mathbf{X}_{pc}\mathbf{G}_f^T + \mathbf{G}_f\mathbf{X}_{pc}\mathbf{C}_f^T &= (\mathbf{B}_u - \mathbf{Z}\mathbf{B}_\infty)(\mathbf{B}_u - \mathbf{Z}\mathbf{B}_\infty)^T,\\
\mathbf{C}_f^T\mathbf{X}_{po}\mathbf{G}_f + \mathbf{G}_f^T\mathbf{X}_{pc}\mathbf{C}_f &= \mathbf{L}_f\mathbf{L}_f^T,\\
\mathbf{G}_\infty\mathbf{X}_{ic}\mathbf{G}_\infty^T - \mathbf{C}_\infty\mathbf{X}_{ic}\mathbf{C}_\infty^T &= \mathbf{B}_\infty\mathbf{B}_\infty^T,\\
\mathbf{G}_\infty^T\mathbf{X}_{io}\mathbf{G}_\infty - \mathbf{C}_\infty^T\mathbf{X}_{io}\mathbf{C}_\infty &= (\mathbf{Y}^T\mathbf{L}_f + \mathbf{L}_u)(\mathbf{Y}^T\mathbf{L}_f + \mathbf{L}_u)^T.
\end{aligned} \tag{A.2}$$

6. *Compute the skinny singular value decompositions*

$$\mathbf{P}_f^T\mathbf{C}_f\mathbf{R}_f = \begin{bmatrix} \mathbf{U}_1 & \mathbf{U}_2 \end{bmatrix}\begin{bmatrix} \mathbf{\Sigma}_1 & \\ & \mathbf{\Sigma}_2 \end{bmatrix}\begin{bmatrix} \mathbf{V}_1 & \mathbf{V}_2 \end{bmatrix},$$
$$\textit{and} \quad \mathbf{P}_\infty^T(-\mathbf{G}_\infty)\mathbf{R}_\infty = \mathbf{U}_3\mathbf{\Theta}_3\mathbf{V}_3^T,$$

   where $\begin{bmatrix} \mathbf{U}_1 & \mathbf{U}_2 \end{bmatrix}$, $\begin{bmatrix} \mathbf{V}_1 & \mathbf{V}_2 \end{bmatrix}$, $\mathbf{U}_3$ *and* $\mathbf{V}_3$ *have orthonormal columns,* $\mathbf{\Sigma}_1 = diag(\zeta_1, \ldots, \zeta_{l_f})$, $\mathbf{\Sigma}_2 = diag(\zeta_{l_f+1}, \ldots, \zeta_r)$, $\mathbf{\Theta}_3 = diag(\theta_1, \ldots \theta_{l_\infty})$ *with* $r = rank(\mathbf{P}_f^T\mathbf{C}_f\mathbf{R}_f)$, $l_\infty = rank(\mathbf{P}_\infty^T(-\mathbf{G}_\infty)\mathbf{R}_\infty)$ *and* l_f *is the desired size of the proper portion of the reduced system* $\hat{\mathbf{\Sigma}}$.
7. *Compute* $\mathbf{W}_f = \mathbf{P}_f\mathbf{U}_1\mathbf{\Sigma}_1^{-1/2}$, $\mathbf{W}_\infty = \mathbf{P}_\infty\mathbf{U}_3\mathbf{\Theta}_3^{-1/2}$, $\mathbf{T}_f = \mathbf{R}_f\mathbf{V}_1\mathbf{\Sigma}_1^{-1/2}$ *and* $\mathbf{T}_\infty = \mathbf{R}_\infty\mathbf{V}_3\mathbf{\Theta}_3^{-1/2}$.
8. *Compute the reduced-order system* $\hat{\mathbf{\Sigma}} \equiv (\hat{\mathbf{C}}, \hat{\mathbf{G}}, \hat{\mathbf{B}}, \hat{\mathbf{L}})$ *as follows*

$$\hat{\mathbf{C}} = \begin{bmatrix} \mathbf{I}_{l_f} & \mathbf{0} \\ \mathbf{0} & \mathbf{W}_\infty^T\mathbf{C}_\infty\mathbf{T}_\infty \end{bmatrix}, \quad \hat{\mathbf{G}} = \begin{bmatrix} \mathbf{W}_f^T\mathbf{G}_f\mathbf{T}_f & \mathbf{0} \\ \mathbf{0} & -\mathbf{I}_{l_\infty} \end{bmatrix},$$
$$\hat{\mathbf{B}} = \begin{bmatrix} \mathbf{W}_f^T(\mathbf{B}_u - \mathbf{Z}\mathbf{B}_\infty) \\ \mathbf{W}_\infty^T\mathbf{B}_\infty \end{bmatrix}, \quad \hat{\mathbf{L}}^T = \begin{bmatrix} \mathbf{L}_f^T\mathbf{T}_f & (\mathbf{L}_f^T\mathbf{Y} + \mathbf{L}_u^T)\mathbf{T}_\infty \end{bmatrix}.$$

Algorithm 4. *IRKA Method for SISO Systems*

1. *Make an initial selection of shifts* σ_i *for* $i = 1, \ldots, k$ *that is closed under conjugation, where* k *is the desized size of the reduced system* $\hat{\mathbf{\Sigma}}$. *In addition, fix a convergence tolerance* `tol`.
2. *Construct* $\mathbf{V}_k$ *and* $\mathbf{W}_k$ *so that*

$$R(\mathbf{V}_k) = span\{(\sigma_1\mathbf{C} + \mathbf{G})^{-1}\mathbf{B}, \ldots, (\sigma_k\mathbf{C} + \mathbf{G})^{-1}\mathbf{B}\}$$
$$R(\mathbf{W}_k) = span\{(\sigma_1\mathbf{C}^T + \mathbf{G}^T)^{-1}\mathbf{L}, \ldots, (\sigma_k\mathbf{C}^T + \mathbf{G}^T)^{-1}\mathbf{L}\}.$$

3. *while (relative change in* $\{\sigma_i\} >$ `tol`*)*
   (a) $\mathbf{C}_k = (\mathbf{W}_k^T\mathbf{V}_k)^{-1}\mathbf{W}_k^T\mathbf{C}\mathbf{V}_k$
   (b) $\mathbf{G}_k = (\mathbf{W}_k^T\mathbf{V}_k)^{-1}\mathbf{W}_k^T\mathbf{G}\mathbf{V}_k$
   (c) *Assign* $\sigma_i \leftarrow \lambda_i(\mathbf{G}_k, \mathbf{C}_k)$ *for* $i = 1, \ldots, k$.
   (d) *Update* $\mathbf{V}_k$ *and* $\mathbf{W}_k$ *so that*

$$R(\mathbf{V}_k) = span\{(\sigma_1\mathbf{C} + \mathbf{G})^{-1}\mathbf{B}, \ldots, (\sigma_k\mathbf{C} + \mathbf{G})^{-1}\mathbf{B}\}$$
$$R(\mathbf{W}_k) = span\{(\sigma_1\mathbf{C}^T + \mathbf{G}^T)^{-1}\mathbf{L}, \ldots, (\sigma_k\mathbf{C}^T + \mathbf{G}^T)^{-1}\mathbf{L}\}.$$

4. *Compute* $\hat{\mathbf{C}} = (\mathbf{W}_k^T\mathbf{V}_k)^{-1}\mathbf{W}_k^T\mathbf{C}\mathbf{V}_k$, $\hat{\mathbf{G}} = (\mathbf{W}_k^T\mathbf{V}_k)^{-1}\mathbf{W}_k^T\mathbf{G}\mathbf{V}_k$, $\hat{\mathbf{B}} = (\mathbf{W}_k^T\mathbf{V}_k)^{-1}\mathbf{W}_k^T\mathbf{B}$ *and* $\hat{\mathbf{L}}^T = \mathbf{L}^T\mathbf{V}_k$ *to obtain the state space realization of the reduced system* $\hat{\mathbf{\Sigma}} \equiv (\hat{\mathbf{C}}, \hat{\mathbf{G}}, \hat{\mathbf{B}}, \hat{\mathbf{L}})$.

As suggested in [2], the algorithm can be extended for MIMO systems as follows: In Algorithm 4, replace

$$(\sigma_i \mathbf{C} + \mathbf{G})^{-1}\mathbf{B} \quad \text{and} \quad (\sigma_i \mathbf{C}^T + \mathbf{G}^T)^{-1}\mathbf{L}$$

respectively by

$$(\sigma_i \mathbf{C} + \mathbf{G})^{-1}\mathbf{B}\mathbf{b}_i \quad \text{and} \quad (\sigma_i \mathbf{C}^T + \mathbf{G}^T)^{-1}\mathbf{L}\mathbf{l}_i,$$

with

$$\mathbf{b}_i^T = \mathbf{y}_i^T \mathbf{B}_k \quad \text{and} \quad \mathbf{l}_i = \mathbf{L}_k^T \mathbf{x}_i,$$

where

$$-\mathbf{G}_k \mathbf{x}_i = \mathbf{C}_k \mathbf{x}_i \lambda_i, \quad -\mathbf{y}_i^T \mathbf{G}_k = \lambda_i \mathbf{y}_i^T \mathbf{C}_k \quad \text{and} \quad \mathbf{y}_i^T \mathbf{x}_i = 1,$$

and $(\mathbf{C}_k, \mathbf{G}_k, \mathbf{B}_k, \mathbf{L}_k)$ is the reduced-order system at each step. The idea is that instead of interpolating $\mathbf{G}(s)$, the transfer function of the original system $\mathbf{\Sigma}$, at the mirror images of the poles of $\hat{\mathbf{G}}(s)$, the transfer function of the reduced system $\hat{\mathbf{\Sigma}}$, as in the SISO case, $\hat{\mathbf{G}}(s)$ for the MIMO case interpolates $\mathbf{G}(s)$ tangentially at the mirror images of the poles of $\hat{\mathbf{G}}(s)$.

B. Error Control for PRIMA. The following is a short summary of the discussion on error control for PRIMA in [4]. Consider an original LTI system $\mathbf{\Sigma} \equiv (\mathbf{C}, \mathbf{G}, \mathbf{B}, \mathbf{L})$ as in (2.1) and its reduced model $\hat{\mathbf{\Sigma}} \equiv (\hat{\mathbf{C}}, \hat{\mathbf{G}}, \hat{\mathbf{B}}, \hat{\mathbf{L}})$ as in (2.2), which results from PRIMA as shown in Algorithm 1 in Appendix A. Also as mentioned in Section 2.2, the corresponding transfer functions are

$$\mathbf{H}(s) = \mathbf{L}^T(s\mathbf{C} + \mathbf{G})^{-1}\mathbf{B},$$
$$\hat{\mathbf{H}}(s) = \hat{\mathbf{L}}^T(s\hat{\mathbf{C}} + \hat{\mathbf{G}})^{-1}\hat{\mathbf{B}}.$$

For a choice of the expansion point s_0, the transfer function $\mathbf{H}(s)$ is well approximated in the range from $s = 0$ to s_0. Choose equally divided points in the interval as in

$$\frac{s_0}{4}, \quad \frac{s_0}{2}, \quad \frac{3s_0}{4}, \quad s_0.$$

Then the error e is defined as follows:

$$e = \frac{\left\| \sum_{k=1}^{4} \mathbf{H}\left(\frac{4s_0}{k}\right) - \hat{\mathbf{H}}\left(\frac{4s_0}{k}\right) \right\|_\infty}{\left\| \sum_{k=1}^{4} \hat{\mathbf{H}}\left(\frac{4s_0}{k}\right) \right\|_\infty}. \tag{B.1}$$

Since $\mathbf{H}(s)$ is quite expensive to compute if the size of the original system $\mathbf{\Sigma}$ is large, the error e in (B.1) can be approximated as follows:

$$e_{appr} = \frac{\left\| \sum_{k=1}^{4} \hat{\mathbf{H}}_q\left(\frac{4s_0}{k}\right) - \hat{\mathbf{H}}_{q-1}\left(\frac{4s_0}{k}\right) \right\|_\infty}{\left\| \sum_{k=1}^{4} \hat{\mathbf{H}}_q\left(\frac{4s_0}{k}\right) \right\|_\infty}, \tag{B.2}$$

where $\hat{\mathbf{H}}_{q-1}(s)$ and $\hat{\mathbf{H}}_q(s)$ are the transfer functions of the two reduced systems $\hat{\mathbf{\Sigma}}_{q-1}$ and $\hat{\mathbf{\Sigma}}_q$ of size $k - p$ and k, respectively.

C. GSHSR on Systems with Slow Decay in Hankel Singular Values. In this section, we present a thorough study on how the slow decay in the Hankel singular values of a system affects the quality of the reduced system.

SISO RLC Ladder Circuit

For the SISO RLC ladder circuit in Figure 3.2, the singular value decay can be seen in Figure C.1(c). Using GSHSR, we construct five different reduced systems whose sizes increase from 20 to 180 with an increment of 40. Figures C.1(a) and (b) show that as the size of the reduced system increases, its responses in magnitude and phase get closer to those of the original system. In addition, the size of the reduced system needs to be quite large for its frequency responses to match those of the original system well. These observations perfectly agree with what is observed in Figure C.1(c). The larger the size of the reduced system, the more information from the original system it obtains, the better an approximation to the original system it presents. In addition, due to the slow decay in the Hankel singular values, the size of the reduced system (which is associated with the number of Hankel singular values) has to be significantly large in order for it to be a good approximation to the original system.

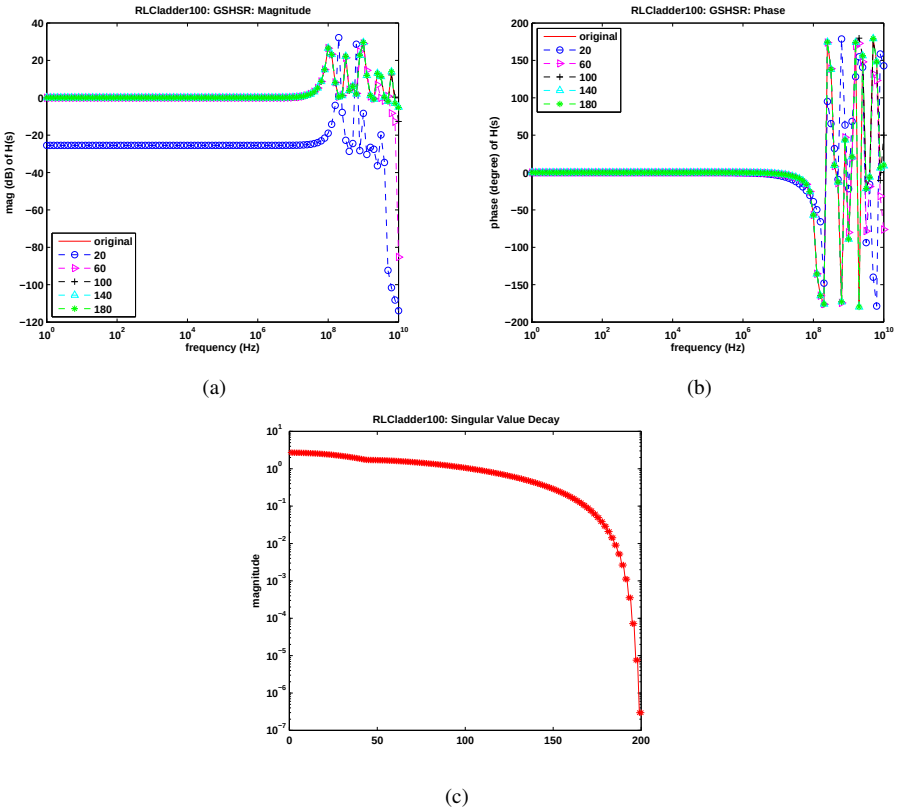


FIG. C.1. *SISO RLC ladder circuit: The magnitudes and phases of the frequency responses of the original system of size* 302 *and reduced systems using GSHSR are shown in (a) and (b) respectively. Since the decay in the Hankel singular values are quite slow as can be seen in (c), reduced systems of small size are not quite good approximations.*

MIMO RLC Ladder Circuit

For the MIMO RLC ladder circuit, the singular value decay can be seen in Figure C.2(c). Using GSHSR, we construct six different reduced systems whose sizes increase from 80 to 180 with an increment of 20. Figures C.2(a) and (b) show that as the size of the reduced system increases, its responses in magnitude and phase get closer to those of the original system. In addition, the size of the reduced system needs to be quite large for its frequency responses to match those of the original system well. These observations perfectly agree with what is observed in Figure C.2(c). The larger the size of the reduced system, the more information from the original system it obtains, the better an approximation to the original system it presents. In addition, due to the slow decay in the Hankel singular values, the size of the reduced system (which is associated the number of Hankel singular values) has to be significantly large in order for it to be a good approximation to the original system.

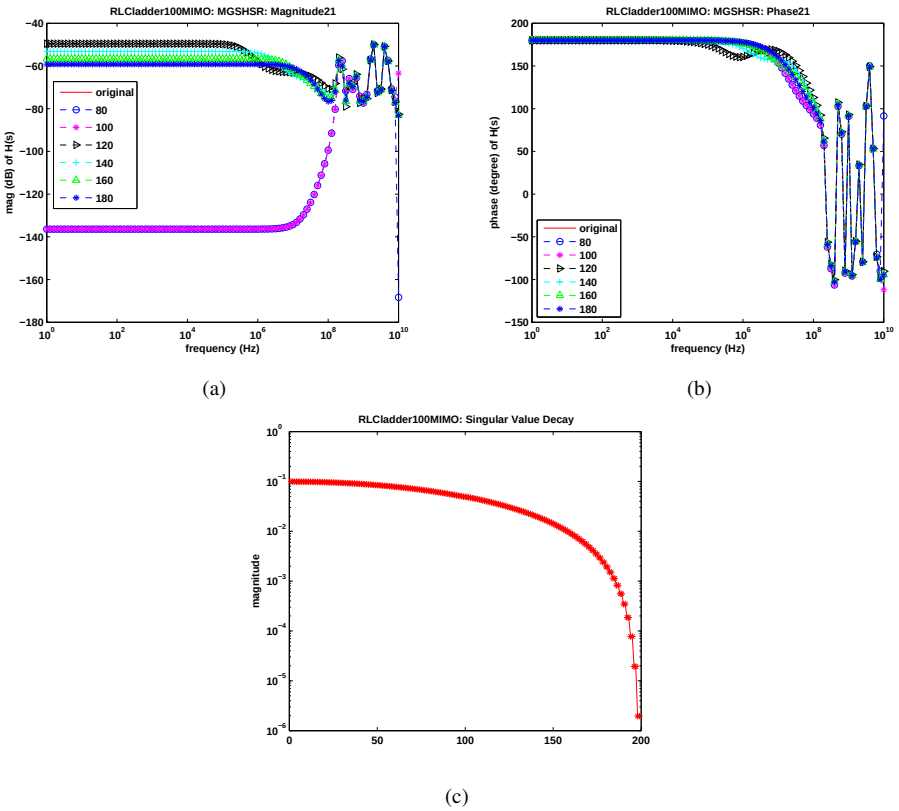


FIG. C.2. *MIMO RLC ladder circuit: The (2,1)-magnitudes and phases of the frequency responses of the original system of size* 303 *and reduced systems using MGSHSR are shown in (a) and (b) respectively. Since the decay in the Hankel singular values are quite slow as can be seen in (c), reduced systems of small size are not quite good approximations.*

MIMO RLC Ladder Mesh

For the MIMO RLC ladder mesh in Figure 3.3, the singular value decay can be seen in Figure C.3(c). Using GSHSR, we construct four different reduced systems whose sizes are 60, 120, 300 and 460. Figures C.3(a) and (b) show that as the size of the reduced system increases, its responses in magnitude and phase get closer to those of the original system. In

addition, the size of the reduced system needs to be quite large for its frequency responses to match those of the original system well. These observations perfectly agree with what is observed in Figure C.3(c). The larger the size of the reduced system, the more information from the original system it obtains, the better an approximation to the original system it presents. In addition, due to the slow decay in the Hankel singular values, the size of the reduced system (which is associated with the number of Hankel singular values) has to be significantly large in order for it to be a good approximation to the original system.

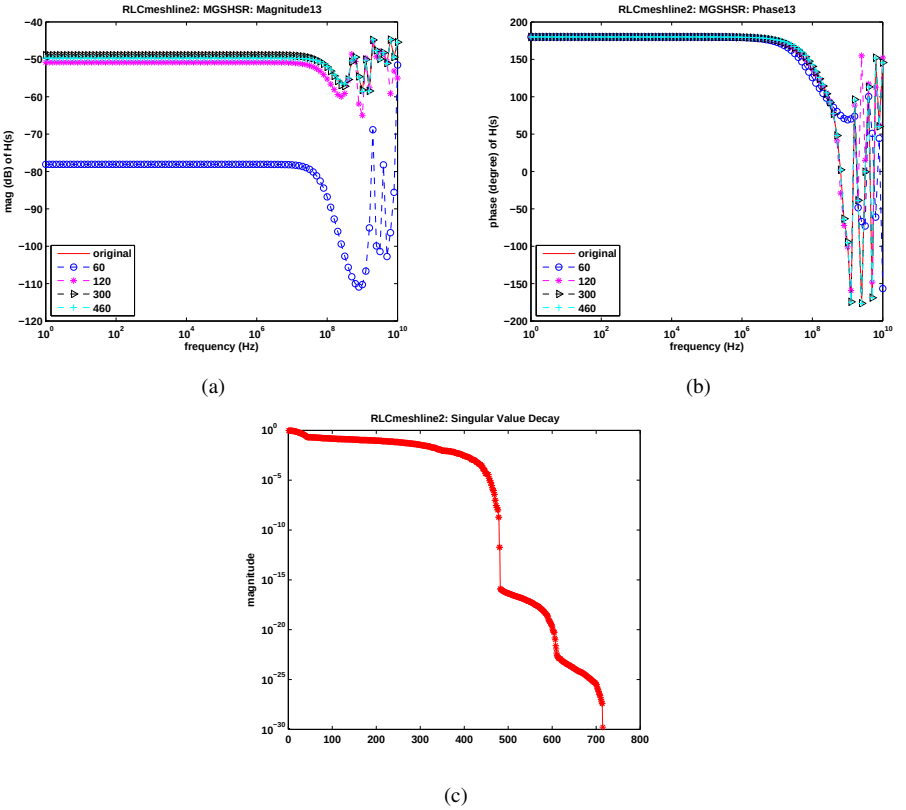


FIG. C.3. *MIMO RLC ladder mesh: The (1,3)-magnitudes and phases of the frequency responses of the original system of size* 1083 *and reduced systems using MGSHSR are shown in (a) and (b) respectively. Since the decay in the Hankel singular values are quite slow as can be seen in (c), reduced systems of small size are not quite good approximations.*

SPARSE–GRID INTEGRATION IN FINITE ELEMENT SPACES

MATTHEW KEEGAN*, DENIS RIDZAL†, AND PAVEL BOCHEV‡

Abstract. We investigate the use of sparse grids, typically used in stochastic sampling, for the numerical integration in "physical" space, i.e. in two and three dimensions, with applications to the finite element solution of PDEs. In addition to studying classical sparse–grid constructions, we introduce an adaptive algorithm for the construction of numerical integration rules on hypercubes, with the particular focus on exact integration of complete polynomial spaces of a given degree. In contrast to the classical sparse–grid rules, our algorithm outperforms the simple tensor–product rule in this regard. We also study the application of the proposed adaptive sparse–grid construction in computing high–order tensor–product finite element PDE approximations on hexahedral grids.

1. Introduction. Numerical computation of continuous multivariate integrals arises in many areas of computational science, such as statistical mechanics, financial mathematics, and the discretization of partial differential equations (PDEs). The most common technique is to approximate such integrals by a *numerical integration rule*, also known as a *quadrature* or *cubature rule*, of the type

$$\int_\Omega f(x)dx \approx \sum_{i\in\mathcal{I}} w_i f(x_i), \tag{1.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a given integrand, $\Omega \subseteq \mathbb{R}^n$ is the domain of integration, $\mathcal{I}$ denotes a finite set of indices i, $x_i \in \mathbb{R}^n$ are integration points or abscissae, and $w_i \in \mathbb{R}$ are the corresponding integration weights.

An important property of a numerical integration rule is its *degree of exactness*, defined as the number m such that the integration rule recovers exactly the analytical value of the integral of all polynomials of total degree m or lower, but fails to integrate at least one polynomial of total degree $m + 1$. If this is true for a given rule, then

$$\left| \int_\Omega f(x)dx - \sum_{i\in\mathcal{I}} w_i f(x_i) \right| \le C \cdot e_m,$$

where $e_m = \inf\{\|p - f\|_\infty : p \in \mathbb{P}^d_m\}$, $\mathbb{P}^d_m$ denotes the space of all multivariate polynomials of dimension d and total degree m, and C is a constant that depends on the type of the numerical integration rule and the domain of integration. It follows from the Stone–Weierstrass approximation theorem that if f is continuous and Ω is compact, $\sum_{i\in\mathcal{I}} w_i f(x_i) \to \int_\Omega f(x)dx$ as $m \to \infty$. At the same time, the higher the desired degree of exactness, the more points are required to approximate the integral, and hence a tradeoff between cost and accuracy is required.

As an example of this tradeoff, solution of PDEs by a finite element method requires numerical integration over so–called *reference elements*, which can be rather costly. If a high–order finite element approximation is desired, the selection of the integration rule that achieves a sufficiently high degree of exactness using the fewest integration points is critical in reducing computational complexity. In this paper, we examine the effectiveness of sparse–grid integration rules, typically used in high–dimensional stochastic sampling and simulation, as a means of numerical integration in finite element spaces. Since sparse–grid rules are typically developed for integration over hypercubes, we limit our presentation to integration domains given by the reference element $[-1, 1]^d$.

*University of California, Los Angeles, mkeegan@math.ucla.edu
†Sandia National Laboratories, dridzal@sandia.gov
‡Sandia National Laboratories, pbboche@sandia.gov

Furthermore, we propose a new set of numerical integration rules, based on sparse–grid constructions, that exactly integrate polynomials of a given total degree, and, for such integrands, are more efficient than the rules most commonly employed in finite element integration on the domain $[-1,1]^d$. Albeit suboptimal, these rules may be desirable because they are easily extendable to arbitrary dimension and arbitrary degree of exactness, which is a significant advantage over the close–to–optimal rules developed in, e.g., [3]. We demonstrate the effectiveness of the proposed rules in three dimensions. Additionally, we present data for four and five dimensions, where they might be used for stochastic sampling and simulation or in constructing higher–dimensional finite element approximations. We also explore the accuracy of finite element PDE solutions obtained using the proposed rules in conjunction with high–order tensor–product finite element bases on hexahedral elements.

The multivariate integration rules studied in this paper are constructed from a variety of one-dimensional integration rules discussed in Section 2. In Section 3 we review multivariate integration formulae, including simple tensor–product and Smolyak's sparse–grid constructions. Section 4 explores the issue of the degree of exactness, which is critical for integration in finite element spaces. In Section 5 we present a set of adaptive sparse–grid rules with the focus on polynomial exactness. Section 6 summarizes numerical results.

2. Univariate Integration Rules. There are three sets of univariate integration rules that we will use in the construction of multivariate rules.

The Gauss rules are the best–known numerical integration rules in the literature. These rules are optimal in the sense that an n-point rule has degree of exactness $2n-1$, the theoretical maximum. The points of the nth Gauss rule are the zeros of the nth–degree Legendre polynomial. These rules are never nested (other than the point 0 for odd–degree rules).

The second set of rules considered are those proposed by Clenshaw and Curtis [2]. In theory they are suboptimal since an n–point rule has degree of exactness n. However numerical studies show that the actual degree of exactness is close to optimal for most integrands [10]. This is due to their construction, based on rapidly converging Chebyshev approximations. The integration points of these rules are the extrema of Chebyshev polynomials combined with the endpoints of the integration domain, 1 and -1. Below we propose using the Clenshaw–Curtis rules with $n = 2^{\ell-1}+1$ points since the rules are then nested.

The third set of rules is that proposed by Kronrod and extended by Patterson [8]. Kronrod suggested adding $n+1$ points to an already given n–point rule, which gives a rule with degree of exactness $3n+1$ (if n is even) or $3n+2$ (if n is odd). This method is optimal in the sense that this is the maximum degree of exactness obtainable by a rule constructed in this way. Patterson, using the 3–point Gauss rule as the first rule of his sequence, systematically added the required Kronrod points to obtain a rule of size $n = 2^{\ell+1}-1$ with degree of exactness $3\cdot 2^{\ell+1}-1$. Clearly these rules will also constitute a nested set.

3. Multivariate Integration Rules. We consider a sequence of approximations to (1.1) with an increasing number of nodes. Namely, for $\ell \in \mathbb{N}$, we consider the d–dimensional integration formula

$$Q(\ell,d)f = \sum_{i=1}^{n(\ell,d)} w_{\ell_i} f(x_{\ell_i}),$$

where $n(\ell,d)$, the number of integration points or abscissae, satisfies $n(\ell,d) < n(\ell+1,d)$. Here the abscissae, $X(l,d) = \{x_{\ell_i}, 1 \le \ell_i \le n(\ell,d)\}$ are elements of the hypercube $[-1,1]^d$. We will refer to the subscript ℓ as the *level* of the corresponding integration formula. When we refer to one–dimensional rules we will use the simpler notation Q_ℓ instead of $Q(\ell,1)$, and n_ℓ instead of $n(\ell,1)$. By X_ℓ we mean the abscissae of the level ℓ one–dimensional rule.

We construct d–dimensional integration rules from the tensor product of one–dimensional formulae. Such a construction is useful since it is easily generalized to any dimension and can use univariate rules of any size. We define the tensor–product of univariate integration rules by

$$(Q_{\ell_1} \otimes \cdots \otimes Q_{\ell_d})f := \sum_{i_1=1}^{n_{\ell_1}} \cdots \sum_{i_d=1}^{n_{\ell_d}} w_{\ell_1 i_1} \cdot ... \cdot w_{\ell_d i_d} \cdot f(x_{\ell_1 i_1}, ..., x_{\ell_d i_d}),$$

and introduce two examples of such a construction.

3.1. Simple–Product Formula. The most common rule is the simple tensor product of identical one–dimensional integration rules:

$$T(\ell, d)f := (Q_\ell \otimes ... \otimes Q_\ell)f \tag{3.1}$$

The greatest disadvantage of using this rule is that the number of abscissae needed to keep a given degree of exactness increases exponentially with the dimension of the integration domain. In particular, if the one–dimensional rule Q_ℓ contains n_ℓ points, the multi–dimensional $T(\ell, d)$ rule will require $(n_\ell)^d$ points. This is often called the "curse of dimensionality" and can greatly restrict the usefulness of the simple product rule when either d or n_ℓ are large.

3.2. Sparse–Grid Formula. The sparse–grid tensor–product rule was proposed as an alternative rule to reduce the effects of the curse of dimensionality, and is widely attributed to Smolyak [9]. The sparse–grid rule is constructed via a summation of tensor products of lower–order univariate rules. In particular, each of these tensor products need not be isotropic with respect to the individual dimensions. This has as an implication that we can create individual rules that are very accurate in desired directions but not in others. Therefore, we construct a "sparse" structure that can have high accuracy with a small set of abscissae. Here we review Smolyak's construction.

Given a sequence of univariate rules $Q_1, Q_2, ...$, we define the difference of such rules as

$$\Delta_\ell f := (Q_\ell - Q_{\ell-1})f.$$

Notice that these difference rules are univariate rules themselves on the set $X_\ell \cup X_{\ell-1}$. The sparse grid rule is then defined as

$$Q(\ell, d)f := \sum_{|i| \le \ell+d-1} (\Delta_{i_1} \otimes ... \otimes \Delta_{i_d})f \tag{3.2}$$

where i is the multi-index $i = (i_1, ..., i_d)$ and $|i| = i_1 + ... + i_d$. This can be expressed within a combinatorial formula [11], in terms of the original univariate formulae:

$$Q(\ell, d)f := \sum_{\ell \le |i| \le \ell+d-1} (-1)^{\ell+d-1-|i|} \begin{pmatrix} d-1 \\ \ell - |i| \end{pmatrix} (Q_{i_1} \otimes ... \otimes Q_{i_d})f. \tag{3.3}$$

In the multivariate case, we will continue to refer to ℓ as the *level* of the given rule.

3.3. Number of Integration Points. From (3.3) it follows that the set of points for the d–dimensional sparse grid rule will be

$$X(\ell, d) := \bigcup_{\ell \le |i| \le \ell+d-1} X_{i_1} \times ... \times X_{i_d}.$$

We say that univariate rules are nested if $X_\ell \subset X_{\ell+1}$ for all ℓ. Similarly, we say that multivariate rules are nested if $X(\ell, d) \subset X(\ell + 1, d)$ for all ℓ. If the chosen one–dimensional rules are nested, we get that

$$X(\ell, d) = \bigcup_{|i|=\ell+d-1} X_{i_1} \times ... \times X_{i_d},$$

i.e. the resulting multivariate sparse grid rules are nested as well.

In the nested case,

$$X(\ell, d + 1) := \bigcup_{s=1}^{\ell} X(\ell + 1 - s, d) \times (X_s \backslash X_{s-1}),$$

from which we can deduce the recursive formula for $n(\ell, d)$,

$$n(\ell, d + 1) = \sum_{s=1}^{\ell} n(\ell + 1 - s, d) \cdot (n_s - n_{s-1}). \tag{3.4}$$

If a univariate integration rule is chosen so that it is entirely non–nested (i.e. $X_i \cap X_j = \emptyset$ for $i \neq j$), then

$$\widetilde{n}(\ell, d) = \sum_{\ell \leq |i| \leq \ell+d-1} n_{i_1} \cdot ... \cdot n_{i_d} \tag{3.5}$$

denotes, in general, an upper bound for the number of integration points. If a rule is neither nested nor entirely non–nested, there is, in general, no explicit formula for the number of integration points. Nonetheless, this number can usually be computed without much difficulty.

4. Polynomial Exactness. Let $\mathbb{P}_k = \mathbb{P}_k^1$ be the space of one–dimensional polynomials of degree k or less. Define m_ℓ as the highest degree for which the univariate rule Q_ℓ is exact. The simple–product rule $T(\ell, d)$ is exact on the tensor–product space

$$\mathbb{P}_{m_\ell} \otimes ... \otimes \mathbb{P}_{m_\ell}, \tag{4.1}$$

which is a superset of the space $\mathbb{P}^d_{m_\ell}$, however, the degree of exactness of $T(\ell, d)$, as defined in the introduction, is limited to m_ℓ.

For the sparse grid, we will assume that our choice of the sequence of univariate integration rules satisfies

$$m_1 = 1,\ m_2 \geq 3,\ m_{i+1} - m_i \geq m_i - m_{i-1}. \tag{4.2}$$

The integration rule $Q(\ell, d)$ as defined by (3.2) is exact on the "non–classical" polynomial space

$$\sum_{|i|=\ell+d-1} \mathbb{P}_{m_{i_1}} \otimes ... \otimes \mathbb{P}_{m_{i_d}}. \tag{4.3}$$

Note the relation between this space and the rule defined by (3.3). Theorem 1, see [7], is useful in helping us determine the largest complete multivariate polynomial space integrated exactly by sparse–grid rules.

Theorem 1. *Assuming* (4.2) *with* $m_0 = -1$*, define* $\delta(\ell, d)$ *by*

$$\delta(\ell, d) = (m_{\sigma-1} + 1)(d - (\tau + 1)) + (m_\sigma + 1)(\tau + 1) - 1 \tag{4.4}$$

where

$$\ell + d - 1 = \sigma d + \tau \tag{4.5}$$

for some $\sigma \in \mathbb{N}$ *and* $\tau \in \{0, ..., d - 1\}$*. Then* $Q(\ell, d)$ *has degree* $\delta(\ell, d)$ *of exactness.*

4.1. Degree of Exactness for Common Sparse–Grid Constructions. A sparse–grid rule often considered in the literature is that based on the univariate Clenshaw–Curtis rule [6, 4]. We denote it by $Q_{\mathrm{CC}}(\ell, d)$ for the d–variate sparse grid. Since nested rules grow more slowly than non–nested rules, we ensure that this rule is nested and still growing as slowly as possible, by taking

$$n_1 = 1 \qquad \text{and} \qquad n_\ell = 2^{\ell-1} + 1$$

for the levels $\ell > 1$, and since the number of one–dimensional points for any given level is odd, we get

$$m_\ell = n_\ell.$$

From Theorem 1 we deduce that the degree of exactness of the $Q_{\mathrm{CC}}(\ell, d)$ rule is given by

$$\delta(\ell, d) = \begin{cases} 2(\ell - 1) + 1, & \text{for } \ell + d - 1 < 4d \\ 2^{\sigma-2}(d + \tau + 1) + 2d - 1, & \text{otherwise.} \end{cases}$$

When using one–dimensional Gauss rules to construct multivariate rules it is impossible to exploit nestedness, since $\mathbf{0} \in \mathbb{R}^d$ is the only nested abscissa, and only in the case of rules of odd order. This implies that there could be comparatively large growth as both d and ℓ increase. The second rule considered here is constructed from entirely non–nested one–dimensional Gauss rules, with

$$n_i = 2^{i-1}, \quad \text{and} \quad m_i = 2^i - 1.$$

We denote it by $Q_{\mathrm{Gfast}}(\ell, d)$. Its degree of exactness satisfies

$$\delta(\ell, d) = \begin{cases} 2(\ell - 1) + 1, & \text{for } \ell + d - 1 < 3d \\ 2^{\sigma-1}(d + \tau + 1) + d - 1, & \text{otherwise.} \end{cases}$$

The third rule also uses one–dimensional Gauss points, but, instead of allowing exponential growth in the size of the univariate rule, we limit it to constant growth. In this case we fix

$$n_i = 2i - 1, \quad \text{and} \quad m_i = 4i - 3.$$

Although fast–growing rules have advantages with respect to the number of integration points and accuracy for polynomials of very high degree and dimension, slow growth can be beneficial for lower polynomial degrees and dimensions. Examples of this are given in Section 6. We denote this rule by $Q_{\mathrm{Gslow}}(\ell, d)$. Its degree of exactness is given by

$$\delta(\ell, d) = \begin{cases} 2(l - 1) + 1, & \text{for } \ell + d - 1 < 2d \\ 4\sigma d - 6d + 4\tau + 3, & \text{otherwise.} \end{cases}$$

The fourth rule studied in this paper uses the Patterson univariate rule. This sparse grid construction is considered in [4]. We will use $Q_{\mathrm{GP}}(\ell, d)$ to denote the multivariate rule thus constructed. Here we fix

$$n_\ell = 2^\ell - 1,$$

as described above, and arrive at another nested multivariate rule. As

$$m_l = 3 \cdot 2^\ell - 2,$$

we obtain from Theorem 1 that the degree of exactness of the $Q_{\mathrm{GP}}(\ell, d)$ rule is

$$\delta(\ell, d) = \begin{cases} 2(\ell - 1) + 1, & \text{for } \ell + d - 1 < 2d \\ 3 \cdot 2^{\sigma-2}(d + \tau + 1) - d - 1, & \text{otherwise.} \end{cases}$$

5. Adaptive Sparse Grid Rules. Many applications, such as the finite element solution of PDEs, require accurate numerical integration of multivariate polynomials. Since integration rules often need to be applied repeatedly, decreasing the number of integration points is an important goal. Sparse grid rules have good convergence properties with increasing polynomial degree and dimension, but can be wasteful for the task of integrating a complete polynomial space of lower degree in, e.g. three dimensions. In this section we relax the definition of a sparse grid to include terms that contribute to the exact integration of a complete polynomial space and exclude terms that do not, with the goal of decreasing the total number of integration points.

5.1. Definition. The adaptive sparse grid [4] is defined as

$$Q_{\mathcal{I}} f := \sum_{i \in \mathcal{I}} (\Delta_{i_1} \otimes ... \otimes \Delta_{i_d}) f, \tag{5.1}$$

where $\mathcal{I}$ is an admissible index set in $\mathbb{N}^d$. We say that $\mathcal{I}$ is *admissible* if

$$i - e_j \in \mathcal{I}, \quad \text{for all } j \in \{1, ..., d\}, \;\; i_j > 0,$$

whenever $i \in \mathcal{I}$. Here e_j is the jth unit vector. Notice that the usual sparse grid satisfies

$$\mathcal{I} = \{i \in \mathbb{N}^d, \, |i| \leq \ell + d - 1\}$$

for the appropriate level and dimension.

In [5] this idea is used to find the most appropriate rule to integrate a chosen integrand. Our focus is on the integration of complete polynomial spaces.

5.2. A 2D Explanation. To illustrate how this adaption can be used in our case, we choose dimension $d = 2$. The set of all monomials spanning $\mathbb{P}_3^2$ is shown in the diagram below, where each row contains all monomials of total degree given in the left–hand column.

0					1				
1				x		y			
2			x^2		xy		y^2		
3		x^3		x^2y		xy^2		y^3	

The simple–product rule and sparse–grid rules integrate out complete polynomials, here viewed as linear combinations of monomials, in different manners. Examples are shown in Table 5.1 for the $T(\ell, d)$ rule and in Table 5.2 for the $Q_{\text{Gslow}}(\ell, d)$ rule. We showcase the minimal $T(\ell, d)$ and $Q_{\text{Gslow}}(\ell, d)$ rules that exactly integrate complete polynomial spaces $\mathbb{P}_3^2$ and $\mathbb{P}_5^2$, respectively. The monomials marked below the horizontal lines are integrated unnecessarily, and thus represent wasted effort. From that perspective, the $Q_{\text{Gslow}}(\ell, d)$ rule is particularly "wasteful", because it not only integrates the desired polynomial space $\mathbb{P}_5^2$ exactly, but it unintentionally recovers 31 additional monomials, including the full spaces $\mathbb{P}_6^2$ and $\mathbb{P}_7^2$.

The increase in the level of a sparse–grid rule corresponds to the addition of new higher–degree tensor–product terms, as shown by (3.3). For instance, the new monomials to the left and to the right of the polynomial pyramid, shown in frame two of Table 5.2, are integrated exactly by the terms $Q_3 \otimes Q_1$ and $Q_1 \otimes Q_3$, respectively, while the new terms in the middle are integrated by $Q_2 \otimes Q_2$. If we remove the terms $Q_3 \otimes Q_1$ and $Q_1 \otimes Q_3$ from the rule, then we have a rule that is less expensive and still integrates the polynomial space $\mathbb{P}_5^2$ exactly, as shown in Table 5.2. This corresponds to using the index set

$$\mathcal{I} = \{|i| \leq 3\} \cup \{(2, 2)\}.$$

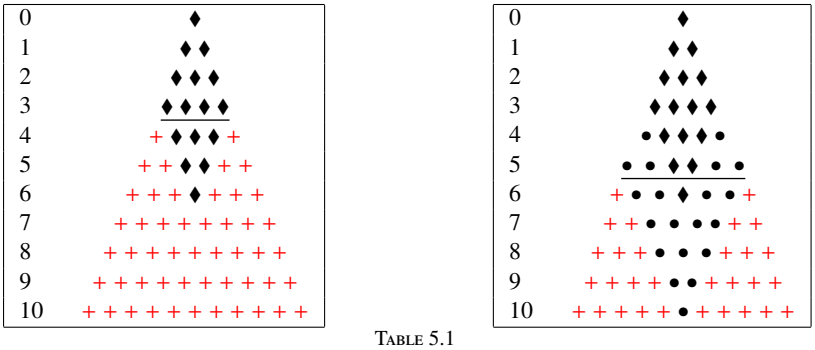


TABLE 5.1

Exactly integrated monomials for the simple–product rule. Left: $T(3,2)$, the minimal rule that integrates $\mathbb{P}_3^2$ exactly. Right: $T(5,2)$, the minimal rule that integrates $\mathbb{P}_5^2$ exactly.

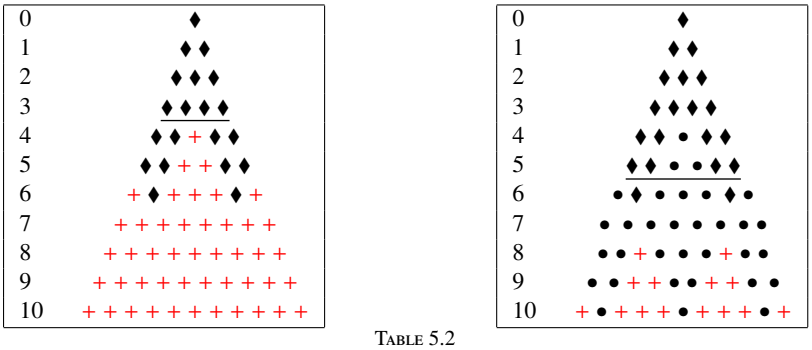


TABLE 5.2

Exactly integrated monomials for the slow–Gauss rule. Left: $Q_{GS\,low}(2,2)$, the minimal rule that integrates $\mathbb{P}_3^2$ exactly. Right: $Q_{GS\,low}(3,2)$, the minimal rule that integrates $\mathbb{P}_5^2$ exactly.

Continuing this process, to integrate $\mathbb{P}_7^2$ exactly, we would use the rule $Q_{GS\,low}(3,2)$. To integrate $\mathbb{P}_9^2$, we would use the index set

$$\mathcal{I} = \{|i| \leq 4\} \cup \{(3,2),(2,3)\},$$

etc. We denote adaptive sparse–grid rules thus defined by Q_{ADPV}.

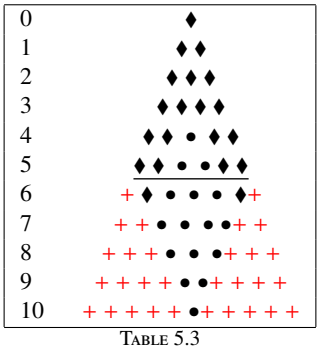


TABLE 5.3

Exactly integrated monomials for the adaptive sparse–grid construction.

5.3. Adaptive Sparse–Grid Construction Formalized. Assume that a set of one–dimensional rules Q_ℓ and the polynomial space $\mathbb{P}_m^d$ have been chosen. The goal is to find the most efficient adaptive sparse–grid rule Q_{ADPV} that integrates the polynomial space exactly. The algorithm follows.

Algorithm 5.1 (Exact Adaptive Sparse–Grid Integration of $\mathbb{P}_m^d$).

1. Loop over every d–dimensional monomial $p(x)$ of total degree m or less. If $(i_1, ..., i_d)$ is the vector of exponents of $p(x)$, then for i_j pick the smallest level l_j such that Q_{l_j} integrates x^{i_j} exactly.
2. Include $(l_1, ..., l_s)$ in a pre–index set $\mathcal{J}$. Once completed, the set $\mathcal{J}$ will contain the largest indices required to construct the desired sparse grid.
3. Define index set
$$\mathcal{I} = \{i \in \mathbb{N}^d, i \leq j, j \in \mathcal{J}\}.$$
4. Construct sparse grid using either (5.1), or an analogous form of (3.3)
$$Q_{\mathcal{I}} f := \sum_{i \in \mathcal{I}} \left(\sum_{z_1=1}^{1} \cdots \sum_{z_d=0}^{1} (-1)^{|z|} \chi_{\mathcal{I}}(i+z) \right) (Q_{i_1} \otimes ... \otimes Q_{i_d}) f,$$
where $\chi_{\mathcal{I}}$ is the characteristic function of $\mathcal{I}$,
$$\chi_{\mathcal{I}}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in \mathcal{I}, \\ 0 & \text{otherwise.} \end{cases}$$

6. Numerical Results. There are two important aspects of the use of numerical integration in finite element simulation. The first is the integration of single integrands, such as user–provided or right–hand side functionals, which are not always related to the choice of the polynomial basis for the finite element space. The second, and likely more important aspect is the computation of discrete differential operators, in which case the cost of numerical integration is closely tied to the choice of the finite element basis (in particular, to its degree or "order"). We investigate the performance of classical sparse–grid rules, described in Section 4.1, as well as that of the proposed adaptive constructions in both scenarios.

6.1. Exact Integration in $\mathbb{P}_m^d$ Spaces. In Table 6.1 we compare the computational cost of classical sparse grid rules, described in Section 4.1, to that of the simple–product rule, described in Section 3.1, in the context of integration of complete polynomial spaces in three dimensions. It is evident that the only rule that competes with the simple–product rule is the slow–Gauss rule. In two dimensions the greater efficiency of the simple–product rule is even clearer. We omit this comparison.

Table 6.2 compares the simple–product rule to the adaptive sparse–grid construction based on the univariate slow–Gauss rule. For polynomials degree 17 or lower, the adaptive rule outperforms the simple–product rule. Its advantages are particularly pronounced for polynomial degrees 5, 7, and 9. For completeness, we include the data for the symmetric rules provided in [3], denoted by *Dun*, and the corresponding *theoretical* minimum (generally not achieved in practice), denoted by $MinS$, also see [3]. Although the *Dun* rules clearly use fewer integration points than the adaptive sparse grid, they are difficult to construct for arbitrarily high polynomial degrees and may be difficult to incorporate in finite element codes since the points often lie outside of the reference domain $[-1, 1]^d$.

Although general sparse grids already outperform the simple–product rule for dimensions larger than 3, it should be noted that the adaptive construction can give even better performance when integrating complete polynomial spaces. Furthermore, adaptive constructions can be used for any of the sparse grids discussed in Section 4.1. Tables 6.3 and 6.4 show

Degree	$T(\ell,3)$		$Q_{\mathrm{CC}}(\ell,3)$		$Q_{\mathrm{GFast}}(\ell,3)$		$Q_{\mathrm{GSlow}}(\ell,3)$		$Q_{\mathrm{GP}}(\ell,3)$	
m	ℓ	n_ℓ	ℓ	n_ℓ	ℓ	n_ℓ	ℓ	n_ℓ	ℓ	n_ℓ
3	2	8	2	7	2	7	2	7	2	7
5	3	27	3	25	3	31	3	31	3	31
7	4	64	4	69	4	110	4	105	4	111
9	5	125	4	69	5	344	4	105	4	111
11	6	216	6	441	6	992	5	297	5	351
13	7	343	7	441	7	2704	5	297	6	1023
15	8	512	8	2561	8	7072	6	735	7	2815
17	9	729	9	6017	8	7072	6	735	7	2815
19	10	1000	10	13953	8	7072	7	1631	7	2815

TABLE 6.1

This table lists the minimum level, denoted by ℓ, and the number of integration points, denoted by n_ℓ, required by each rule of degree of exactness m, in three dimensions.

Degree	$T(\ell,3)$	Q_{ADPV}	*Dun*	*MinS*
3	8	7	6	6
5	27	19	14	14
7	64	39	27	27
9	125	87	53	52
11	216	153	89	77
13	343	273	151	127
15	512	465	235	175
17	729	705	307	253
19	1000	1183	435	333

TABLE 6.2

Comparison of the number of integration points required by the simple–product, adaptive slow–Gauss, and Dunavant rules, as well as the theoretical minimum (for a symmetric rule), to exactly integrate a complete polynomial space of a given degree.

a performance comparison between the adaptive algorithm applied to the slow–Gauss rule, the classical slow–Gauss rule, and the simple–product rule in four and five dimensions.

6.2. Finite Element Solution of a PDE. In this section we investigate the numerical solution of a model PDE, obtained using a high–order finite element discretization combined with the simple–product and adaptive slow–Gauss integration rules. We solve the Poisson equation

$$-\Delta u(x) = f(x) \qquad x \text{ in } \Omega, \tag{6.1}$$

$$u(x) = d(x) \qquad x \text{ on } \partial\Omega, \tag{6.2}$$

where $\Omega = (0,1)^3$, and the right–hand side f and Dirichlet boundary conditions d are manufactured so that the analytic solution u_0 satisfies

$$u_0(x) = \sin(\pi x_1)\sin(\pi x_2)\sin(\pi x_3)e^{x_1+x_2+x_3},$$

for $(x_1, x_2, x_3) \in (\Omega \cup \partial\Omega)$. The Poisson equation is discretized using *3–rectangles of type (6)*, where we refer to the notation of Ciarlet, [1, p.59ff], i.e. a 6th order Lagrange tensor–product finite element basis with 343 degrees of freedom per reference element.

We study the accuracy of finite element approximations to u_0 on a sequence of four uniform partitions, 3×3×3 through 6×6×6, of the domain Ω. In addition, we check whether

Degree	$T(\ell,4)$	Q_{GSlow}	Q_{ADPV}
3	16	9	9
5	81	49	33
7	256	201	81
9	625	681	193
11	1296	681	409
13	2401	2001	777
15	4096	2001	1482
17	6561	5257	2537
19	10000	5257	4369

TABLE 6.3

Comparison of the number of integration points required by the simple–product, classical slow–Gauss, and adaptive slow–Gauss rules in four dimensions, to exactly integrate a complete polynomial space of a given degree.

Degree	$T(\ell,5)$	Q_{GSlow}	Q_{ADPV}
3	32	11	11
5	243	71	51
7	1024	341	151
9	3125	1341	391
11	7776	4543	933
13	16807	4543	1973
15	32768	13683	4013
17	59049	13683	7693
19	100000	37433	13983

TABLE 6.4

Comparison of the number of integration points required by the simple–product, classical slow–Gauss, and adaptive slow–Gauss rules in five dimensions, to exactly integrate a complete polynomial space of a given degree.

the chosen discretization combined with the simple–product and adaptive slow–Gauss rules passes the so–called *patch test*, i.e. whether a PDE solution belonging to the space $\mathbb{P}_6^3$ can be recovered to near machine precision. For the patch test we use the linear combination of all multivariate monomials in $\mathbb{P}_6^3$ with unit weights as the reference solution.

$T(\ell,3)$ Degree	$\|u^h-u_0\|_{L^2}$ order	$\|u^h-u_0\|_{H^1}$ order	$\|u^h-u_0\|_{L^2}$ on 6×6×6	patch test (L^2 error)
6	—	—	7.0e+10	Fail (> 1)
8	—	—	4.5e+01	Fail (> 1)
10	6.9998	6.0098	9.3e−10	Pass (1.9e−13)
12	6.9998	6.0098	9.3e−10	Pass (2.6e−13)

TABLE 6.5

L^2 and H^1 orders of convergence, L^2 error on the finest grid, and patch test results for the simple–product rule.

The results are presented in Tables 6.5 and 6.6. We make several observations. First, for a sufficiently high degree of exactness, theoretical orders of convergence, namely 7 in the L^2 norm and 6 in the H^1 norm, are obtained for both the simple–product and the adaptive slow–Gauss formula. Second, it is evident that the adaptive sparse–grid rule is inferior to the simple–product rule. This can be explained by the fact that the adaptive sparse–grid rule is designed for exact integration of complete polynomial spaces — in contrast, the differential

Q_{ADPV} Degree	$\|u^h - u_0\|_{L^2}$ order	$\|u^h - u_0\|_{H^1}$ order	$\|u^h - u_0\|_{L^2}$ on 6×6×6	patch test (L^2 error)
16	—	—	4.1e+00	Fail (> 1)
18	5.0298	3.9884	2.0e−06	Fail (1.0e−03)
20	7.6813	5.8886	1.5e−06	Fail (5.5e−04)
22	7.0091	6.0151	9.3e−10	Pass (5.7e−13)
24	7.0091	6.0151	9.3e−10	Pass (4.5e−13)

TABLE 6.6

L^2 and H^1 orders of convergence, L^2 error on the finest grid, and patch test results for the adaptive slow–Gauss rule.

operators used herein for the solution of the PDE are computed using products of gradients of tensor–product polynomials in $\mathbb{P}^1_6 \times \mathbb{P}^1_6 \times \mathbb{P}^1_6$. Such products belong to a polynomial space much larger than $\mathbb{P}^3_{10}$, the space that contains products of gradients of polynomials in $\mathbb{P}^3_6$. On the other hand, the simple–product rule is a full tensor–product rule, and thus a natural, perhaps optimal choice for integration in a tensor–product finite element space. This and other higher–order examples support the conjecture that an mth–order tensor–product finite element discretization passes the patch test if combined with a simple–product rule of degree of exactness $2m - 2$, whereas an adaptive slow–Gauss rule of degree of exactness $4m - 2$ is otherwise necessary. Finally, we note that in either case, success of the patch test always indicates optimal order of convergence of the finite element approximation. At the same time, for the simple–product rule, failure of the patch test always implies complete loss of convergence of the finite element approximation. In contrast, failure of the patch test in the case of the adaptive slow–Gauss rule is often related to merely partial loss of convergence.

7. Conclusion. We have investigated the use of sparse grids, typically used in stochastic sampling, for the numerical integration in "physical" space, i.e. in two and three dimensions. In addition to studying classical sparse–grid constructions, we have introduced an adaptive algorithm for the construction of numerical integration rules on the domain $[-1, 1]^d$ of arbitrary dimension d, with the particular focus on exact integration of complete polynomial spaces of a given degree. In contrast to the classical sparse–grid rules, our algorithm outperforms the simple tensor–product rule in this regard, for the spaces $\mathbb{P}^3_3$ through $\mathbb{P}^3_{17}$. Integration formulae exist that perform even better than the proposed adaptive rule, see [3], however, they are difficult to extend to arbitrary polynomial degree and dimension.

Additionally, we have studied the application of the proposed adaptive sparse–grid rules in computing high–order tensor–product finite element PDE approximations using hexahedral elements. Due to the tensor–product nature of finite element spaces used on such elements, the adaptive rule cannot match the performance of the simple tensor–product rule. We offer a conjecture on the degree of exactness necessary by either rule to pass the so–called patch test and recover the optimal order of convergence of the finite element approximation. It remains to be seen whether the simple tensor–product rule is in fact optimal in this regard.

REFERENCES

[1] P. G. CIARLET, *The Finite Element Method for Elliptic Problems*, SIAM, Philadelphia, 2002.

[2] C. W. CLENSHAW AND A. R. CURTIS, *A method for numerical integration on an automatic computer*, Numer. Math., 2 (1960), pp. 197–205.

[3] D. DUNAVANT, *Efficient symmetrical cubature rules for complete polynomials of high degree over the unit cube*, Int. J. Num. Meth. Engng., 23 (1986), pp. 397–407.

[4] T. Gerstner and M. Griebel, *Numerical integration using sparse grids*, Numer. Algorithms, 18 (1998), pp. 209–232.
[5] ———, *Dimension-adaptive tensor-product quadrature*, Computing, 71 (2003), pp. 65–87.
[6] E. Novak and K. Ritter, *High-dimensional integration of smooth functions over cubes*, Numer. Math., 75 (1996), pp. 79–97.
[7] ———, *Simple cubature formulas with high polynomial exactness*, Constr. Approx., 15 (1999), pp. 499–522.
[8] T. N. L. Patterson, *Table errata: "The optimum addition of points to quadrature formulae" (Math. Comp.* **22** *(1968), 847–856; addendum, ibid.* **22** *(1968), no. 104, loose microfiche suppl. C1-C11)*, Math. Comp., 23 (1969), p. 892.
[9] S. A. Smolyak, *Quadrature and interpolation formulas for tensor products of certain classes of functions*, Dokl. Akad. Nauk SSSR, 148 (1963), pp. 1042–1043. Russian, Engl. Transl.: Soviet Math. Dokl. 4:240–243, 1963.
[10] L. N. Trefethen, *Is Gauss quadrature better than Clenshaw-Curtis?*, SIAM Rev., 50 (2008), pp. 67–87.
[11] G. W. Wasilkowski and H. Woźniakowski, *Explicit cost bounds of algorithms for multivariate tensor product problems*, J. Complexity, 11 (1995), pp. 1–56.

OVERVIEW AND PERFORMANCE ANALYSIS OF THE EPETRA/OSKI MATRIX CLASS INTERFACE IN TRILINOS

IAN KARLIN * AND JONATHAN HU †

Abstract. In this paper, we describe a new matrix class in Epetra that gives a Trilinos application access to the Optimized Sparse Kernel Interface (OSKI) package. Epetra is the core basic linear algebra package within Trilinos, Sandia's numerical algorithms framework. We give an overview of OSKI and the new Epetra class design. We also present numerical results that compare performance of equivalent OSKI and Epetra kernels in serial and in parallel. Finally, we discuss potential impact of OSKI on applications that currently use Trilinos.

1. Introduction. Many real world scientific problems, in fields such as atmospheric science, quantum physics, and structural engineering, are simulated on computers. Due to model complexity, fidelity, or time scales, such simulations often must be run on massively parallel computers. The time and effort involved in designing these simulations is large. Therefore, many simulations leverage existing optimized kernels and algorithms provided by other software libraries. At Sandia, one such source of state-of-the-art numerical algorithms is the Trilinos project [7].

Trilinos is a collection of scientific computing libraries called "packages". Each package in Trilinos has unique functionality, and is written by domain experts. Packages are typically autonomous, but can leverage capabilities in other Trilinos packages. Functionality available within Trilinos packages includes basic linear algebra operations, preconditioning, solvers, data distribution and load balancing. The Trilinos project provides application developers a suite of modern optimized numerical methods. In turn, Trilinos leverages basic libraries such as the BLAS [4] and LAPACK [1].

Epetra, a foundational package within Trilinos, is frequently used by other packages [9]. Epetra provides fundamental classes and methods for serial and parallel linear algebra. Classes available include point and block matrices, multivectors, and graphs. These and other classes support the usual linear algebra operations. All solver packages within Trilinos can use Epetra kernels as building blocks for both serial and parallel algorithms. For this reason, the performance of solvers depends upon Epetra's performance. Therefore, making Epetra as efficient as possible will improve the performance and efficiency of other packages that depend on it.

Just as a program is only as efficient as its underlying components, a parallel program can only be as efficient as the code run on each processor. Even if a program scales efficiently, if its underlying serial code is inefficient, its parallel implementation will be inefficient. By improving the performance of the single-processor portion of a parallel program, the potential top speed of a parallel program is improved. For example, in many scientific programs an important kernel operation is matrix-vector multiplication. By speeding up this kernel, overall simulation speed can be improved.

The Optimized Sparse Kernel Interface (OSKI) provides many highly tuned matrix vector multiply kernels [13, 2, 14]. OSKI provides five optimized, serial, sparse matrix-vector kernels: four routines that perform matrix-vector multiplication and one that performs a triangular solve of a system of equations. At install time, OSKI's kernels can be tuned according to the underlying machines architecture. At runtime, OSKI's kernels can be tuned according to matrix/vector structure. The new Epetra/OSKI interface enables Trilinos and application developers to leverage the highly tuned kernels provided by OSKI in a standardized manner.

*University of Colorado, Boulder, Ian.Karlin@colorado.edu
†Sandia National Laboratories, jhu@sandia.gov

In this paper, we discuss our implementation of an interface to OSKI within Epetra and assess its performance. In Section 2, we give an overview of the design and features of the OSKI package itself. In Section 3, we discuss the design of the Epetra interface to OSKI. In Section 4, we discuss the results of performance tests run on the OSKI kernels within Epetra. Tests were run on individual OSKI kernels, and include small scaling studies. In Section 5, conclusions of the work and results described in this paper are presented. In Section 6, ways to add more functionality to our implementation, and suggestions of things to test in new OSKI releases are presented.

2. OSKI High Level Overview. OSKI is a package used to perform optimized sparse matrix-vector operations. It provides both a statically tuned library created upon installation and dynamically tuned routines created at runtime. OSKI provides support for single and double precision values of both real and complex types, along with indexing using both integer and long types. When possible it follows the sparse BLAS standard [5] as closely as possible in defining operations and functions.

Before a matrix can use OSKI functionality, it first must be converted to the matrix type `oski_matrix_t`. To store a matrix as an `oski_matrix_t` object, a create function must be called on a CSR or CSC matrix. An `oski_matrix_t` object can either be created using a deep or shallow copy of the matrix. When a shallow copy is created, the user must only make changes to the matrix's structure through the OSKI interface. When a deep copy is created, the matrix that was passed in can be edited by the user as desired. OSKI automatically makes a deep copy when any matrix is tuned in a manner that changes its structure.

Routine	Calculation
Matrix-Vector Multiply	$y = \alpha Ax + \beta y$ or $y = \alpha A^T x + \beta y$
Triangular Solve	$x = \alpha A^{-1} x$ or $x = \alpha A^{T^{-1}} x$
Matrix Transpose Matrix-Vector Multiply	$y = \alpha A^T Ax + \beta y$ or $y = \alpha AA^T x + \beta y$
Matrix Power Vector Multiply	$y = \alpha A^p x + \beta y$ or $y = \alpha A^{T^p} x + \beta y$
Matrix-Vector Multiply and Matrix Transpose Vector Multiply	$y = \alpha Ax + \beta y$ and $z = \omega Aw + \zeta z$ or $z = \omega A^T w + \zeta z$

TABLE 2.1
Computational kernels from OSKI available in Epetra.

OSKI provides five matrix-vector operations to the user. The operations are shown in Table 2.1. Hermitian operations are available in OSKI, but are not shown in the table since Epetra does not include Hermitian functionality. The last three kernels are composed operations using loop fusion [6] to increase data reuse. To further improve performance, OSKI can link to a highly tuned BLAS library.

OSKI creates optimized routines for the target machine's hardware based on empirical search, in the same manner as ATLAS [15] and PHiPAC [3]. The goal of the search is create efficient static kernels to perform the operations listed in Table 2.1. The static kernels then become the defaults that are called by OSKI when runtime tuning is not used. Static tuning can create efficient kernels for a given data structure. To use the most efficient kernel, the matrix data structure may need to be reorganized.

When an operation is called enough times to amortize the cost of rearranging the data structure, runtime tuning can be more profitable than using statically tuned functions. OSKI provides multiple ways to invoke runtime tuning, along with multiple levels of tuning. A user can explicitly ask for a matrix to always be tuned for a specific kernel by selecting either the moderate or aggressive tuning option. If the user wishes for OSKI to decide whether enough calls to a function occur to justify tuning, hints can be used. Possible hints include telling OSKI the number of calls expected to the routine and information about the matrix, such as block structure or symmetry. In either case, OSKI tunes the matrix either according to the user's requested tuning level, or whether it expects to be able to amortize the cost of tuning if hints are provided. Instead of providing hints the user may, periodically call the tune function. In this case, the tune function predicts the number of future kernel calls based on past history, and tunes the routine only if it expects the tuning cost to be recovered via future routine calls.

OSKI can also save tuning transformations for later reuse. Thus, the cost of tuning searches can be amortized over future runs. Specifically, a search for the best tuning options does not need to be run again, and only the prescribed transformations need to be applied.

OSKI is under active development. As of this writing, the current version is 1.0.1h, with a multi-core version under development [12]. While OSKI provides many optimized sparse matrix kernels, some features have yet to be implemented, and certain optimizations are missing. OSKI is lacking multi-vector kernels and stock versions of the composed kernels. These would greatly add to both OSKI's usability and performance. The Matrix Power Vector Multiply is not functional. Finally, OSKI cannot transform (nearly) symmetric matrices to reduce storage or convert from a CSR to a CSC matrix (or vice versa). Both could provide significant memory savings. Thus, performance gains from runtime tuning should not be expected for point matrices. An exception is pseudo-random matrices, which may benefit from cache blocking.

3. Design and Implementation. In the design and implementation of the Epetra OSKI interface the Epetra coding guidelines [8] were followed as closely as possible. In doing so, we ensured the consistency of our code with the existing Epetra code base, as well as its readability and maintainability. Finally, the Epetra interface to OSKI will likely be ported to Kokkos [10], and the interface's design will make this process easier.

In the design phase we focused on allowing the greatest amount of flexibility to the user, and exposing as much of the functionality of OSKI as possible. In some places, however, OSKI functionality is not exposed because there is not a corresponding Epetra function. For example, OSKI has a function that allows changing a single value in a matrix, but Epetra does not. When two copies of a matrix exist, as when the OSKI constructor makes a deep copy of the underlying data, the corresponding Epetra copy is guaranteed to contain the same data. Since Epetra can only change data values one row at a time, a point set function is not included in the OSKI interface. Instead, we include a function to change a row of data within OSKI by overloading the Epetra function to change row data. When a single copy of the data exists, the Epetra function is called on the matrix. When both an OSKI and Epetra matrix exist, both the matrix copies are modified to keep the data consistent. The Epetra function is called once for the Epetra version of the matrix, and the OSKI matrix has its point function called once for each entry in the row.

When there are clear equivalent functions in OSKI and Epetra, the OSKI function is designed to overload the Epetra function. In the cases where OSKI provides more functionality than Epetra, the interface is designed with two functions to perform the operation. The first function mimics Epetra's functionality and passes values that eliminate the extra functionality from OSKI. The second function exposes the full functionality OSKI provides. Also, as appropriate new functions are added that are specific to OSKI, such as the tuning functions.

Conversely, Epetra functions without any analogue in the OSKI context are not overloaded in the `Epetra_Oski` namespace.

The interface promotes robustness and ease of use. All `Epetra_OskiMatrix` functions that take in vectors or multi-vectors allow for the input of both `Epetra_Vector` or `Epetra_MultiVector` objects, and `Epetra_OskiVector` or `Epetra_OskiMultiVector` objects. The objects are converted to the proper types as necessary through the use of the lightest weight wrapper or converter possible.

The implementation follows the idea of wrapping and converting data structures in as lightweight a fashion as possible, to maximize speed and minimize space used. In addition, the implementation provides the user with as much flexibility as possible. For example, the user can specify as many tuning hints as they like. Alternatively, the user can ask Epetra to figure out as much as it can about the matrix and pass along those hints to OSKI. Both options can be combined, with user-specified hints taking precedence over automatically generated hints. Options are passed by the user via Teuchos parameter lists [11].

Class	Function
`Epetra_OskiMatrix`	Derived from `Epetra_CrsMatrix`. Provides all OSKI matrix operations.
`Epetra_OskiMultiVector`	Derived from `Epetra_MultiVector`. Provides all OSKI multi-vector operations.
`Epetra_OskiVector`	Derived from `Epetra_OskiMultiVector`. Provides all OSKI vector operations.
`Epetra_OskiPermutation`	Stores permutations and provides Permutation functions not performed on a `Epetra_OskiMatrix`.
`Epetra_OskiError`	Provides access to OSKI error handling functions and the ability to change the default OSKI error handler.
`Epetra_OskiUtils`	Provides the initialize and finalize routines for OSKI.

TABLE 3.1
OSKI classes within Epetra.

Finally, the design is broken into six separate classes. Table 3.1 shows the classes and provides information about which classes each derives from, and what functions each contains. The design is as modular as possible to allow for the easy addition of new functions, and to logically group related functions together.

4. Results. To assess the potential benefit of using OSKI in Sandia applications, we ran tests on representative data and a variety of advanced architectures. For these tests OSKI version 1.0.1h was used. OSKI runtimes were compared to the runtimes of the currently used Epetra algorithms, in both serial and parallel. In this section, we first present our test environment and methodology, and then present the results of performance tests run comparing Epetra to OSKI.

4.1. Test Environment and Methodology. Performance tests were run on two different machine architectures in serial and parallel. The first test machine has two Intel Clovertown processors. The second test machine has one Sun Niagara-2 processor. Machine specifications and compilers are shown in Table 4.1. On each machine, Trilinos was compiled with widely used optimizations levels, and OSKI was allowed to pick the best optimization flags itself.

processor	#chips	cores	threads	frequency	L2 cache	compiler
Clovertown	2	8	8	1.87 Ghz	4 M per 2 cores	Intel
Niagara-2	1	8	64	1.4 Ghz	4 M per core	Sun

TABLE 4.1
Test machines used for performance testing.

These machines were chosen for their diversity and potential for use at Sandia. The Clovertown is one of Intel's latest processors, and the Niagara is an example of an extremely parallel chip.

On each machine, tests were run on three matrices arising from Sandia applications. The first matrix is from a finite element discretization within a magnetics simulation. The second is a block-structured Poisson matrix. The third matrix is unstructured and represents term-document connectivity. The data is from the Citeseer application. Table 4.2 gives some matrix properties. Each matrix was able to fit within the main memory of each test machine. These matrices were also used in a scaling study. Tests were run up to the total number of

matrix	rows	columns	nnz	structure
point	556356	556356	17185984	nearly symmetric point
block	174246	174246	13300445	symmetric 3 by 3 blocks
Citeseer	607159	716770	57260599	unstructured point

TABLE 4.2
Test machines for Epetra OSKI performance testing.

available threads that can be executed simultaneously, on each machine.

4.2. Performance Test Results. The serial results for each machine are shown in Figures 4.1 and 4.2 for four OSKI kernels: Ax, A^Tx, A^TAx, and the two-vector multiplication $y = Ax$; $z = Aw$. The last operation is henceforth referred to as "2Mult". In addition, Table 4.3 shows the speeds of Epetra calculations as a baseline. Since OSKI has no atomic versions of the composed kernels, the OSKI stock numbers represent two separate matrix-vector multiply calls to OSKI. There is potential that the tuned composed kernels are not performing optimally due to tuning to a non-ideal data structure, as is seen in the tuning cost data later. Results for the matrix power kernel are unavailable due to a bug in the kernel. Also results for the AA^T kernel were excluded because Epetra only stores matrices in CSR. OSKI cannot convert CSR to CSC, which is needed to take advantage of these kernels in serial. Finally, the direct solve kernel was not profiled, as it is not critical to many Sandia applications.

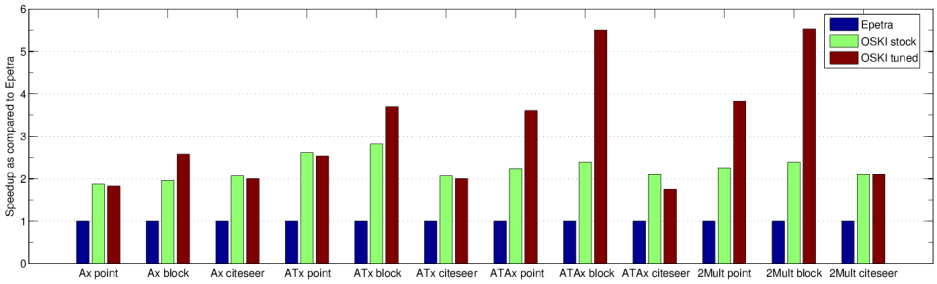


FIG. 4.1. *Relative performance of Epetra and OSKI in serial on Clovertown.*

Machine	Ax	A^Tx	A^TA	2Mult
Clovertown	220/227/55	150/154/43	178/183/48	178/184/48
Niagara	58.3/69.9/20.7	56/66.4/20.3	57.1/68.1/20.5	57.1/68.1/20.5

TABLE 4.3
Epetra serial routine speeds in Mflops. Results are in the form point/block/Citeseer.

On the Clovertown, OSKI produced large speedups over Epetra for all matrices in serial, as shown in Figure 4.1. The stock kernels demonstrated speedups of 1.8 to 2.8. Tuning improved the block matrices by about one third when compared to the stock kernels. The composed algorithms demonstrated even more significant speedups of up to 5.5, when composing and blocking were combined. Tuning did not improve the runtime of point matrices, except when a composed kernel was used. In the case of the Citeseer matrix, a composed kernel resulted in either no performance gain or performance degradation.

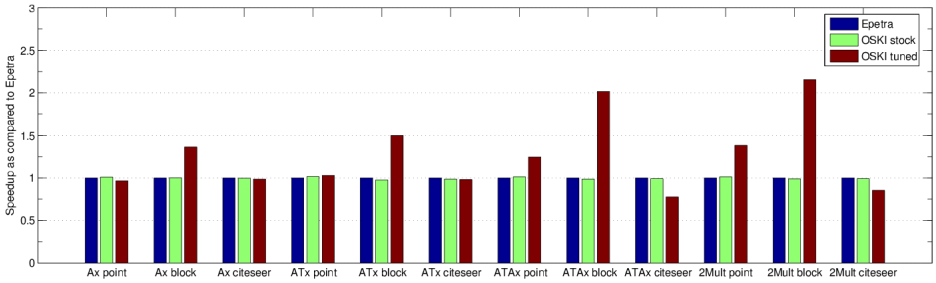


FIG. 4.2. *Relative performance of Epetra and OSKI in serial on Niagara.*

Figure 4.2 shows that on the Niagara, the stock OSKI and Epetra kernels had roughly the same performance Tuning for point matrices once again resulted in either no gains or slight losses. Tuning for block matrices resulted in a one third to one half gain in speed. Again, composing increased the speed of all kernels significantly, except for the Citeseer matrix, for which the OSKI kernels where actually slower.

As expected, the serial tests show that the tuning of point matrices is counterproductive, except when needed to use composed kernels. However, tuning of block matrices results in significant speedups through the reduction of indirect addressing. For the pseudo random Citeseer matrix, tuning is never beneficial. This is probably due to either lack of cache-blocking in the composed kernels and/or more random access, which create a greater number of cache misses. For structured matrices, composing results in a 25% to 60% gain over the faster of the stock and tuned kernels.

Even if the tuning gains shown above are large, the amount of time it takes to tune a matrix at runtime is important in determining whether tuning will result in performance gains. Tables 4.4, 4.5 and 4.6 show the cost of tuning and the number of matrix-vector calls needed to amortize that cost for the point, block, and Citeseer matrices, respectively. The tuning and retuning costs are expressed in terms of the number of matrix-vector multiplies that could be performed in the time it takes to tune. *Tuning cost* is the amount of time it takes to tune a matrix the first time, and includes time to analyze the matrix to determine what optimizations are beneficial. *Retuning cost* is the amount of time it takes to tune the matrix if the optimizations to be performed are already known. All comparisons are to the faster of the Epetra and OSKI matrix-vector multiplies. The amortize columns show the number of calls to the tuned kernel needed to realize tuning gains. When N/A is listed in an amortize column,

it is never better to tune because the tuned kernels are no faster than the untuned kernels. We note that the tuning cost depends only on the matrix structure, not on the matrix kernel to be performed.

Machine	Tune/Retune	Amortize Ax/Retune	Amortize A^TA/Retune	Amortize 2Mult/Retune
Clovertown	37.6 / 20.1	N/A	48 / 26	45 / 24
Niagara	22.1 / 12.7	N/A	56 / 33	40 / 24

TABLE 4.4
OSKI tuning costs for point matrix. Cost is equivalent number of matrix-vector multiplications.

Machine	Tune/Retune	Amortize Ax/Retune	Amortize A^TA/Retune	Amortize 2Mult/Retune
Clovertown	31.1 / 17.7	131 / 75	27 / 16	28 / 16
Niagara	22.5 / 14.1	86 / 54	22 / 14	21 / 13

TABLE 4.5
OSKI tuning costs for block matrix. Cost is equivalent number of matrix-vector multiplications.

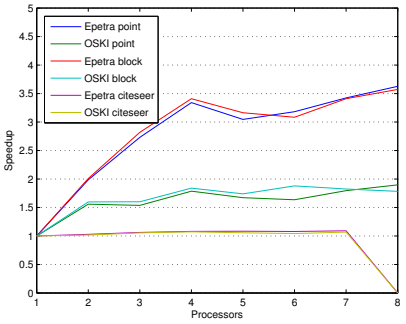
Machine	Tune/Retune	Amortize Ax/Retune	Amortize A^TA/Retune	Amortize 2Mult/Retune
Clovertown	14.5 / 6.7	N/A	N/A	N/A
Niagara	11.5 / 5.2	N/A	N/A	N/A

TABLE 4.6
OSKI tuning costs for Citeseer matrix. Cost is equivalent number of matrix-vector multiplications.

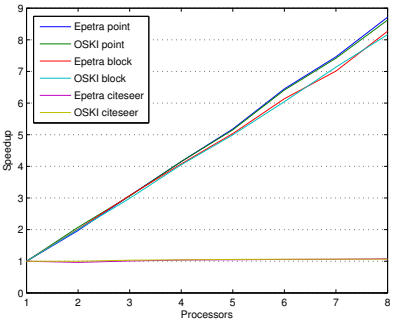
In many cases, the tuned OSKI kernels are much more efficient than the Epetra and OSKI stock kernels. However, the data structure rearrangement required to create an OSKI kernel is non-trivial. The cost of tunings ranges from 11.5 to 37.6 equivalent matrix-vector multiplies. It can require as many as 131 subsequent kernel applications to recoup the cost of initial tuning. However, re-tuning costs are usually slightly over half the cost of the initial tuning, so saving transformations for later use could be profitable. Block matrices require the smallest number of calls to recover tuning costs, and when combined with composed kernels, this number drops even more. For point matrices tuning the matrix-vector multiply is never profitable, but the tuning of composed kernels can be profitable for structured matrices.

While serial performance is important to application performance, most scientific simulations are run on parallel machines. The first level of parallelism is within a single node, which typically contains one or two multicore processors. To test the scalability of our implementation of OSKI, within Epetra, we ran tests on each matrix on 1 to 8 cores of each machine and also on 1 to 8 threads per core on the Niagara.

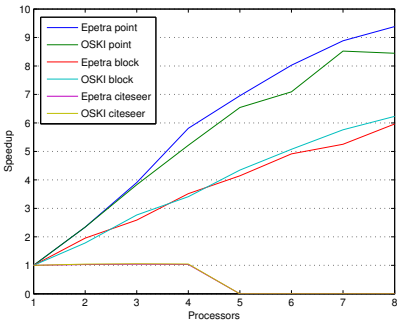
Figures 4.3(a)-4.3(c) show the strong scaling of the matrix-vector kernel for each matrix. Figure 4.3(a) shows that on the Clovertown that Epetra has better scaling than OSKI. Table 4.7 shows, however, that the overall performance of OSKI is either comparable or better to that of Epetra. The better scaling for Epetra comes from its slower performance in the single processor case, which allows for more improvement within a limited memory bandwidth situation. For the point matrix, both Epetra and OSKI improve significantly until each is running at about 735 Mflops on 4 cores. At this point, the calculations likely become memory bandwidth limited. With added processing power, the speeds then improve to slightly under



(a) Clovertown



(b) Niagara



(c) Niagara multi-threaded

FIG. 4.3. *OSKI matrix-vector multiply strong scaling results.*

800 Mflops. The block matrix results show a similar pattern, with the OSKI block matrix remaining more efficient throughout. The Citeseer matrix does not scale most likely due to the large amounts of data it needs to exchange, because its unstructured. Also it could not be run on 8 processors due to an increasing memory footprint, perhaps due to exchanged data.

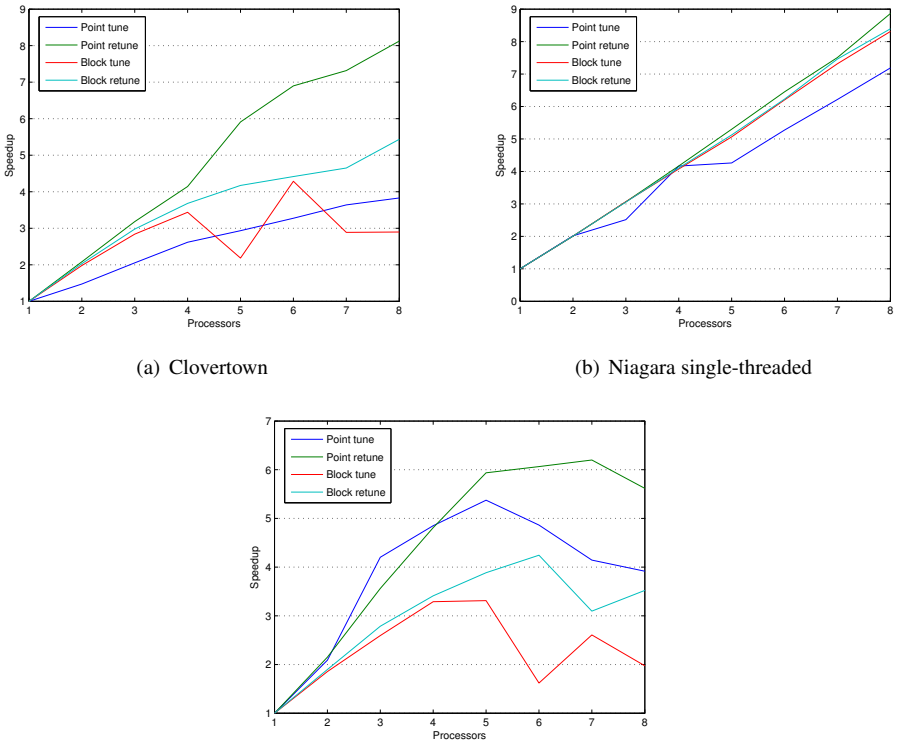
machine	point Epetra/OSKI	block Epetra/OSKI	Citeseer Epetra/OSKI
Clovertown	798/782	810/1099	59.6/122
Niagara 1 thread/core	508/507	578/778	22.3/22.0
Niagara multiple threads/core	4767/4321	3447/4847	23.2/23.2

TABLE 4.7
Epetra and OSKI maximum parallel matrix vector multiply speeds in Mflops.

Figure 4.3(b) shows that on the Niagara both the point and block matrix algorithms scale linearly with the number of cores. Essentially, there is enough memory bandwidth to feed each core. As seen in Figure 4.3(c), adding more threads per core to the calculating power leads to approximately linear speedup for all matrices. This begins to tail off at 5 threads for block matrices, and 7 threads for point matrices. The Citeseer matrix once again does not scale and becomes too large to run above 32 threads.

Scalability also matters when a matrix is being tuned. Figures 4.4(a)-4.4(c) show how well each matrix scales on each machine in terms of tuning cost. Scaling is usually linear

or slightly better with the number of processors. This result is expected as tuning is a local computation with no communication between processors. As seen in Figure 4.4(c), increasing the number of threads per Niagara processor initially leads to improved performance, before dropping off at 6 or more threads per processor. The dropoff is most likely due to threads competing for processor resources. Results for the Citeseer matrix were not shown, as OSKI does not tune its matrix-vector multiply kernel for the Citeseer matrix. Finally, note that the retune function demonstrates better scaling than the same tune function in all cases.



(a) Clovertown

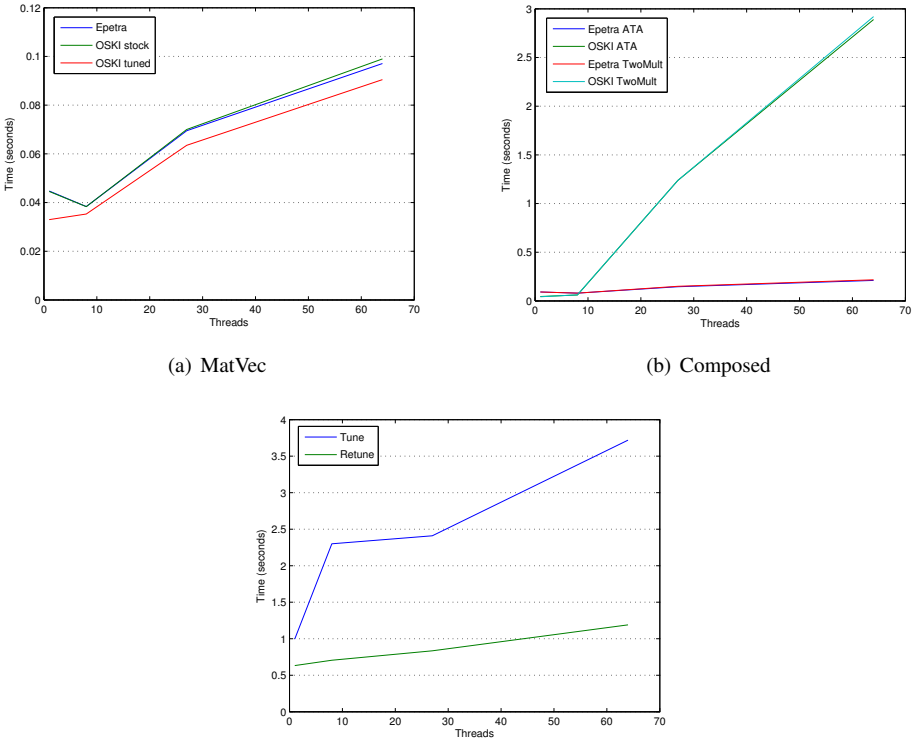
(b) Niagara single-threaded

(c) Niagara multi-threaded

FIG. 4.4. *Scalability of OSKI tuning.*

In addition to strong scaling tests, we also ran a weak scaling test on the Niagara. We used the block matrix from the 8 thread test case in Table 4.2. Tests were run on 1, 8, 27 and 64 threads. Results are shown in Figures 4.5(a)-4.5(c). As seen in Figure 4.5(a), the OSKI tuned and untuned matrix-vector multiplies both scale similarly to Epetra's matrix-vector multiply. Figure 4.5(b), shows that the tuned composed kernels do not scale well. The same result was seen for the untuned composed kernels. For these operations to be possible there is extra data copying in the wrapping of the serial kernels, which could be the problem. There could also be inefficiencies in the code in other places or resource contention on the processor. Figure 4.5(c) shows that re-tuning scales better than tuning as the problem size grows.

5. Conclusions. Overall, OSKI can produce large speedups in sparse matrix computational kernels. This is especially true when the matrix is block structured or multiple multiplications are performed using the same matrix. In some cases it can also produce large gains



(a) MatVec

(b) Composed

(c) Tuning

FIG. 4.5. *Weak scalability of OSKI on Niagara*

for matrix-vector multiplies involving only a single matrix. However, OSKI is still missing some features, such as a multi-vector kernel and the ability to tune matrices to make them symmetric. Both could produce large runtime gains. Our Epetra/OSKI interface has stubs to allow the use of these missing features as soon as they become available in OSKI. Our experiments show that Sandia applications that make heavy use certain sparse matrix kernels can benefit from the current version of OSKI. As new OSKI features become available, its potential impact on other Sandia applications should increase.

6. Future Work. For the current (1.0.1h) version of OSKI, a developer may want to implement the solve function and run more weak scalability or other parallel tests to determine why the composed kernels do not scale well. For a newer version of OSKI, a developer may want to test any new tuning features, the matrix power kernel, as well as any other new functions. Finally, we recommend any new version of OSKI be tested on the Barcelona and Xeon chips, as we were never able to successfully install OSKI on these architectures. The Barcelona is of particular interest, as it is the processor found in the center section of Red Storm.

7. Acknowledgments. We would like to thank Brian Barrett and Doug Doerfler for access to the Niagara and Clovertown architectures, respectively. We also would like to thank Danny Dunlavy and Chris Siefert for providing us with the test matrices. In addition, we would like to thank Mike Heroux, Chris Siefert and Jim Willenbring for reviewing the interface design and answering questions along the way. Jim's partial OSKI implementation

of an interface within Kokkos helped serve as a model for our development. Finally, we would also like to thank Rich Vuduc for his help with Oski-related questions.

REFERENCES

[1] E. Angerson, Z. Bai, J. Dongarra, A. Greenbaum, A. McKenney, J. Du Croz, S. Hammarling, J. Demmel, C. Bischof, and D. Sorensen, *Lapack: A portable linear algebra library for high-performance computers*, Nov 1990, pp. 2–11.

[2] Berkeley Benchmarking and OPtimization Group, *OSKI: Optimized Sparse Kernel Interface*. http://bebop.cs.berkeley.edu/oski/about.html, May 2008.

[3] J. Bilmes, K. Asanovic, C.-W. Chin, and J. Demmel, *Optimizing matrix multiply using phipac: a portable, high-performance, ansi c coding methodology*, in ICS '97: Proceedings of the 11th international conference on Supercomputing, New York, NY, USA, 1997, ACM, pp. 340–347.

[4] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley, *An updated set of Basic Linear Algebra Subprograms (BLAS)*, ACM Transactions on Mathematical Software, 28 (2002), pp. 135–151.

[5] I. S. Duff, M. A. Heroux, and R. Pozo, *An overview of the sparse basic linear algebra subprograms: The new standard from the BLAS technical forum*, ACM Transactions on Mathematical Software (TOMS), 28 (2002).

[6] G. R. Gao, R. Olsen, V. Sarkar, and R. Thekkath, *Collective loop fusion for array contraction*, in 1992 Workshop on Languages and Compilers for Parallel Computing, no. 757, New Haven, Conn., 1992, Berlin: Springer Verlag, pp. 281–295.

[7] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley, *An overview of the Trilinos project*, ACM Trans. Math. Softw., 31 (2005), pp. 397–423.

[8] M. A. Heroux and P. M. Sexton, *Epetra developers coding guidelines*, Tech. Rep. SAND2003-4169, Sandia National Laboratories, Albuquerque, NM, December 2003.

[9] Sandia National Laboratories, *Epetra - Home*. http://trilinos.sandia.gov/packages/epetra/index.html, May 2008.

[10] ———, *Kokkos - Home*. http://trilinos.sandia.gov/packages/kokkos/index.html, May 2008.

[11] ———, *Teuchos - Home*. http://trilinos.sandia.gov/packages/teuchos, May 2008.

[12] R. Vuduc, *Personal Communication*, July 2008.

[13] R. Vuduc, J. W. Demmel, and K. A. Yelick, *Oski: A library of automatically tuned sparse matrix kernels*, Journal of Physics Conference Series, 16 (2005), pp. 521–530.

[14] ———, *The Optimized Sparse Kernel Interface (OSKI) library user's guide for version 1.0.1h*, tech. rep., University of California at Berkeley, Berkeley, CA, June 2007.

[15] R. C. Whaley and J. J. Dongarra, *Automatically tuned linear algebra software*, Procceding of the 1998 ACM/IEEE conference on Supercomputing (CDROM), (1998), pp. 1–27.

ALGEBRAIC MULTIGRID FOR POWER GRID NETWORKS

YAO CHEN * AND RAY TUMINARO †

Abstract. We propose a multigrid algorithm for solving linear systems associated with saddle point systems where the constraint equations correspond to either Dirichlet or Neumann type. Such systems arise in power grid networks. A key aspect of this multigrid algorithm is a special prolongator. We show that if the prolongator satisfies conditions with respect to the constraints, one is able to project the original saddle point system to a coarse level that is symmetric positive definite without constraint equations. This implies that a more standard multigrid algorithm can be applied to the coarse level matrix. Numerical results are given to demonstrate the multigrid convergence rate.

1. Problem description and notation. We are interested in solving linear systems of the form

$$\begin{pmatrix} A & B \\ B^T & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} g \\ h \end{pmatrix}$$

where the coefficient matrix will be denoted by K. These type of matrices arise in many applications including power grid networks. For power grid networks, the matrix A describes the interconnection of resistors, capacitors, and inductors, while B is used to describe voltage sources and short circuits. Figure 1 gives an example of a power grid circuit (with resistors of 1Ω) and the corresponding matrix representation of Kirchhoff's law.

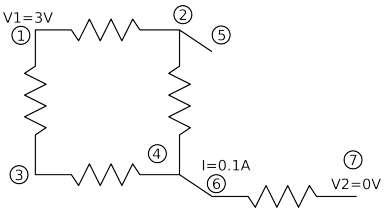


Figure 1

$$A = \begin{pmatrix} 2 & -1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 2 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 2 & -1 & 0 & 0 & 0 \\ 0 & -1 & -1 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix}, B = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

$$g = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0.1 & 0 \end{pmatrix}^T, \quad h = \begin{pmatrix} 3 & 0 & 0 & 0 \end{pmatrix}^T$$

We assume the following properties:

1. A is symmetric positive semidefinite.
   In the case of circuits this is typically a weighted graph Laplacian. That is, the negative sum of the off-diagonal elements within a row gives the diagonal element.
2. B is a $m \times n$ matrix, $m \geq n$. B is of full rank. The j-th column of B is either
   (a) Dirichlet type: $B_{ij} = 1, B_{kj} = 0, \forall k \neq i$.
   There is a one in one location and zeros elsewhere,
   e.g. $(0, 0, \ldots, 1, 0, \ldots, 0)^T$.

*The Pennsylvania State University, chen_y@math.psu.edu
†Sandia National Laboratories, rstumin@sandia.gov

(b) Neumann type: $B_{i_1 j} = 1, B_{i_2 j} = -1, B_{kj} = 0, \forall k \neq i_1, k \neq i_2$.
There is a $(1, -1)$ pair and zeros elsewhere,
e.g. $(0, 0, \ldots, 1, 0, \ldots, -1, 0, \ldots, 0)^T$.
In the above example, the first two columns of B are Dirichlet type and the last two are Neumann type.

3. K is nonsingular.

Clearly, the circuit in Figure 1 satisfies these assumptions as do most power grid networks.

The above properties are the only assumptions that we make in the algorithms and analysis that follows. It should be noted, however, that often matrices arising from power grids also satisfy these additional assumptions:

1. A is weakly diagonal dominant.
2. the eigenvector of A corresponding to the eigenvalue zero can only be the constant vector $\vec{1}$ or a canonical unit vector, e_i.
3. When the i-th column of B is of "Neumann type", the i-th component of h is zero.

However, our algorithm is more general than this as it does not require these additional assumptions.

2. Basic idea. It is well known that K is indefinite with n negative eigenvalues. This complicates the application of iterative methods such as multigrid. Our aim is to "reduce" (or simplify) K to a symmetric positive definite matrix so that we could apply standard iterative methods such as conjugate gradient or some standard multigrid methods to solve it. This is done by essentially projecting the equations into a subspace where the constraints are satisfied. To do this, we first determine an initial guess x_0, which satisfies

$$B^T x_0 = h.$$

It is not difficult to find such an x_0 due to the simple structure of B (see appendix for details).

Assign $y_0 = 0$, and let $\hat{g} = Ax_0$ so that

$$K \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = \begin{pmatrix} \hat{g} \\ h \end{pmatrix}.$$

Then solve the residual equations

$$K \begin{pmatrix} \delta x \\ \delta y \end{pmatrix} = \begin{pmatrix} g - \hat{g} \\ 0 \end{pmatrix}. \tag{2.1}$$

Define Q to be

$$Q = I - B(B^T B)^{-1} B^T.$$

Notice that Q is a projection so $QB = 0$, $B^T Q = 0$, and Ker Q = Range B. We then use Q to rewrite (2.1).

THEOREM 2.1. *the following three steps generate the solution of the above residual equation.*

1) Solve for z in

$$QAQz = Q(g - \hat{g}).$$

2) Compute δx via

$$\delta x = Qz.$$

3) Solve

$$B^T B\delta y = B^T(g - \hat{g} - A\delta x).$$

for δy.

Proof. Suppose that δx and δy are obtained by this three-step method. Then it follows that

$$B^T \delta x = B^T Q z = 0 \quad \text{as } B^T Q = 0.$$

Thus, δx satisfies the constraint relation in (2.1). Step 1 and Step 2 indicate that

$$QA\delta x = Q(g - \hat{g}).$$

Therefore,

$$A\delta x = g - \hat{g} + c$$

where $c \in \ker Q$. Since $\ker Q = \text{Range } B$, there exists a δy such that $B\delta y = -c$. This δy can be obtained by the normal equations system given in Step 3 as B is of full column rank. □

Since A is positive semidefinite, QAQ is positive semidefinite. We can now apply a general iterative method to solve z in Step 1, e.g. preconditioned conjugate gradient. We must determine a suitable multi-level method for QAQ without requiring the explicit formation of QAQ.

3. Two-level method, designing the prolongation operator. We now focus on defining a multigrid method for Step 1 of the above method. A two-level multigrid method is given below.

Set up initial guess $x = x_0$.
Do the following until some error tolerance is satisfied.
(1) Do ν_1 times relaxation (smoothing):

$$z \leftarrow 0, \quad z \leftarrow R^{\nu_1}(QAQ, z, Q(g - Ax)), \quad x \leftarrow x + Qz.$$

(2) Solve z_c on coarse grid,

$$P^T QAQP z_c = P^T Q(g - Ax).$$

(3) Project correction back to fine grid, or

$$x \leftarrow x + QPz_c.$$

Relaxation refers to a simple iterative procedure such as the Jacobi method. Since this is a two-level method, it is not necessary to do post-smoothing, though this can easily be added. The problem is that in order to calculate $P^T QAQP$, the coefficient matrix on the coarse grid, it appears necessary to form QAQ. However, Q contains $(B^T B)^{-1}$, which is hard to evaluate and sometimes not sparse. Our idea is to instead construct a prolongation P, such that $QP = P$. Therefore we can avoid matrix-matrix multiplication with Q as the coefficient matrix on the coarse level is

$$A_2 = P^T QAQP = P^T AP.$$

Thus the coarse grid correction (Step 2 and Step 3 above) is

$$x \leftarrow x + PA_2^{-1}P^T(g - \hat{g}).$$

P is typically defined using some structure of an underlying mesh within geometric multigrid. While this may be natural for partial differential equations, it is not very practical for circuits as there is no notion of an underlying mesh. Instead, algebraic multigrid methods seem more appropriate. With algebraic methods, P is chosen based on the fine grid matrix. In our case this corresponds to the matrix QAQ. However, we wish to avoid the need to explicitly calculate QAQ as Q contains $(B^TB)^{-1}$, and it may give rise to a matrix QAQ with many more non-zeros than A. Instead, we seek a matrix P which can be constructed without forming QAQ and which satisfies the following properties:

1. P is of full rank.
2. $QP = P$.
3. P^TQAQP is sparse.
4. each row of P has at most one non-zero entry.

One important property of such a prolongator is that it guarantees that P^TQAQP is non-singular though the matrix QAQ can be singular.

THEOREM 3.1. *Assume P is of full rank and that $QP = P$. Then, $A_2 = P^TQAQP$ is positive definite.*

Proof. As A is positive semidefinite, we have

$$\forall x, \quad x^TAx \geq 0.$$

When $x = Pv$, it follows immediately that

$$\forall v, \quad v^TP^TAPv \geq 0.$$

Suppose that there exists a $v \neq 0$, such that

$$v^TP^TAPv = 0.$$

The assumption that P is of full rank implies that

$$x = Pv \neq 0$$

and so x must be a linear combination of eigenvectors of A corresponding to the zero eigenvalues. Therefore $Ax = 0$. We also have that $B^Tx = 0$ as $B^Tx = B^TPv = B^TQPv = 0$. This, however, implies that

$$\begin{pmatrix} A & B \\ B^T & 0 \end{pmatrix}\begin{pmatrix} x \\ 0 \end{pmatrix} = K\begin{pmatrix} x \\ 0 \end{pmatrix} = 0$$

which contradicts the assumption that K is non-singular. □

The algorithm to construct the prolongator is given in the appendix. The basic idea somewhat mirrors the tentative prolongator that is used in smoothed aggregation [3, 2] which is based on piecewise-constants. However, special care is needed to satisfy $QP = P$.

4. Numerical results. To test this method, we use a realistic benchmark provided by IBM. [1] The sizes of the matrices from the first benchmark are

- A : 30635×30635;

[1] IBM powergrid benchmark, `http://dropzone.tamu.edu/~pli/PGBench/`.

- $B : 30635 \times 14308$;

In this particular case $B^T B$ is a diagonal matrix. This is due to the simple structure of the Neumann type constraints which represent these trivial short circuits and makes evaluations with Q straight-forward.

We consider the following algorithms in our experiments:

1. (GMRES[10]) GMRES method restarting for every 10 steps.
2. (MG-CG)
   (a) 8 conjugate gradient iterations with preconditioner `diag(`QAQ`)` as a multigrid pre-smoother.
   (b) Exact solver on the second level (exact A_2^{-1}).
   (c) No post-smoother.
3. (MG-JAC)
   (a) 2 Jacobi iterations as a multigrid pre-smoother.
   (b) Exact solver on the second level.
   (c) No post-smoother.
4. (CG-MG-JAC) We use one MG-JAC V-cycle as a preconditioner of an outer CG method. Here, we use one Jacobi iteration as pre-smoother and another one as the post-smoother.

The following graph shows the convergence history. We tested the relative 2-norm of the residual. The x axis indicates the number of relaxations.

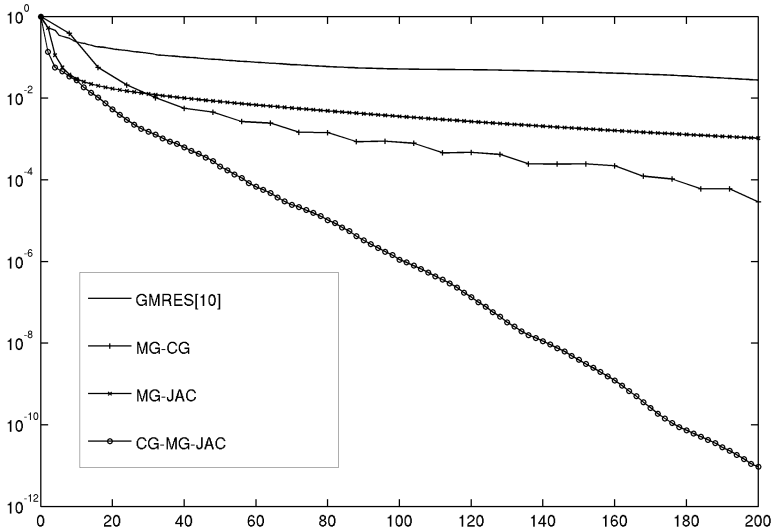


Figure 2

We can see that the best method is clearly CG-MG-JAC as it has a better asymptotic convergence rate than the other methods.

In order to show the scalability of this method, we applied it to a larger problem. Consider a piece of a square semiconductor sheet with impurities inside. The square is discretized on a $2^9 \times 2^9$ uniform grid and can be modeled as a resistor network on that grid, with many open circuits and shorts. In this example, the sizes of the matrices are

- $A : 2^{18} \times 2^{18}$;
- $B : 2^{18} \times 1423$;

and Q has a 400×400 full block so we did not store it but instead evaluate matrix-vector products and solves involving $B^T B$. See appendix for a mathematical description of the problem.

We consider the following algorithms

1. (MG-JAC)
   (a) 1 Jacobi iteration as the pre-smoother.
   (b) 1 V-cycle multigrid correction.
   (c) 1 Jacobi iteration as the post-smoother.
2. (MG-CG)
   (a) 8 conjugate gradient iterations with preconditioner `diag(`A`)` as the pre-smoother.
   (b) 1 V-cycle multigrid correction.
   (c) No post-smoother.
3. (CG-MG-JAC[2,2]) We use one MG-JAC V-cycle as a preconditioner of an outer CG method. Here, we use two Jacobi iterations as the pre-smoother and another two as the post-smoother.
4. (CG-MG-JAC[1,1]) We use one MG-JAC V-cycle as a preconditioner of an outer CG method. Only one Jacobi iteration is used as the pre-smoother and another one as the post-smoother.

The convergence history is showed below, where the x-axes stands for the number of relaxation sweeps on the fine grid and the y-axes is the relative residual in 2-norm.

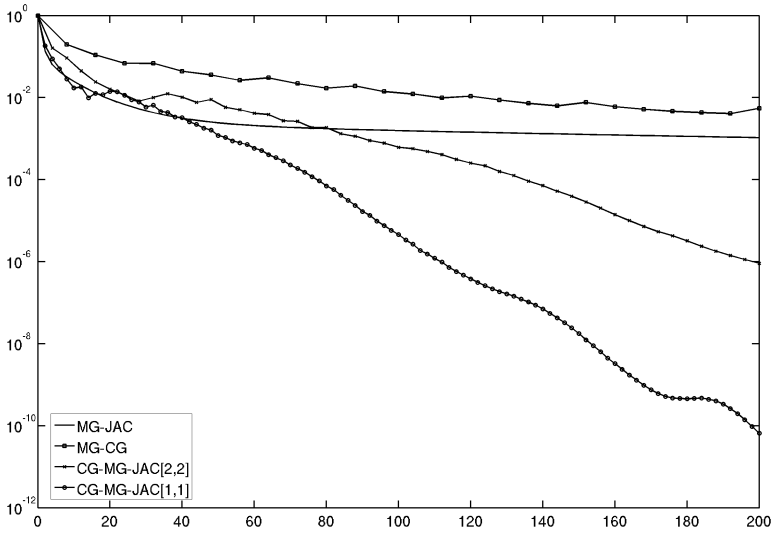


Figure 3

We notice that although the matrix in this case is larger than that of the previous example, we could obtain the the same accuracy by carrying out approximately the same number of iterations.

5. Conclusion. We have presented a new multigrid algorithm for saddle point systems where the constraints are of either Dirichlet type or Neumann type. This new algorithm essentially converts the saddle point system to a symmetric positive definite system where more standard algorithms can be applied. Additionally, it avoids matrix-matrix multiplication involving the projection operator associated with the constraints. This is done by means of a special prolongator which guarantees that constraints are satisfied after the prolongation step.

REFERENCES

[1] W. L. Briggs, V. E. Henson, and S. McCormick, *A Multigrid Tutorial, Second Edition*, SIAM, Philadelphia, 2000.

[2] P. Vaněk, J. Mandel, and M. Brezina, *Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems*, Computing, 56 (1996), pp. 179–196.
[3] ———, *Convergence of algebraic multigrid based on smoothed aggregation*, Numerische Mathematik, 88 (2001), pp. 559–579.

Appendix

A. Satisfying constraints via x_0. We use Gaussian elimination to obtain a possible solution x_0 such that $B^T x_0 = h$. Thus

$$[B^T, h] \rightarrow \text{Gaussian elimination} \rightarrow [B^{*T}, h^*]$$

so that B^{*T} is an upper triangular matrix. Then we do back-substitution on B^{*T} and assign all free variables zero. In the case of a power grid circuit, the structure of B is simple. In fact, the number of non-zero entries in B^* is less than or equal to that of B. In particular, suppose that the i-th row of B^{*T} is of Dirichlet type and $(B^{*T})_{ij}$ is the non-zero entry. The Gaussian elimination process will make all entries in the j-th column below $(B^{*T})_{ij}$ zero and will not create any new nonzeros. Suppose that the i-th row of B^{*T} is of Neumann type and $(B^{*T})_{ij_1}$ and $(B^{*T})_{ij_2}$ are the non-zero entries. Every time Gaussian elimination zeros out one non-zero in $(B^{*T})_{kj_1}$, it might introduce at most one new non-zero in $(B^{*T})_{kj_2}$.

In summary, the total number of non-zero entries does not increase. So it is not numerically difficult to find the solution x_0. For the example appearing in the first section, $x_0 = (3, 0, 0, 0, 0, 0, 0)^T$ is an initial guess.

B. Prolongator Construction. First we exclude the trivial case $Q = 0$ where $P^T QAQP$ is always zero. Since column vectors in B are linearly independent, $Q = 0$ only if B is a square matrix. We can assume that B is not a square matrix, otherwise the initial guess x_0 is the solution.

Define U^i to be a binary vector of length m indicating which fine grid nodes need to be aggregated at the beginning of the i-th iteration. The k-th component of U^i is

$$(U^i)_k = \begin{cases} 0, & \text{if the } k\text{-th node is aggregated} \\ 1, & \text{if the } k\text{-th node has not been aggregated.} \end{cases}$$

Let m be the number of rows in A. The following method generates a prolongation matrix P with the required properties. Set $i = 1$ and

$$(U^i)_k = \begin{cases} 0, & \text{if } \exists j, \text{ such that } (B)_{kj} \text{ is the only nonzero entry on the } j\text{-th column} \\ 1, & \text{elsewhere.} \end{cases}$$

Repeat the following process until U^i is a zero vector, or $\|U^i\|_2 = 0$.

(1) Find an aggregate p by using the sparsity pattern of A, where p is a binary m-vector and $(p)_k = 1$ indicates that the k-th vertex is in the aggregate and $(p)_k = 0$ indicates that it is not .[2] The nonzero entries of p form a subset of that of U.

(2) Define the k-th column of P as

$$(P)_{ki} = \begin{cases} 0, & \text{if } (Qp)_k \times (U^i)_k = 0 \\ 1, & \text{if } (Qp)_k \times (U^i)_k \neq 0. \end{cases}$$

(3) $U^{i+1} \leftarrow U^i - (P)_{*i}$ where $(P)_{*i}$ is the i-th column of P.

(4) $i \leftarrow i + 1$ This method has no matrix-matrix multiplications.

We introduce two lemmas.

Lemma B.1 (A1). *Given U^i, if $B^T U^i = 0$, then after one iteration, we get a column $(P)_{*i}$ satisfying $B^T (P)_{*i} = 0$.*

[2] Typically, a graph algorithm is performed to group together a set of nearest neighbor entries in the graph of A.

Proof. Rewrite the definition of $(P)_{*i}$ as

$$(P)_{ji} = \begin{cases} 0, & \text{if } (Qp)_j = 0 \text{ or } (U^i)_j = 0 \\ 1, & \text{elsewhere} \end{cases} .$$

Define

$$v = B^T (P)_{*i}$$

so we must show $\forall k, v_k = 0$.

Case 1: $(B^T)_{k*}$ corresponds to a Dirichlet row. This means that there is only one nonzero in this row, $(B^T)_{kj}$.
$(B^T)_{k*}U^i = 0$ then implies that $(U^i)_j = 0$ and so $P_{ji} = 0$. It follows that $v_k = (B^T)_{k*}(P)_{*i} = 0$.

Case 2: $(B^T)_{k*}$ corresponds to a Neumann row. Let the two nonzeros in this row be denoted $(B^T)_{kj_1}$ and $(B^T)_{kj_2}$.
$(B^T)_{k*}U^i = 0$ implies $(U^i)_{j_1} = (U^i)_{j_2}$. (1)
$B^T Q = 0$ implies $(B^T)_{k*}Qp = 0$, and so $(Qp)_{j_1} = (Qp)_{j_2}$. (2)
From equations (1) and (2) we can infer that

$$(P)_{j_1 i} = (U^i)_{j_1}(Qp)_{j_1} = (U^i)_{j_2}(Qp)_{j_2} = (P)_{j_2 i}$$

and thus $(B^T)_{k*}(P)_{*i} = 0$.

So $B^T(P)_{*i} = 0$. □

Lemma B.2 (A2). *Assume $B^T(P)_{*i} = 0$ and $B^T U^i = 0$. Define $U^{i+1} \leftarrow U^i - (P)_{*i}$. Then $B^T U^{i+1} = 0$.*

Proof. $B^T U^{i+1} = B^T(U^i - P_{*i}) = 0$ □

Finally, we show that P has the desired properties.

Theorem B.3. *Let P be constructed by the algorithm above. Then, the following is true.*

1. *P is of full rank.*
2. *QP = P.*
3. *each row of P has at most one non-zero entry.*

Proof.
(Property 3) By construction of the i-th column of P, P_{ji} can only be nonzero if $(U^i)_j = 1$. This only occurs if $P_{jk} = 0, k < i$. $(U^{i+1})_j$ is then set to zero by the update $(U^{i+1})_j = (U^i)_j - P_{*i}$.

(Property 1) Follows Property 3 and the assumption that no column is entirely zero. We verify that columns of P are orthogonal to each other, therefore P is of full rank.
(Property 2) To show $QP = P$, we verify that $Q(P)_* i = (P)_* i$, for the i-th column of P. It is enough to show $B^T(P)_* i = 0$, which is done inductively.

Base Case

U^1 satisfies $B^T U^1 = 0$.

By Lemma (A1), $B^T(P)_{*1} = 0$.

Inductive Step

Assume $B^T U^i = 0$, $B^T(P)_{*i} = 0$, then $B^T U^{i+1} = 0$ and $B^T(P)_{*i+1} = 0$. These follow directly from Lemma (A1) and Lemma (A2).

Repeat this process to obtain all columns of P.

Therefore $B^T p = 0$ so $QP = P$. □

For the example on page 1, the whole process to obtain P is given below.

1. $i = 1$.
2. Initialize U^1 as

$$U^1 = (0, 1, 1, 1, 1, 1, 0)^T.$$

3. Repeat,
   (a) Find an aggregate by examining A, gives $p = (0, 1, 1, 1, 0, 0, 0, 1)^T$
   (b) Evaluate $Qp = (0, 1/2, 1, 1/2, 1/2, 1/2, 0)^T$
   (c) Determine $P = (0, 1, 1, 1, 1, 1, 0)^T$
   (d) Update $U^2 \leftarrow U^1 - P = 0$

This loop is only executed once as U^2 is empty. The prolongator contains only one column. We evaluate $A_2 = P^T AT = 3$, which is non-singular.

C. The second numerical test example. Define Ω to be a square domain $[0, 1] \times [0, 1]$. The left, right, top, and bottom edges are denoted Γ_l, Γ_r, Γ_t, and Γ_b respectively. Define Ω_1 to be $[0.2, 0.4] \times [0.2, 0.4]$, Ω_2 to be $[0.6, 0.8] \times [0.6, 0.8]$.

Then

$$\begin{aligned} \Delta u &= 0 \quad \text{on } \Omega - \Omega_1 - \Omega_2, \\ u &= 1 \quad \text{on } \Gamma_l, \\ u &= 0 \quad \text{on } \Gamma_r, \\ \nabla u \cdot \mathbf{n} &= 0 \quad \text{on } \Gamma_t, \Gamma_b, \partial\Omega_1, \end{aligned}$$

and

$$|\nabla u| = 0 \quad \text{on } \partial\Omega_2$$

which is discretized on a 512×512 uniform grid.

MULTIGRID CONSIDERATIONS FOR STOCHASTIC PROJECTION SYSTEMS

ROBERT D. BERRY [‡], RAY S. TUMINARO [§], AND HABIB N. NAJM [¶]

Abstract. We are interested in employing algebraic multigrid methods to large linear systems associated with stochastic projection systems. To this end, we first seek to understand some basic matrix properties of these linear systems. In some cases, it is possible to give conditions for positive definiteness of the linear system associated with the stochastic PDE as well as to gain some insight into the structure of this matrix. We then give a multigrid method that is suitable for some of these systems. This method is primarily based on coarsening in the spatial direction. We also explore some coarsening in the stochastic direction and present some preliminary findings.

1. Introduction. We consider diffusion in a one-dimensional medium with a stochastic diffusivity. Let $(\Omega, \mathcal{M}, \mu)$ be a (probability) measure space, where Ω is a sample space, $\mathcal{M}$ is a σ-algebra, and μ is a probability measure. Our diffusion will occur on a domain $[0,1] \subset \mathbb{R}$.

Consider a stochastic diffusivity coefficient $\lambda : \Omega \times [0,1] \to \mathbb{R}$. We wish to solve the diffusion equation

$$-\mathrm{div}_x(\lambda(\omega, x)\nabla_x u(\omega, x)) = f(\omega, x)$$

with boundary conditions $u(\omega, 0) = u(\omega, 1) = 0$. In general, u is a function of both space and the sample $\omega \in \Omega$. Since we are in one space dimension, we write

$$-\frac{\partial}{\partial x}\left(\lambda \frac{\partial u}{\partial x}\right) = f. \tag{1.1}$$

Now, let $\{\xi_i(\omega)\}_{i\in\mathbb{N}}$ be a sequence of independent uniform random variables (RVs) on Ω, that is $\xi_i : \Omega \to [-1,1]$ for each i. Then any random variable $A : \Omega \to \mathbb{R}$ (with proper regularity) may be represented as a chaos expansion:

$$A(\omega) = \sum_{k=0}^{\infty} \alpha_k \Psi_k(\xi_1(\omega), \xi_2(\omega), \ldots)$$

for $\alpha_* \in \mathbb{R}$ and where Ψ_* are $\mu-$orthonormal functions.

We expand u, λ, and f by writing

$$\begin{aligned} u(\omega, x) &= \sum_k u_k(x)\Psi_k(\omega), \\ \lambda(\omega, x) &= \sum_k \lambda_k(x)\Psi_k(\omega), \text{ and} \\ f(\omega, x) &= \sum_k f_k(x)\Psi_k(\omega). \end{aligned} \tag{1.2}$$

so that

$$\begin{aligned} u_k(x) &= \int_\Omega u(\omega, x)\Psi_k(\omega)\, d\mu(\omega), \\ \lambda_k(x) &= \int_\Omega \lambda(\omega, x)\Psi_k(\omega)\, d\mu(\omega), \text{ and} \\ f_k(x) &= \int_\Omega f(\omega, x)\Psi_k(\omega)\, d\mu(\omega). \end{aligned}$$

[‡]University of Pittsburgh, rdb6@pitt.edu
[§]Sandia National Laboratories, rstumin@sandia.gov
[¶]Sandia National Laboratories, hnnajm@sandia.gov

Substituting these expressions into (1.1), we obtain

$$-\sum_{j,k} \frac{\partial}{\partial x}\left(\lambda_k(x)\frac{\partial}{\partial x}u_j(x)\right)\Psi_j(\omega)\Psi_k(\omega) = f(\omega, x).$$

Multiplying both sides by $\Psi_i(\omega)$ and integrating over Ω yields:

$$-\sum_{j,k} M_{ijk}\frac{\partial}{\partial x}\left(\lambda_k(x)\frac{\partial}{\partial x}u_j(x)\right) = f_i(x)$$

where

$$M_{ijk} = \int_\Omega \Psi_i\Psi_j\Psi_k \, d\mu.$$

The stochastic Laplace operator Δ, projected in this polynomial chaos basis, may be thought of as a matrix of operators, whose entries (or blocks) Δ_{ij} are diffusion operators, each with diffusivity

$$\lambda_{ij} = \sum_k M_{ijk}\lambda_k.$$

For convenience then we may also use the alternative notation

$$\sum_j -\Delta_{ij}u_j = f_i. \tag{1.3}$$

This operator acts on a vector whose elements are the functions u_j. In each of these expressions the random variables have been averaged over and we are left with a deterministic system of partial differential equations. Once one solves for the $\{u_k(x)\}$, the properties of $u(\omega, x)$ are determined.

We wish to numerically solve this system. In order to accomplish this, we limit the polynomial order of approximation of u and f to be P, the order of approximation of λ to be L, and the number of random variables are limited to N. In discretizing the spatial variable x, the operators Δ_{ij} are approximated by (tri-diagonal) blocks D_{ij}, so that it is the system

$$D\tilde{u} = \tilde{f}$$

that we want to solve by multigrid methods. To do this, we first wish to understand some of the basic properties of Δ (inherited by D).

2. Properties of M_{ijk}. The constants M_{ijk} determine the structure of the operator Δ. First, since we are using ortho*normal* polynomials, the operator is symmetric: $\Delta_{ij} = \Delta_{ji}$. If Δ is also positive-definite, methods such as conjugate gradient may viably be applied in solving this system. In addition to symmetry, Δ has a very particular sparse structure which is associated with the fact that many of the M_{ijk}'s are zero.

Consider the case when there is only one RV ξ, so Ψ_i is a polynomials of order i of only one variable. When the product $\Psi_i\Psi_j$ is orthogonal to Ψ_k, then $M_{ijk} = 0$. By construction of these polynomials, Ψ_k is orthogonal to any polynomial of order less than k. Thus, when the order of $\Psi_i\Psi_j$ is less than the order of Ψ_k, in other words, if $i + j < k$, then $M_{ijk} = 0$. This fact is indeed symmetric in i, j, and k.

The symmetries of ξ give further information about M. For instance, if this measure has mirror-symmetry about some point (such as Gaussian or Uniform RVs), then Ψ_k is "odd"

or "even" (with respect to the symmetry) if k is. In these cases, the integral of an "odd" polynomial is zero. Thus $M_{ijk} = 0$ if $i + j + k$ is odd.

It is not clear, *a priori*, that Δ (or D) is positive definite; the blocks of D off the diagonal include values of M_{ijk} which may be quite large. For instance, it can be shown that when Ψ_k are univariate (normalized) Hermite polynomials,

$$M_{ijk} = \frac{\sqrt{i!\ j!\ k!}}{\left(\frac{i+j-k}{2}\right)!\left(\frac{j+k-i}{2}\right)!\left(\frac{k+i-j}{2}\right)!}$$

when this expression makes sense, and $M_{ijk} = 0$ otherwise.

3. Necessary Conditions for a Positive Definite Diffusion. It is natural to ask when the operator on the l.h.s. is positive definite. The energy functional associated with this operator is (see [1])

$$K(u) = -\int_0^1 \sum_{i,j} u_i(x)\Delta_{ij}u_j(x)\,dx.$$

When this functional is positive definite, the stochastic Laplacian is also. First integrating by parts (assuming sufficient regularity to interchange summation and integration)

$$K(u) = -\int_0^1 \sum_{i,j} u_i \frac{\partial}{\partial x}\left(\lambda_{ij}\frac{\partial}{\partial x}u_j\right)\,dx = \int_0^1 \sum_{i,j} \frac{\partial u_i}{\partial x}\left(\lambda_{ij}\frac{\partial u_j}{\partial x}\right)\,dx.$$

By definition of M_{ijk} we obtain

$$\begin{aligned} K(u) &= \sum_{i,j,k}\int_0^1 M_{ijk}\lambda_k(x)\frac{\partial u_i}{\partial x}\frac{\partial u_j}{\partial x}\,dx \\ &= \int_0^1\int_\Omega \left(\sum_k \lambda_k(x)\Psi_k\right)\frac{\partial}{\partial x}\left(\sum_i u_i(x)\Psi_i\right)\frac{\partial}{\partial x}\left(\sum_j u_i(x)\Psi_j\right)\,d\mu\,dx \\ &= \int_0^1\int_\Omega \lambda(\omega,x)\left(\frac{\partial}{\partial x}u(\omega,x)\right)^2\,d\mu(\omega)\,dx. \end{aligned}$$

A *sufficient* condition for K to be positive definite is $\lambda(\omega, x) \geq \varepsilon > 0$, $(d\mu\,dx)$-almost surely (a.s.). For $u \in H^1$, this condition is also *necessary.* This can be seen by a standard argument: choosing $(\partial_x u)^2$ to approximate a delta-distribution supported within any measurable set where $\lambda \leq \varepsilon$, letting $\varepsilon \to 0$ we contradict $K(u) > 0$.

This strong necessary condition can be weakened if u resides in some other function space. This has particular implications for using unbounded random variables, such as in Gaussian models. Choosing u in a space where the delta-approximation cannot be made may lead to situations where $\lambda > 0$ a.s. is no longer necessary for positive definiteness. For instance, when the $\Psi_k(\xi_1, \xi_2, \ldots)$ are polynomials, and the order of the expansion of u is truncated at a fixed order P, then one may construct an example where $\mu\{\omega : \lambda(\omega) < 0\} > 0$ and K is positive definite. This is because in truncating the expansions, our solution space does not contain a test function (such as a delta distribution approximation) which can capture the set in $\Omega \times [0, 1]$ for which $\lambda \leq 0$. These situations appear most frequently when employing unbounded RV's, but may occur for any statistical model. Such examples exists in the literature (see e.g. [3, 4]), and we shall construct one below. If a truncation is meant as an approximation technique (such as for computational reasons), it may not be appropriate to allow $\mu\{\omega : \lambda(\omega) < 0\} > 0$.

As a statistical model for λ, consider for $\varepsilon > 0$,

$$\lambda(x, \omega) = 1 + \varepsilon\xi(\omega),$$

where ξ is a standard Gaussian random variable. The only case when λ is a.s. positive is for $\epsilon = 0$. We expand u and f as

$$u(\omega, x) = \sum_{k=0}^{p} u_k(x)\phi_k(\xi) \text{ and}$$
$$f(\omega, x) = \sum_{k=0}^{p} f_k(x)\phi_k(\xi)$$

where ϕ_k is the k-th order (normalized) Hermite polynomial. Since λ has no spatial dependence, (1.3) becomes

$$\sum_{j=0}^{p} A_{ij}(-u_j)_{xx} = f_i$$

where

$$A = \begin{pmatrix} 1 & -\varepsilon & & & & & \\ -\varepsilon & 1 & -\varepsilon\sqrt{2} & & & & \\ & -\varepsilon\sqrt{2} & 1 & -\varepsilon\sqrt{3} & & & \\ & & -\varepsilon\sqrt{3} & \ddots & \ddots & & \\ & & & \ddots & 1 & -\varepsilon\sqrt{p} & \\ & & & & -\varepsilon\sqrt{p} & 1 & \end{pmatrix}.$$

The operator in this example is positive definite if A is positive definite. For each p, there is an $\varepsilon_*(p)$ for which if $|\varepsilon| < \varepsilon_*(p)$, then A is positive definite (despite λ not being a.s. positive). Suppose A is positive definite. Then, applying Sylow's Criterion to the bottom right 2×2 minor, we have that

$$1 - p\varepsilon^2 > 0$$

whereby

$$|\varepsilon| < \frac{1}{\sqrt{p}}.$$

Allowing $\varepsilon \to \varepsilon_*(p)$, we obtain an upper bound:

$$\varepsilon_*(p) \leq \frac{1}{\sqrt{p}}.$$

In the limit $p \to \infty$, ε must be zero in order for the operator to be positive definite. Therefore (in the limit) a diffusion operator with Gaussian diffusivity is not positive definite.

4. The Karhunen-Loeve Expansion. Now we introduce a well-known expansion for stochastic functions (such as λ), known as the Karhunen-Loeve (KL) expansion. When the (spatial) covariance structure

$$C(x, y) = \mathbb{E}\{\lambda(x)\lambda(y)\}$$

for a random variable $\lambda(\omega, x)$ is known, the KL expansion provides a statistical model of the form

$$\lambda(\omega, x) = \sum_k \lambda_k(x)\theta_k(\omega),$$

where $\{\theta_k\}_{k\in\mathbb{N}}$ are a collection of RVs. This expansion decouples the variables x and ω. Posing such a model for λ, and selecting a model which is efficient in the sense of least-squares convergence, provides that the $\lambda_k(x)$ are eigenfunctions of the covariance structure $C(x, y)$. In particular,

$$\lambda_k(x) = \sqrt{a_k}\phi_k(x)$$

where

$$\int C(x, y)\phi_k(y)\, dy = a_k\phi_k(x)$$

and the ϕ_k are orthonormal with respect to C.

Furthermore, (by orthogonality) the random variables θ_k may be explicitly formed as

$$\theta_k(\omega) = \frac{1}{a_k}\int \lambda(\omega, x)\lambda_k(x)\, dx$$

and are mean-zero and uncorrelated:

$$\mathbb{E}\{\theta_k\} = 0 \qquad\qquad \mathbb{E}\{\theta_i\theta_j\} = \delta_{ij}.$$

This formulation has the advantage that the chaos expansion is only first-order. If the random variables are symmetrically distributed about some point, then our diffusion operator has on its diagonal the expectation-mode diffusion operators:

$$\Delta_{ii} = \Delta_{\lambda_0}.$$

This is determined by M_{iik} being zero unless $i + i + k$ is even. Since our model is linear in the RV's, $k \in \{0, 1\}$. Thus $k = 0$, and so only λ_0 appears along the (block) diagonal. This means that the diagonal blocks of D are identical to a standard deterministic diffusion operator. If the corresponding diagonal block entries are in some sense larger, the implication is that a standard multigrid method created to solve the diagonal blocks should be effective.

When the θ_k are Gaussian RV's, they must be independent as they are uncorrelated. Yet in this case we have the unfortunate difficulty that models for the diffusivity which give a Gaussian KL expansion (where the RV's enter linearly) will not represent a positive definite λ, as previously illustrated. When the representations are finite and the variance modes of λ are small enough, however, one may still have a positive definite approximation.

Unfortunately, these θ_k are not in general *independent* so that in order to sample a process employing this framework the joint distribution generating these θ_k must also be found. Furthermore, the joint distribution of the θ_k determines the constants M_{ijk}, which would have to be calculated for any given stochastic model for λ under consideration. Since the constants also determine the structure of our operator, little can be said about the structure of Δ, *a priori*.

Noting such computational limitations, we project the random variables θ_k onto a uniform-PC basis (on the variables ξ). Our model is then of the form

$$\lambda(\omega, x) = \sum_k \lambda_k(x)\Psi_k(\xi_1(\omega), \xi_2(\omega), \ldots)$$

where the Ψ_k is the Legendre Polynomial Chaos for the RVs $\{\xi_k\}$ independent and uniform on $[-1, 1]$. Employing uniform RVs also has other obvious computational advantages, including ease of generating and boundedness. The M_{ijk}'s may also be computed once and re-used in subsequent stochastic models. Uniform RVs also lend themselves to use in multi-wavelet type expansions.

5. Diffusivity Modes: Truncation, Decay Rates, and Positive Definiteness. For computational purposes, one must truncate the order of expansion for u, f, and λ, as well as employ a finite number of random variables. The immediate question is: to which orders should we truncate? Recall that by the orthogonality of the polynomial basis, $M_{ijk} = 0$ for

$$i + j < k, \qquad k + i < j, \qquad \text{or} \qquad j + k < i.$$

This implies that if the orders of u and f are limited to P, then all non-zero modes of λ of order up to $L = 2P$ are present in the system. For expansions of the diffusivity to these orders, it is possible that none of the operators (blocks) Δ_{ij} are zero.

Now, if λ is represented as a finite-order polynomial in ξ, its coefficients (in the standard basis $\{1, \xi, \xi^2, \ldots, \xi^n, \ldots\}$) will decay appropriately to ensure convergence. When this polynomial is expressed in the Legendre basis, we obtain a (finite) linear combination of the representation of each of the ξ^n. In this change of basis, the Legendre coefficients of each ξ^n decay as $\exp(-k^2)$. Therefore, the coefficients λ_k will also. Implications of this are that "relatively high" order Legendre expansions are unnecessary.

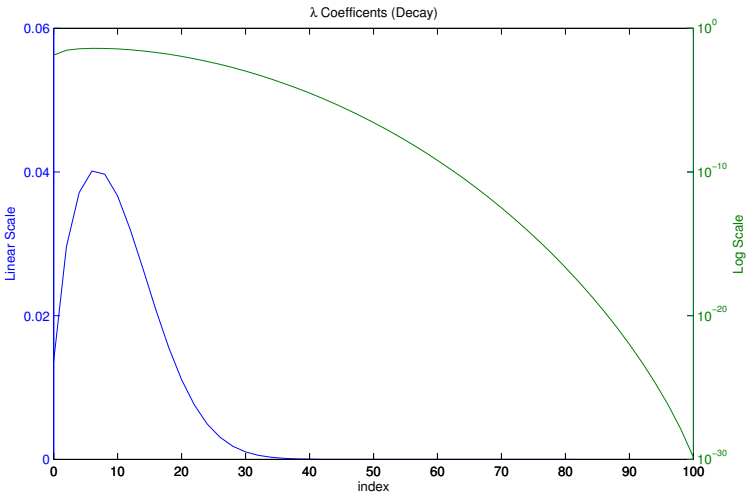


FIG. 5.1. *Plot of λ_k for $\lambda = \xi^{100}$.*

To ensure positivity of λ when the PC expansion employs unbounded random variables (such as Gaussian RVs) the polynomial's highest order term must be even, and λ_0 must be large enough to ensure the absolute minimum is positive. The constraints for the coefficients for a model utilizing bounded random variables (such as Uniform RVs) are a bit weaker: λ_0 need only be large enough.

6. Multigrid Methods. For modeling stochastic diffusion, the almost sure positivity of the diffusivity has been shown to be a strong requirement, eliminating linear-Gaussian models. This limits the utility in employing a KL expansion: in general the random variables

appearing are not independent, and so their joint distribution must be given in order to calculate the (structure) constants M_{ijk}.

Because of the positivity requirements, we prefer to expand the stochastic variables via the polynomial chaos for independent and identically distributed random variables. Computationally, bounded RVs are more natural, so we expand on the PC for uniform random variables. Furthermore, the lower-order modes in this expansion dominate significantly.

We are interested in solving the PC representation of the stochastic diffusion equation. We truncate these expansions to order P for u and f, to order L for λ, and use N random variables. If we discretize on a spatial grid of M points, then in our system

$$Du = f \tag{6.1}$$

the matrix D is symmetric, positive definite, and square of size

$$\frac{(P+N)!}{P!N!}(M-1). \tag{6.2}$$

Note that the size of this matrix does not depend on L, the expansion order of λ. L does, however, influence the sparsity of D. For even modest choices of P and N, this system becomes quite large.

For our initial computations, we will consider a Legendre PC with $L = 1$. In this case, the diagonal blocks are simply a diffusion operator with diffusivity λ_0, the expectation-mode, and D is quite sparse.

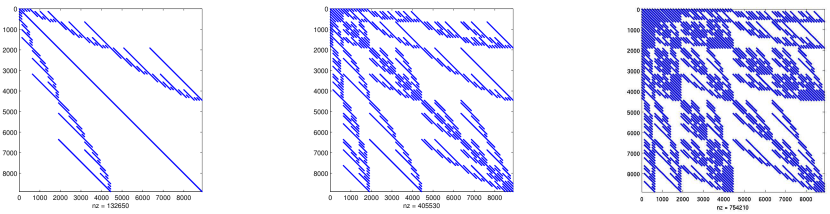


FIG. 6.1. *Non-zero structure of D for* $N = 4$, $P = 4$, *and (left to right)* $L = 1$, $L = 2$, *or* $L = 3$.

A multigrid method is employed to iteratively approximate solutions to large linear systems. The goal is to find a representative system which is much smaller, and in a sense interpolate its solution back to the original large system. Often several other systems, intermediate in size, are employed to transition from the large to the small system, and then back. The operators used to transfer between the various levels (or grids) are prolongators (or interpolators). On each level, an error-smoothing function (or relaxation method) is also employed. (For more details see e.g. [2].)

A multigrid method is fully specified once the prolongation matrices

$$\{I_{(1)}, I_{(2)}, \ldots, I_{(m)}\}$$

and the relaxation (or smoothing) function R are defined. An outline for this technique is given in the following algorithm, often called a V-cycle.

1.1 Relax n_1 times on $A_{(1)}v_{(1)} = f_{(1)}$ with initial guess $v_{(1)}$.
1.2 Compute the residual $r_{(1)} \leftarrow f_{(1)} - A_{(1)}v_{(1)}$.
1.3 Project the system into the next grid:

$$A_{(2)} \leftarrow I_{(1)}A_{(1)}I_{(1)}^T, \qquad v_{(2)} \leftarrow I_{(1)}^T v_{(1)}, \qquad f_{(2)} \leftarrow I_{(1)}^T r_{(1)}.$$

2.1 Relax n_2 times on $A_{(2)}v_{(2)} = f_{(2)}$ with initial guess 0.
2.2 Compute ...
2.3 Project ...
⋮
m.1 Solve exactly $A_{(m)}v_{(m)} = f_{(m)}$.
(m-1).4 Correct finer solution: $v_{(m-1)} \leftarrow v_{(m-1)} + I_{(m-1)}v_{(m)}$.
(m-1).5 Relax ν_{m-1} times on $A_{(m-1)}v_{(m-1)} = f_{(m-1)}$ with initial guess $v_{(m-1)}$.
(m-2).4 Correct ...
(m-2).5 Relax ...
⋮
1.5 Relax ν_1 times on $A_{(1)}v_{(1)} = f_{(1)}$ with initial guess $v_{(1)}$.
- Repeat this cycle until it converges.

What remains is to choose appropriate interpolation matrices and a good smoother.

6.1. Geometric Multigrid. Since our stochastic diffusion operator is a matrix of deterministic diffusion operators in the PC projection, a natural choice for each of the $I_{(k)}$ is a block diagonal structure which corresponds to the block structure of D (i.e., the D_{ij}). Each diagonal block in the prolongator linearly interpolates in the spatial dimension vectors corresponding to each u_k in (1.2). This effectively coarsens each D_{ij} in (1.3) via linear interpolation. The number of blocks, however, remains the same as one coarsens. When the stochastic system is positive definite, a Gauss-Seidel smoother acting on the entire matrix is sufficient. This kind of spatial coarsening is well understood. One difficulty may remain: at the coarsest geometric grid possible ($M = 2$), the system may still be quite large in a practical model, with moderate values for P and N.

A V-cycle multigrid method where $m = 7$ levels are used on a spatial grid of 128 uniform intervals. Iterations are terminated when the residual is reduced by 10^6. Notice that the number of iterations in Table 6.1 is constant over a variety of problems.

TABLE 6.1
Convergence Results for Geometric Multigrid

P	N	modes of u	size of D	iterations
3	10	286	36,322	6
3	12	455	57,785	6
5	10	3003	381,381	6
5	12	6188	785,867	6

6.2. Spectral Projections. In light of the fact that we are dealing with polynomial expansions, another natural way to "coarsen" the system is to consider lower-order expansions. These $I_{(k)}$ matrices would increase N or P, so that the $I_{(k)}^T$ effectively truncate the expansion of u to a smaller order, or reduce the number of random variables. This approach is also natural from the standpoint that the Gauss-Seidel smoother converges faster for high order modes and for random variables which enter the expansion via diffusivity modes $\lambda_k(x)$ which have high (spatial) frequencies.

While Gauss-Seidel converges for any symmetric, positive definite matrix, its smoothing properties in the stochastic direction are somewhat less understood. In general we have found that first coarsening spatially (e.g., until $m = 2$) and then coarsening in the RV dimensions is far more effective, from a multigrid convergence perspective, than conversely. One must

be careful in mixing spectral coarsening with the standard spatial geometric coarsening. In considering spatial grids which are more coarse, the Gauss-Seidel smoother is able to improve the convergence of lower frequency errors. If such spatial coarsening occurs after spectral coarsening, low frequency errors in modes of u that are eliminated will not be corrected.

Even in situations where the geometric-only Gauss-Seidel smoother converges quickly, a spectral projection will significantly reduce the size of the system, improving the ability of machines to handle larger systems.

6.3. Algebraic Multigrid. An algebraic multigrid (AMG) method could also be developed based along similar reasoning as the geometric multigrid algorithm. In particular, one can consider a block diagonal structure for the prolongator matrix. Each block diagonal of the prolongator is generated by applying a standard AMG method for deterministic equations. Specifically, the standard AMG algorithm is applied to each D_{ii}. These prolongators are then used to generate the block diagonal prolongator for the full system. Another possibility is to build a block prolongator where each block corresponds to piecewise-constant interpolation. Following the smoothed aggregation multigrid method [5, 6], this prolongator could then be improved via a Jacobi algorithm that is effectively applied to each column of the piecewise-constant block prolongator. One interesting aspect of this method is that the block Jacobi algorithm uses the entire matrix D (including the off-diagonal blocks) and so this prolongator uses information from both the diagonal and off-diagonal blocks of D. These algebraic multigrid variants will be explored in the future.

7. Conclusions. We have shown necessary and sufficient conditions on stochastic diffusivity in order for the diffusion operator to be positive definite. The a.s. positivity of λ may be violated in a truncation solution space.

For computational purposes, we project the KL expansion onto the Legendre PC chaos. Furthermore, we have investigated some basic matrix properties of these linear systems. In some cases we have illustrated the structure of these systems.

We then gave a multigrid method that is suitable for some of these systems. This method is primarily based on coarsening in the spatial direction. We also explored some coarsening in the stochastic directions and presented some preliminary findings.

While Gauss-Seidel converges for any symmetric, positive definite matrix, its smoothing properties in the stochastic direction are somewhat less understood. In general we have found that first coarsening spatially (e.g., until $m = 2$) and then coarsening in the RV dimensions is far more effective, from a multigrid convergence perspective, than conversely.

From here, we wish to understand the role of stochastic domains of small (or zero) diffusivity. Such considerations will have a strong influence on an AMG approach to solving these stochastic diffusion equations.

REFERENCES

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

[2] W. L. Briggs, V. E. Henson, and S. McCormick, *A Multigrid Tutorial, Second Edition*, SIAM, Philadelphia, 2000.

[3] H. Elman and D. Furnival, *Solving the stochastic steady-state diffusion problem using multigrid*, IMA J. Numer. Anal., 27 (2007), pp. 675–688.

[4] O. Le Maitre, O. Knio, B. Debusschere, H. Najm, and R. Ghanem, *A multigrid solver for two-dimensional stochastic diffusion equations*, Comput. Methods Appl. Mech. Engrg., 192 (2003), pp. 4723–4744.

[5] P. Vaněk, J. Mandel, and M. Brezina, *Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems*, Computing, 56 (1996), pp. 179–196.

[6] ———, *Convergence of algebraic multigrid based on smoothed aggregation*, Numerische Mathematik, 88 (2001), pp. 559–579.

Discrete Mathematics and Informatics

Discrete mathematics is the study of fundamentally discrete mathematical structures with application to problems arising in the computing sciences. Likewise, the field of informatics includes processing and reasoning about collected information or data — often to identify associations and to extract knowledge, with the objective to enable an informed decision–making process. Articles in this section encompass both of these disciplines with application to numerical linear algebra, data classification, and combinatorial optimization.

Wolf and Boman study how to best distribute sparse matrices among processors in order to reduce communication in parallel sparse matrix–vector multiplication. They propose improvements to a previously devised nested disection algorithm and show that the new approach outperforms the commonly used "fine–grain" method in many cases. *Gilpin and Dunlavy* explore the advantages of using heterogeneous ensemble classifiers in multi–class data classification. They show that heterogeneous ensembles, defined as sets of classifier models created using several types of classification algorithms, can lead to performance improvements over homogeneous ensembles. Additionally, they introduce the Hemlock software framework, which will be used in future classification studies. *Buluç and Boman* describe a novel attempt at scalable and robust multilevel partitioning of hypergraphs. Hypergraphs are generalizations of graphs in which the interaction between vertices can be beyond pairwise. They present two promising algorithms for hypergraph coarsening that are based on a new vertex aggregation paradigm. *Benavides et al.* describe SUCASA, the Solver Utility for Customization with Automatic Symbol Access. SUCASA is a mechanism for generating (integer) linear programming solvers derived from the parallel integer and combinatorial optimization package PICO that integrate algebraic modeling constructs. It allows application developers to access parameters, constraints, and variables from the application algebraic model within PICO, thereby enabling rapid development of problem–specific incumbent heuristics and cutting planes.

D. Ridzal
S.S. Collis

December 11, 2008

IMPROVEMENTS TO A NESTED DISSECTION APPROACH FOR PARTITIONING SPARSE MATRICES

MICHAEL M. WOLF* AND ERIK G. BOMAN†

Abstract. We consider how to distribute sparse matrices among processes to reduce communication in parallel sparse matrix-vector multiplication. In previous work, we introduced an exact graph model for 2d matrix partitioning and an algorithm based on nested dissection to solve this model. Our results indicated that our new approach was superior to traditional 1d partitioning and comparable to the fine-grain hypergraph method. We showed that our nested dissection method has two advantages over the fine-grain method: it was faster to compute, and the resulting distribution required fewer communication messages.

In this paper, we revisit our previous nested dissection partitioning algorithm and improve upon the previous algorithm. We solve a second, smaller partitioning problem to further reduce the communication volume. Our improved implementation greatly reduces the communication volume over our previous implementation for several matrices. With this improved implementation, our method seems to be superior to the fine-grain method for most structurally symmetric matrices and comparable for some of the nonsymmetric matrices.

1. Introduction. Sparse matrix-vector multiplication (SpMV) is a common kernel in many computations, e.g., iterative solvers for linear systems of equations and PageRank computation for ranking web pages. Often the same matrix is used for many iterations. An important combinatorial problem in parallel computing is how to distribute the matrix and the vectors among processes to minimize the communication. Such "communication" is also important on serial computers with deep memory hierarchies, where slow memory is much slower than fast memory. Since processor speeds increase much more rapidly than memory, we expect memory latency and bandwidth to grow in importance. Our present work is relevant to both parallel computation on distributed memory computers and serial computation on machines with hierarchical memory, but we phrase our work in the context of parallel computing.

Sparse matrix-vector multiplication $y = Ax$ is usually parallelized such that the process that owns element a_{ij} computes the contribution $a_{ij}x_j$. This is a local operation if x_j, y_i and a_{ij} all reside on the same process; otherwise communication is required. In general, the following four steps are performed [6, 20]:

1. **Expand:** Send entries x_j to processes with a nonzero a_{ij} for some i.
2. **Local multiply-add:** $y_i := y_i + a_{ij}x_j$.
3. **Fold:** Send partial y values to relevant processes.
4. **Sum:** Sum up the partial y values.

In this paper, we address sparse matrix-vector partitioning.

DEFINITION 1.1. Sparse matrix-vector partitioning: *Given a sparse matrix A, an integer $k > 1$, and $\epsilon > 0$, compute*

(i) a matrix partition $A = \sum_{i=1}^{k} A^i$ where each A^i contains a subset of the nonzeros of A, such that $nnz(A^i) \leq (1 + \epsilon)nnz(A)/k$, $\forall i$,

(ii) partitions of the input and output vectors,

such that when distributed across processes by these partitions, the communication volume in sparse matrix-vector multiply, $y = Ax$, is minimized.

This problem is NP-hard since it contains as a special case hypergraph partitioning. It has been observed [6, 3] that the matrix and vector partitioning problems can be separated. For

*Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, mmwolf@illinois.edu

†Scalable Algorithms Dept., Sandia National Laboratories, egboman@sandia.gov

any given matrix distribution (partition), it is fairly easy to find a "compatible" vector partition and these together give a solution to the combined matrix-vector problem. Additional objectives can be minimized in the vector partitioning phase [3, 4]. We focus on the matrix partitioning step but simultaneously obtain a compatible vector partitioning as well.

By far, the most common way to partition a sparse matrix among processors is to use a 1d scheme where each process is assigned the nonzeros for a set of rows or columns. This approach has two advantages, simplicity for the user and only one communication phase (not two). The simplest 1d method is to assign $\approx n/p$ consecutive rows (or columns) to each process, where n denotes the number of rows and p the number of processes (Figure 1.1). However, it is often possible to reduce the communication by partitioning the rows in a non-contiguous way, using graphs, bipartite graphs, or hypergraphs to model this problem (Subsections 2.1 - 2.3) [15, 6].

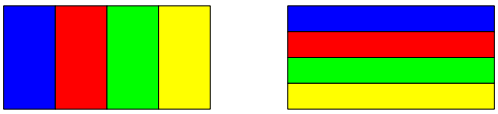


FIG. 1.1. *1d row and column partitioning of a matrix. Each color denotes a part that is assigned a different process.*

Recently, several 2d decompositions have been proposed [7, 8, 20]. The idea is to reduce the communication volume further by giving up the simplicity of the 1d structure. The fine-grain distribution [7] is of particular interest since it is the most general. We outline a graph model that also accurately describes communication in fine-grain distribution. In the symmetric case, this reduces to a standard graph. This led to a new graph-based "nested dissection partitioning algorithm" (Section 3). This nested dissection partitioning algorithm is related to previous nested dissection work for parallel Cholesky factorization [13, 14]. An important aspect to both our partitioning method and the previous parallel Cholesky factorization work is that communication is limited to separator vertices in the corresponding graph.

The rest of this paper is organized as follows. In Section 2 we discuss 1d and 2d data distribution. In Section 3, we review our nested dissection partitioning algorithm and our initial implementation of this algorithm. In particular, we discuss using this algorithm to partition structurally symmetric matrices in Subsection 3.1 and nonsymmetric matrices in Subsection 3.2. In Section 4, we outline new improvements to our original algorithm. Finally, we present numerical results in Section 5 that validate our general approach and our improved implementation.

2. Background: 1d and 2d Distributions.

2.1. 1d Graph Model. The standard graph model is limited to structurally symmetric matrices. In this case, the graph G is defined such that the adjacency matrix of G has the same nonzero pattern as the matrix A. Each row (or column) in A corresponds to a vertex in G. A partitioning of the vertices in G gives a partitioning of the rows (columns) in A. The standard objective is to minimize the number of cut edges, but this does not accurately reflect communication volume. Figure 2.1 illustrates this. Twice the number of cut edges (highlighted in magenta) yields a communication volume of 6 words, which overcounts the correct volume of 4 words. The problem is that vertices 1 and 8 are counted twice in the metric but each only contributes one word to the volume. The communication required is associated with the boundary vertices, so a better metric is to minimize the boundary vertices (4 words for Figure 2.1). This is an exact metric for bisection, while for $k > 2$ one should also take into account the number of adjacent processes.

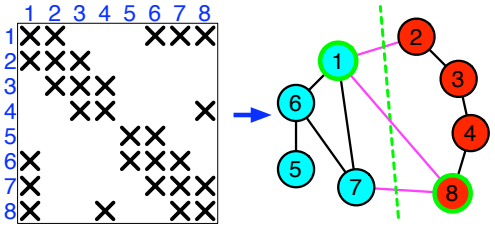


FIG. 2.1. *1d graph partitioning of matrix into two parts. Correct communication volume is 4 words. Communication of highlighted vertices is overcounted in edge metric.*

2.2. 1d Bipartite Graph Model. The graph model works poorly on nonsymmetric square matrices because they need be symmetrized, and does not apply to rectangular matrices. The bipartite graph model was designed to rectify this [15]. The bipartite graph $G = (R, C, E)$ is defined such that vertices R corresponds to rows, C corresponds to columns, and edges E correspond to nonzeros. The standard (simplest) objective is to partition both R and C such that the number of cut edges is minimized. Only one of the vertex partitionings (either R for rows, or C for columns) is used to obtain a 1d matrix partitioning. Again, the cut edges do not correctly count communication volume, and boundary vertices should be used instead.

2.3. 1d Hypergraph Model. Aykanat and Catalyurek introduced the hypergraph model to count communication volume accurately [6]. A hypergraph generalizes a graph. Whereas a graph edge contains exactly two vertices, a hyperedge can contain an arbitrary set of vertices [1, 2]. There are two 1d hypergraph models. In the row-net model, each column is a vertex and each row a hyperedge, while in the column-net model, each row is a vertex and each column a hyperedge. The objective is to find a balanced vertex partitioning and minimize the number of cut hyperedges. The communication volume is $\sum_i(\lambda_i - 1)$, where λ_i is the number of parts that touch hyperedge i. Finding the optimal balanced vertex partitioning is NP-hard but in practice good partitions can be found in polynomial time [6, 11].

2.4. 2d Matrix Distributions. Although the simplicity of 1d distributions can be advantageous, the communication volume can often be reduced by using 2d distributions. Figure 2.2 shows an example where 1d partitioning will always be poor. Consider the arrowhead matrix of dimension n, and bisection ($k = 2$). Due to a single dense row and column, any load balanced 1d partitioning will have a communication volume of approximately $(3/4)n$ words. The optimal volume is actually 2 words as demonstrated in the 2d partitioning of Figure 2.2 (right).

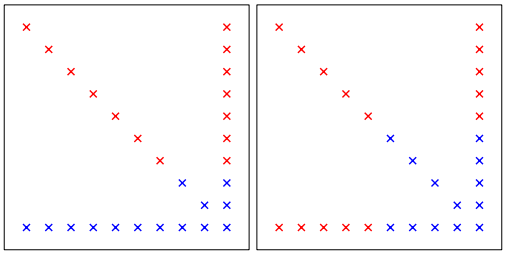


FIG. 2.2. *Arrowhead matrix with 1d (left) and 2d (right) distribution, for two processes (red and blue). The communication volumes in this example are eight and two words, respectively.*

2.5. Current 2d Methods. Two-dimensional partitioning is a more flexible alternative to one-dimensional partitioning. For dense matrices, it was realized that a 2d block (checkerboard) distribution reduces communication since communication is limited to process rows and columns. For sparse matrices, the situation is more complex. Several algorithms have been proposed to take advantage of the flexibility afforded by a two-dimensional partitioning but none have become dominant. Catalyurek and Aykanat proposed both a fine-grain [7] and a coarse grain [8] decomposition, while Bisseling and Vastenhouw later developed the Mondriaan method [20]. The coarse-grain method is similar to the 2d block decomposition in the dense case, but is difficult to compute and often gives relatively high communication volume. The Mondriaan method is based on recursive 1d hypergraph partitioning and thus is relatively fast but still produces partitionings with low communication cost.

The most flexible approach to matrix partitioning is to allow any nonzero to be assigned any part (process). This is the idea underlying the fine-grain method [7]. The authors propose a hypergraph model that exactly represents communication volume. In the fine-grain hypergraph model, each nonzero corresponds to a vertex, and each row and each column corresponds to a hyperedge. Catalyurek and Aykanat proved that this fine-grain hypergraph model yields a minimum volume partition when optimally solved [7]. As with the 1d hypergraph model, finding the optimal partition of the fine-grain model is NP-hard. This hypergraph can be partitioned into k approximately equal parts cutting few hyperedges using standard one-dimensional partitioning algorithms and software. This usually takes significantly longer than a one-dimensional partitioning of a typical matrix since the fine-grain hypergraph is larger than a 1d hypergraph model of the original matrix. In general, the partitioning of the fine-grain method with the one-dimensional algorithms produces good quality partitions. Thus, our goal in developing new two-dimensional methods is to produce similar quality partitions to fine-grain in a shorter runtime.

3. A Vertex Separator Partitioning Algorithm.

3.1. Symmetric Case. In this subsection we study structurally symmetric matrices, while the nonsymmetric case is analyzed in Subsection 3.2. First, we present an accurate graph model for communication volume in matrix-vector multiplication. We restrict our attention here to symmetric partitioning schemes, where a_{ij} and a_{ji} are assigned the same part. We partition both the vertices and edges, which distinguishes our approach from the 1d graph model and allows for 2d partitioning. A vertex incurs communication iff there are incident edges that belong to a different part. The volume depends on how many parts are represented among the incident edges.

THEOREM 3.1. *Let $G(V, E)$ be the graph of a symmetric sparse matrix. Let $E(v)$ denote the set of edges incident to vertex v. Let $\pi(v)$ and $\pi(e)$ denote the parts to which v and e belong, respectively. Then the communication volume in matrix-vector multiplication is* $2 \sum_{v \in V} (|\pi(v) \cup \pi(E(v))| - 1)$.

The factor two arises because any communication occurs in both phases (expand and fold). This exact graph model yields a minimum volume balanced partition for sparse symmetric matrix-vector multiplication when optimally solved.

Figure 3.1 shows an example of the exact graph model for 2d symmetric partitioning of matrices. The graph on the left corresponds to the symmetric matrix (partitioned on the right). The edges and vertices in the graph are partitioned into two parts (represented by cyan and red). The vertices that have incident edges belonging to a different part (and thus incur communication) are highlighted in green. Each contributes two words to the communication volume of the resulting matrix-vector multiplication. The matrix shows the 2d symmetric matrix partition obtained from the partitioned graph. The partition of the diagonal entries (as

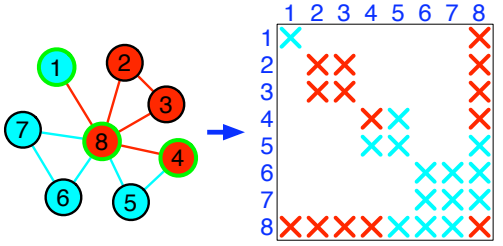


FIG. 3.1. *2d graph bisection for symmetrically partitioned matrix and resulting matrix partition. Part (color) of graph edge corresponds to symmetric pair of off-diagonal nonzeros.*

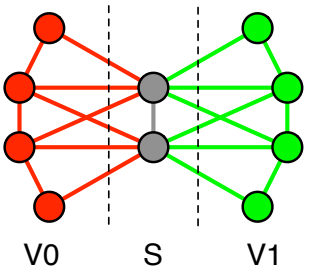


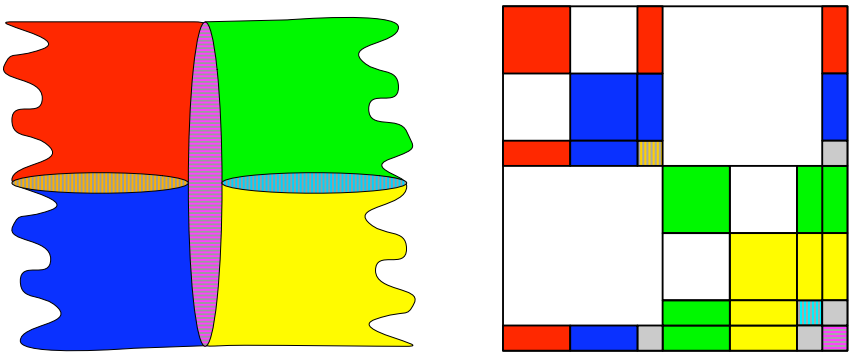
FIG. 3.2. *Bisection. Vertex separator (gray) partititions vertices into disjoint subsets* (V_0, V_1, S).

well as the vector entries) corresponds to the partition of the graph vertices. The partition of the off-diagonal entries corresponds to the partition of the edges in the graph.

If we solved this exact graph model optimally, we would obtain a balanced partition to minimize communication volume for resulting matrix-vector multiplication. However, this problem is NP-hard. In the next two subsubsections, we describe an algorithm for solving this exact graph model suboptimally in polynomial time (assuming the vertex separator is found in polynomial time). One constraint that we impose on our algorithm is that the vertex and edge partitions are *compatible*. A vertex partition is *compatible* with an edge partition if every vertex belongs to the same part as one of its incident edges. Similarly, an edge partition is *compatible* with a vertex partition if every edge belongs to the same part as one of its two vertices. There is no reason to violate this constraint since it will only increase the communication volume.

3.1.1. Bisection. For simplicity, we consider bisection first. In the next subsubsection we generalize to k-way partitioning for $k > 2$ using recursive bisection. First we compute a small balanced vertex separator S for the graph using any vertex separator algorithm. This partitions the vertices into three disjoint subsets (V_0, V_1, S). Let $E_j := \{e \in E | e \cap V_j \neq \emptyset\}$ for $j = 0, 1$, that is, E_j is the set of edges with at least one endpoint in V_j. V_j and E_j are assigned to part P_j for $j = 0, 1$. An example of a graph partitioned using this algorithm is shown in Figure 3.2.

The procedure above intentionally does not specify how to distribute the vertices in S and the edges therein. We showed in our previous paper [5] that if S is a separator such that each vertex in S has at least one non-separator neighbor in V_0 and one in V_1, the communication in SpMV is limited to the vertices in S, and the volume is $2|S|$. Furthermore, the assignment of vertices in S and edges therein does not matter as long as compatibility is maintained. There are several ways to exploit this flexibility, yielding several variations on our basic algorithm.



(a) Generic graph (b) Corresponding permuted matrix

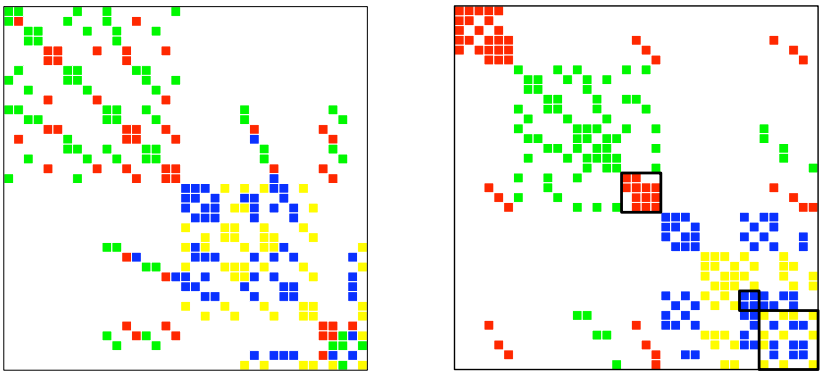
FIG. 3.3. *Graph and matrix partitioned using nested dissection method. Striped areas are separators and nonzeros corresponding to separators, respectively, where we have some flexibility in assignment. Gray blocks of nonzeros correspond to separator-separator edges in the the graph for which we also have flexibility in assignment. Matrix is reordered with nested dissection ordering for visualization purposes.*

3.1.2. Nested Dissection Partitioning Algorithm. In practice, one wishes to partition into $k > 2$ parts. If we knew how to compute a balanced k-separator, a set S such that the removal of S breaks G into k separate subgraphs, we could assign each subgraph to a different part and assign the vertices and edges in S based on one of the methods described above. However, we do not know efficient methods to compute a k-separator and do not consider this option any further. A more practical approach is to use recursive bisection. In fact, the procedure to compute a $k-$separator via recursive bisection is known as "nested dissection" and well studied [12, 18] since it is important for sparse matrix factorization.

The nested dissection algorithm is illustrated in Figure 3.3. In this example there are four parts (one for each process). We show the recursive procedure on a mesh, a generic graph, and the corresponding matrix. The striped and gray areas correspond to separators and separator-separator edges, respectively. We have not specified how to partition this data. It is important to note that it is not necessary to use nested dissection order to permute the matrix, as shown in Figure 3.3(b). We only do this to make the partitioning method more clear. Figure 3.4(a) shows the actual partitioning of the **cage5** matrix [10] with the corresponding nested dissection ordered partitioning in Figure 3.4(b) for easier visualization of our method.

In nested dissection algorithms, there is often a choice how to handle the separator at each level. Say V has been partitioned into V_0, V_1, and S, where S is the separator. The question is whether S should be included in the subproblems or not. In the original nested dissection by George [12] and also the generalized nested dissection method [18], it was included in the recursion, but in many implementations it is not. We have chosen not to include the separator vertices in the subproblems in the recursion since it simplified our implementation. For non-overlapping separators, the steps of our vertex separator graph partitioning algorithm are given in Algorithm 1.

The calculation of the vertex separators (line 1) gives us k disjoint subdomains divided by a hierarchy of $k - 1$ separators. The algorithm does not depend on any particular method for calculating the vertex separators, but smaller separators tend to yield lower communication volumes. Computing a minimal (balanced) vertex separator is NP-hard but existing heuristics can provide good (but suboptimal) separators in polynomial time. The most effective separator heuristics for large, irregular graphs are multilevel algorithms such as those implemented in METIS [17] and Scotch [9]. Lines 2 and 3 are fully expounded in Algorithm 1. However,



(a) Actual partitioning. (b) Permuted partitioning.

FIG. 3.4. *cage5 matrix partitioned using nested dissection. (a) shows how the matrix actually looks after being partitioned. (b) is a symmetric permutation of (a) for visualization purposes with separator blocks boxed.*

Algorithm 1: Nested Dissection Graph Partitioning

1 Compute vertex separators
2 Assign part P_i to vertices in V_i (V_i is set of vertices in the subdomain i)
3 Assign P_i to edges in E_i (E_i is set of edges that contain a vertex in V_i)
4 Assign parts to separator vertices
5 Assign parts to edges connecting vertices of the same separator
6 Assign parts to edges connecting vertices of two different separators

there are many different ways to assign parts in lines 4-6 and we leave this decision to the particular implementation.

3.1.3. Initial Implementation. In our initial implementation, we assigned each vertex in a given separator (line 4 in Algorithm 1) to a part in the range of parts belonging to one half of the subdomain. The half is chosen to keep the vertex partitioning as balanced as possible. We assigned each separator vertex (line 4) to the part of the first traversed neighbor vertex in the correct range of parts that had already been assigned a part. This greedy heuristic can be improved but had the advantage of being simple to implement and yielding better results than some more complicated heuristics. Edges connection vertices of the same separator (line 5) are assigned to the part of the lowered numbered vertex. We assigned edges connecting vertices from two different separators (line 6) to the part of the vertex of the lower-level separator. As with line 4, choices can most likely be made for lines 5-6 to further reduce communication volumes. Results for this initial implementation are given in Section 5 for this implementation. In Section 4, we attempt to improve upon the simplistic choices made for lines 4-6.

3.2. Nonsymmetric Case. Some modifications are required for the nonsymmetric case. We show how we can apply our nested dissection partitioning algorithm to bipartite graphs to partition nonsymmetric matrices. We generalize our symmetric communication (graph) model to the nonsymmetric case. This generalization is equivalent to a model recently proposed by Trifunovic [19]. We start with the bipartite graph $G = (R, C, E)$ of the matrix, where R and C correspond to rows and columns, respectively. In 1d distribution, we partition either the rows (R) or the columns (C). For fine-grain distribution, we partition both (R, C) and E

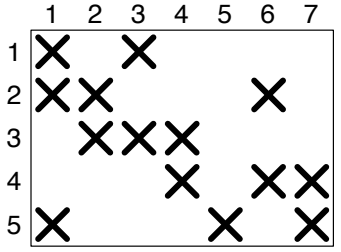


FIG. 3.5. *Rectangular matrix.*

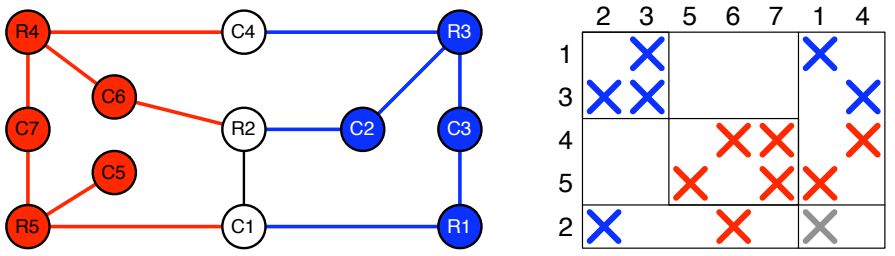
into k sets. Note that we explicitly partition the edges E, which distinguishes our approach from previous work. To balance computation and memory, our primary objective is to balance the edges (matrix nonzeros). Vertex balance is a secondary objective. Again, we assign communication cost to vertices such that a vertex incurs communication if and only if it has at least one incident edge in a different part. The communication volume will depend on the number of different parts to which these edges belong. Similar to the symmetric case, we have:

THEOREM 3.2. *Let $G(R, C, E)$ be the bipartite graph of a sparse matrix. Let $E(v)$ denote the set of edges incident to vertex v. Let $\pi(v)$ and $\pi(e)$ denote the parts to which v and e belong, respectively. Then the communication volume in matrix-vector multiplication is $\sum_{v \in R \cup C} (|\pi(v) \cup \pi(E(v))| - 1)$.*

In the bisection case, the volume is simply equal to the number of vertices that have at least one incident edge in a different part (boundary vertices). A crucial point is that by assigning edges to processors independent of the vertices, we can reduce the number of boundary vertices compared to the traditional 1d distribution, where only vertices are partitioned and the edge assignments are induced from the vertices.

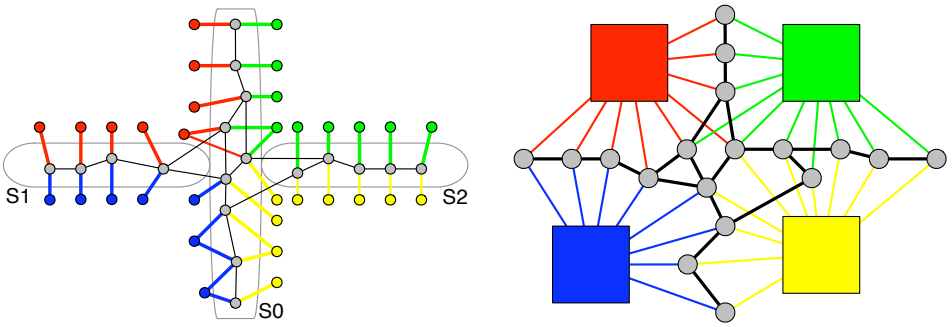
Once we have built the bipartite graph for our nonsymmetric matrix, we can apply our nested dissection algorithm directly to this bipartite graph to partition the matrix. This procedure is outlined in Figures 3.5 and 3.6. Figure 3.5 shows a nonsymmetric matrix. The corresponding bipartite graph is shown in Figure 3.6(a). This bipartite graph is partitioned using our nested dissection partitioning algorithm. The uncolored vertices are a vertex separator for this bipartite graph. They and the one separator-separator edge are left for the particular implementation to partition. Figure 3.6(b) shows the partitioned nonsymmetric matrix corresponding to the partitioned bipartite graph of Figure 3.6(a).

4. Improvements to Nested Dissection Partitioning Algorithm. In Section 3, we outlined a general nested dissection partitioning algorithm for sparse matrices (Algorithm 1). Steps 1-3 in this algorithm specified how to partition all the nonzeros in a matrix with the exception of the diagonal nonzeros corresponding to separator vertices in the graph and the off-diagonal nonzeros corresponding to edges connecting separator vertices in the graph (lines 4-6 in the algorithm). In Subsubsection 3.1.3, we outline the choices we made for lines 4-6 in our initial implementation. We showed in our previous work [5] that this initial implementation produced fairly good results, similar quality to the fine-grain method but requiring less runtime. The question we attempt to address in this section is whether we can improve our partitioning method, in particular improving lines 4-6 of the algorithm.



(a) Bipartite graph. (b) Partitioned matrix.

FIG. 3.6. *Bisection of bipartite graph and resulting partitioned/reordered nonsymmetric matrix. (b) is reordered for visualization purposes.*



(a) Separator connections to interior vertices. (b) Fixed vertices.

FIG. 4.1. *Partitioning model for separator vertices (gray) and edges connecting separator vertices (black). (a) shows the separator vertices (for 3 separators) connected to interior vertices of the different domains. In (b) these interior vertices are replaced by fixed vertices, represented by squares (one for each different part), that are preassigned to a part before the partitioning process.*

4.1. Partitioning Separator Vertices and Edges. In this subsection, we develop a model for improving the partitioning of the separator vertices and edges connecting separator vertices (lines 4-6 in Algorithm 1). We focus only on partitioning these separator vertices and edges, given the previous partitioning of the rest of the graph (lines 1-3 in Algorithm 1). This partitioning problem is shown in Figure 4.1 for $k = 4$ (3 separators). Figure 4.1(a) shows the separator vertices (unpartitioned, thus gray), which are connected to previously partitioned vertices in the subdomains (colored vertices) with previously partitioned edges. The edges (unpartitioned) connecting separator vertices are shown in black. We ignore the separator distinctions and partition the separator vertices and edges independent of the particular separator that each separator vertex belongs (Figure 4.1(b)). We replace the interior subdomain vertices with special fixed vertices (represented by the squares in Figure 4.1(b)), one for each part of the partition. The idea behind the fixed vertices is that it does not matter how many of the interior vertices of a given part a separator vertex is connected to in terms of communication, only that there is at least one such connection. The fixed vertices serve as a mechanism to account for the resulting communication when a separator vertex is assigned a part different from the part of a neighboring interior vertex.

4.2. Implementation for Partitioning Separator Vertices and Edges. We implement the partitioning of the nonzeros corresponding to the separator vertices and edges by using a symmetric version of the fine-grain hypergraph method (outlined in Subsection 2.5). The key

factor in our decision to use the relatively slow fine-grain hypergraph method is that we are only partitioning a small fraction of the original matrix (that corresponding to the separator vertices and edges). We use the aforementioned fixed vertices in this fine-grain hypergraph vertices and a partitioning method that allows us to preassign parts to these fixed vertices. The hope was that this partitioning of the separator vertices and edges would be superior to the heuristics (described in Subsubsection 3.1.3) that we used in our initial implementation.

5. Results. We compare the partitionings of different methods for a set of 11 sparse matrices. These matrices derived from different application areas (structural analysis, information retrieval, linear programming, circuit simulation, etc.), 9 (of 11) that were used and described in [20]. We perform separate experiments for symmetric and nonsymmetric matrices. We summarize the matrix properties in Tables 5.1 and 5.2. The first six matrices are symmetric (Table 5.1), and the final five are rectangular and thus nonsymmetric (Table 5.2). It is important to note that we treat explicit zeros in the sparse matrix storage as nonzeros so our number of nonzeros may differ slightly (but not significantly) from [10].

TABLE 5.1
Symmetric Matrix Info

Name	N	nnz	nnz/N	application
cage10	11,397	150,645	13.2	DNA electrophoresis
finan512	74,752	596,992	8.0	portfolio optimization
bcsstk32	44,609	2,014,701	45.2	structural engineering
bcsstk30	28,924	2,043,492	70.7	structural engineering
asic680ks	682,712	2,329,176	3.4	circuit simulation
pkustk04	55,590	4,218,660	75.9	structural engineering

TABLE 5.2
Nonsymmetric Rectangular Matrix Info

Name	rows	cols	nnz	application
dfl001	6,071	12,230	35,632	linear programming
cre_b	9,648	77,137	260,785	linear programming
tbdmatlab	19,859	5,979	430,171	information retrieval
nug30	52,260	379,350	1,567,800	linear programming
tbdlinux	112,757	20,167	2,157,675	information retrieval

In subsections 5.1 - 5.2, we compare the communication volume of the resulting parallel sparse matrix-vector multiplication for these matrix partitionings. We compare the implementations of our nested dissection algorithm (both the original and the improved implementations) with 1d hypergraph partitioning and fine-grain hypergraph partitioning. Though NP-hard problems, several good codes for graph and hypergraph partitioning are available, all based on the multilevel method. We used PaToH 3.0 [6] and Zoltan 3.0 [11] as our hypergraph partitioners. Metis and ParMetis are often used to find nested dissection orderings, but were not suitable for us because (i) Metis does not return the separators, and (ii) ParMetis runs only in parallel and quality deteriorates with increasing numbers of processes. Instead we derive our vertex separators from edge separators produced by hypergraph partitioning. This choice also enables a fair comparison across methods since the code base is the same.

5.1. Symmetric Matrices. We partition the 6 symmetric matrices shown in Table 5.1 using 1d, fine-grain, and the nested dissection methods of partitioning for 4, 16, 64, and 256

parts. We use the nested dissection implementations outlined in Sections 3 and 4 to partition the matrices directly. The average communication volumes are shown in Table 5.3. For 1d partitioning, we list the total communication volume. For the fine-grain and nested dissection methods, we list a scaled volume relative to the 1d volumes such that scaled volumes less than 1 indicate an improvement over the 1d method. We see that both our nested dissection implementations perform consistently better than 1d (scaled volumes less than 1). When compared to the fine-graph method, we see for most partitionings that the original nested dissection implementation yielded similar or better results for three of the six matrices. The original nested dissection only performed much worse for the **cage10** matrix. Another important point is that we previously showed the nested dissection method runtimes to be significantly lower than that of fine-grain [5], so in general we consider this a success. The improved nested dissection implementation consistently yielded better results than the original one. When compared to the fine-graph method, it yielded similar or better results for four of the six matrices and was always competitive.

TABLE 5.3

Average (20 runs) communication volume (in words) for k-way partitioning of symmetric matrices using different partitioning methods.

Name	k	1d total vol.	fine-grain scaled vol.	orig. ND scaled vol.	impr. ND scaled vol.
cage10	4	5379.0	**0.755**	0.822	0.757
	16	12874.5	**0.689**	0.887	0.716
	64	23463.3	**0.696**	0.980	0.723
	256	40830.9	**0.716**	1.030	0.742
finan512	4	295.7	0.883	0.775	**0.734**
	16	1216.7	0.844	0.770	**0.745**
	64	9986.0	0.864	0.807	**0.768**
	256	38985.4	0.679	0.770	**0.674**
bcsstk32	4	2111.9	**0.763**	0.840	0.833
	16	7893.1	**0.802**	0.861	0.836
	64	19905.4	0.938	0.910	**0.852**
	256	46399.0	1.002	0.944	**0.857**
bcsstk30	4	1794.4	1.079	0.781	**0.761**
	16	8624.7	1.133	0.827	**0.752**
	64	23308.0	1.102	0.902	**0.774**
	256	56100.4	1.031	0.982	**0.824**
asic680ks	4	3560.4	**0.509**	0.612	0.616
	16	9998.5	**0.463**	0.605	0.591
	64	21785.8	**0.439**	0.588	0.581
	256	38869.4	**0.492**	0.613	0.615
pkustk04	4	6610.8	0.626	0.526	**0.496**
	16	27565.4	**0.492**	0.602	0.553
	64	75329.7	**0.416**	0.623	0.513
	256	162105.5	**0.428**	0.558	0.451

5.2. Nonsymmetric Matrices. We partitioned the 5 nonsymmetric matrices shown in Table 5.2 using 1d column, 1d row, fine-grain, and the nested dissection methods of partitioning for 4, 16, 64, and 256 parts. However, in order to use the nested dissection partition methods with the nonsymmetric matrices, we have to form bipartite graphs as described in

Subsection 3.2. We can then apply the nested dissection implementations to partition the bipartite graph, which gives us a partitioning of the nonsymmetric matrix. In this subsection, we report the communication volumes of the matrix-vector multiplication resulting from these partitionings.

Table 5.4 shows communication volumes averaged over 20 runs for the 5 rectangular matrices from Table 5.2. The original nested dissection method results were consistently worse than the fine-grain results for these rectangular results and often worse than one of the 1d methods. Only for the **tbdlinux** matrix did the original nested dissection method yield significantly lower communication volumes than both 1d methods. However, we saw significant decrease in the communication volume for the new, improved nested dissection partitioning implementation when compared to the original implementation for both the **tbdmatlab** and the **tbdlinux** matrices. In fact, for these term-by-document matrices, the improved nested dissection method had a much lower communication volume when compared to the 1d methods and was slightly better than the fine-grain hypergraph method.

TABLE 5.4

*Average (20 runs) communication volume (in words) for k-way partitioning of rectangular nonsymmetric matrices using different partitioning methods. ** - for one run, PaToH hypergraph partitioner failed to produce a partition in 100 times the expected runtime, averaging 19 runs.*

Name	k	1d col. total vol.	1d row scal. vol.	fine-grain scal. vol.	orig. ND scal. vol.	impr. ND scal. vol.
dfl001	4	1388.4	2.141	**0.996**	1.181	1.150
	16	3575.5	1.631	**0.997**	1.155	1.097
	64	6040.2	1.391	**0.995**	1.119	1.077
	256	8897.0	1.377	**0.990**	1.097	1.069
cre_b	4	**1119.6**	29.194	1.027	2.312	2.283
	16	**3509.3**	15.970	1.011	1.848	1.803
	64	**7952.3**	9.315	1.024	1.605	1.608
	256	17077.8	6.048	**0.997**	1.409	1.405
tbdmatlab	4	14991.2	0.937	0.718	0.681	**0.560**
	16	40562.8	1.343	0.778	0.888	**0.716**
	64	81468.6	1.661	0.797	1.041	**0.774**
	256	144098.2	1.673	**0.757**	1.093	0.780
nug30	4	**56796.5**	4.746	1.100	1.307	1.295
	16	**115539.4**	3.320	1.157	1.507	1.499
	64	**199977.0**	2.674	1.172	1.530**	1.496**
	256	**307627.1**	2.090	1.166	1.494	1.455
tbdlinux	4	52021.1	0.813	0.471	0.429	**0.321**
	16	146980.9	1.136	0.565	0.594	**0.491**
	64	307829.8	1.449	0.610	0.733	**0.567**
	256	569152.5	1.600	0.611	0.854	**0.590**

6. Discussion/Conclusions. We presented improvements to our nested dissection method for partitioning sparse matrices. Our previous implementation produced partitions of similar quality to the fine-grain method for symmetric matrices at a great reduction in the runtime. Our improved implementation has further reduced the communication volume so that we argue that it is the best of the four methods for our set of symmetric matrices. For the rectangular nonsymmetric matrices, our original nested dissection algorithm seems to perform rather poorly compared to both 1d and fine-grain. We speculate that this may be a weak-

ness in our method of calculating the vertex separators and may be improved by a different method such as ParMetis [16] or PT-Scotch [9]. Again, our new implementation of the nested dissection partitioning algorithm yielded improved results for these nonsymmetric matrices but not significantly for all the matrices. However, it is interesting to note that our new implementation was the best of the five methods for the two term-by-document matrices. We plan to further study the partitioning of these these types of matrices since they are extremely important in information retrieval. We believe our algorithm can be efficiently implemented in parallel since the core is nested dissection (by vertex separators), so existing software like ParMetis [16] or PT-Scotch [9] may be used. We plan to study parallel performance on larger problems in future work.

Acknowledgments. We wish to thank Rob Bisseling, Umit Catalyurek, Michael Heath, and Bruce Hendrickson for helpful discussions. We thank Florin Dobrian for providing a matching code used to produce vertex separators. This work was funded by the US Dept. of Energy's Office of Science through the CSCAPES Institute and the SciDAC program.

REFERENCES

[1] C. Berge, *Graphs and Hypergraphs*, vol. 6 of North-Holland Mathematical Library, Elsevier Science Publishing Company, 1973.

[2] ———, *Hypergraphs: Combinatorics of Finite Sets*, vol. 45 of North-Holland Mathematical Library, Elsevier Science Publishing Company, 1989.

[3] R. H. Bisseling, *Parallel Scientific Computing: A structured approach using BSP and MPI*, Oxford University Press, 2004.

[4] R. H. Bisseling and W. Meesen, *Communication balancing in parallel sparse matrix-vector multiplication*, Electronic Transactions on Numerical Analysis, 21 (2005), pp. 47–65.

[5] E. Boman and M. Wolf, *A nested dissection approach to sparse matrix partitioning for parallel computations*. Submitted, 2008.

[6] Ü. Çatalyürek and C. Aykanat, *Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication*, IEEE Trans. Parallel Dist. Systems, 10 (1999), pp. 673–693.

[7] ———, *A fine-grain hypergraph model for 2d decomposition of sparse matrices*, in Proc. IPDPS 8th Int'l Workshop on Solving Irregularly Structured Problems in Parallel (Irregular 2001), April 2001.

[8] ———, *A hypergraph-partitioning approach for coarse-grain decomposition*, in Proc. Supercomputing 2001, ACM, 2001.

[9] C. Chevalier and F. Pellegrini, *PT-SCOTCH: A tool for efficient parallel graph ordering*, Parallel Computing, 34 (2007), pp. 318–331.

[10] T. A. Davis. The University of Florida Sparse Matrix Collection, 1994. Matrices found at http://www.cise.ufl.edu/research/sparse/matrices/.

[11] K. Devine, E. Boman, R. Heaphy, B. Hendrickson, and C. Vaughan, *Zoltan data management services for parallel dynamic applications*, Computing in Science and Engineering, 4 (2002), pp. 90–97.

[12] A. George, *Nested dissection of a regular finite-element mesh*, SIAM Journal on Numerical Analysis, 10 (1973), pp. 345–363.

[13] A. George, M. T. Heath, J. W.-H. Liu, and E. G.-Y. Ng, *Solution of sparse positive definite systems on a hypercube*, Journal of Computational and Applied Mathematics, 27 (1989), pp. 129–156. Also available as Technical Report ORNL/TM-10865, Oak Ridge National Laboratory, Oak Ridge, TN, 1988.

[14] A. George, J. W.-H. Liu, and E. G.-Y. Ng, *Communication results for parallel sparse Cholesky factorization on a hypercube*, Parallel Computing, 10 (May 1989), pp. 287–298.

[15] B. Hendrickson and T. G. Kolda, *Partitioning rectangular and structurally nonsymmetric sparse matrices for parallel computation*, SIAM J. Scientific Computing, 21 (2000), pp. 2048–2072.

[16] G. Karypis and V. Kumar, *Parmetis: Parallel graph partitioning and sparse matrix ordering library*, Tech. Rep. 97-060, Dept. Computer Science, University of Minnesota, 1997.

[17] ———, *METIS 4.0: Unstructured graph partitioning and sparse matrix ordering system*, tech. rep., Dept. Computer Science, University of Minnesota, 1998.

[18] R. J. Lipton, D. J. Rose, and R. E. Tarjan, *Generalized nested dissection*, SIAM Journal on Numerical Ananlysis, 16 (1979), pp. 346–358.

[19] A. Trifunovic and W. J. Knottenbelt, *A general graph model for representing exact communication volume in parallel sparse matrix-vector multiplication*, in Proc. of 21st International Symposium on Computer and Information Sciences (ISCIS 2006), 2006, pp. 813–824.

[20] B. Vastenhouw and R. H. Bisseling, *A two-dimensional data distribution method for parallel sparse matrix-vector multiplication*, SIAM Review, 47 (2005), pp. 67–95.

HETEROGENEOUS ENSEMBLE CLASSIFICATION

SEAN A. GILPIN* AND DANIEL M. DUNLAVY†

Abstract. The problem of multi-class classification is explored using heterogeneous ensemble classifiers. Heterogeneous ensembles classifiers are defined as ensembles, or sets, of classifier models created using more than one type of classification algorithm. For example, the outputs of decision tree classifiers could be combined with the outputs of support vector machines (SVM) to create a heterogeneous ensemble. We explore how, when, and why heterogeneous ensembles should be used over other classification methods. Specifically we look into the use of bagging and different fusion methods for heterogeneous and homogeneous ensembles. We also introduce the HEMLOCK framework, a software tool for creating and testing heterogeneous ensembles.

1. Introduction. The problem of data classification, or data labeling, arises in a wide variety of applications. Examples include detecting spam e-mail messages based on the content of the messages (document classification), labeling cells and tumors as malignant or benign based on the context of MRI scan data (image classification), and identification of individuals based on fingerprints, facial features, and iris patterns (biometric identification). In all of these examples, the goal is to predict a discrete label (e.g., "spam" versus "not spam") for a particular data instance (e.g., a particular e-mail message) based on the attributes of that instance.

More formally, classification is the task of learning a function, f, that maps a set of data instance attributes, $\mathbf{x} = \langle a_1(\mathbf{x}), \ldots, a_m(\mathbf{x}) \rangle$, to one of several predefined class labels, $\mathcal{Y} = \{y_1, \ldots, y_k\}$. When a data instance can be deduced easily from the context, attribute j of that instances will be denoted simply as a_j. The function f is often called a classifier, classifier model, or hypothesis. The set of data instances used to learn, or train, a classifier model is called the training set and is denoted $\mathcal{D}_{tr} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, where n is the number of instances, $\mathbf{x} \in \mathbb{R}^m$ is a vector of attributes, or features, for data instance i, and y_i is the label for data instance i. In order to validate the models learned, it is common practice to select some of the training data to be used in testing the resulting classifier models. This testing, or validation data, is denoted $\mathcal{D}_{te}$ and is not used in training the classifier model. Throughout this paper, we assume that all labels given for the training and testing data are correct—i.e., there are no mislabeled instances.

Recent results in solving classification problems indicate that the use of ensembles, or sets of classifier models, often leads to improved performance over using single classifier models [3, 4, 5, 24]. Much of the previous work on ensembles of classifier models (see e.g., [7]) has focused on *homogeneous ensemble classifiers*—i.e., collections of classifier models of a single type. In this work, we focus on *heterogeneous ensemble classifiers*, where the collection of classifiers are not of the same type. Note that such classifier models are also referred to as hybrid ensemble classifiers. Our goal is to find when and how the use of heterogeneous ensembles can be advantageous.

The motivation for our current work stems from previous work in classifying text documents [4]. The problems in that domain sometimes involve two classes (e.g., "spam" versus "not spam" in the e-mail classification problem), but more generally involve more than two classes (e.g., mapping scientific articles to appropriate journals for publication). Thus, we focus on the general problem of multi-class classification in this paper (i.e., $k >= 2$). We are also interested in incorporating data with missing attributes or with both continuous and discrete attributes into our models, as such data often arises in text document classification problems (e.g., documents do not contain all terms [i.e., features] and documents can contain

*Computer Science, San Jose State University, sgilpin80@gmail.com
†Computer Science and Informatics, Sandia National Laboratories, dmdunla@sandia.gov

both continuous features [via vector space models] and discrete features [dates, publication names, etc.]).

As part of this work, we have created a software framework called HEMLOCK (Heterogeneous Ensemble Machine Learning Open Classification Kit) for creating and evaluating heterogeneous ensemble classifiers. Although the methods described in this paper for classifier models, ensemble creation, and classifier validation/performance are applicable to the problem of classification in general, the majority of the focus is on those methods currently available in HEMLOCK.

2. Methods. In this section, we describe the methods used in the HEMLOCK software package to generate classifier models. Currently, HEMLOCK interfaces a software library called WEKA [25] for all of its classification methods. These methods formulate models that include mathematical descriptions of decision boundaries—i.e., hyperplanes, piecewise hyperplanes, or nonlinear manifolds that partition the feature vector space induced from a given training set of data. Combined with decision rules particular to each method, these decision boundaries are used to determine which class labels are associated with different areas of the feature space. Note that some methods generate explicit representations of the decision boundaries via parameters of some explicit function (e.g., support vector machines), whereas others generate implicit boundaries (e.g., nearest neighbor classifiers). Appendix A has information about the methods used from WEKA. The rest of this section focuses on the ensemble methods implemented in HEMLOCK. Throughout this section, "data" refers to "training data" unless otherwise indicated.

2.1. Ensemble Classifiers. Ensemble classifiers are a type of meta-model that use a set of base classifiers as input to a combination function. The combination function is intended to make the best use of the information provided from the base classifiers in order to make class label predictions as accurately as possible. These ensembles are homogeneous, referring to the fact that all of the base classifiers are of a single type (e.g., decision trees), differing by model parameters, the data used for training, or a combination of the two.

Ensemble classifiers have been found to be generally more accurate than non-ensemble classifiers. Following are different situations in which using an ensemble classifier model over a single classifier model have led to improved classifier performance in practice [11].

- Base classifiers may not be able to model the true class decision boundaries exactly. For example, a linear model can never exactly represent a quadratic decision boundary. However an ensemble with linear models as base classifiers will in general lead to more flexible decision boundaries than those of the simple, underlying linear models.
- A lack of data can lead to many good estimations of the true class boundaries. Instead of choosing one, an ensemble can use all of them as base classifiers to eliminate the chance of picking the worst classifier.
- Globally optimal searches of the classifier function space are not computationally feasible for large sets of data or data instances with large numbers of features. Most classification algorithms are therefore limited to searches that lead to locally optimal parametrization. In this case, a combination of models may better approximate globally optimal estimation of true decision boundaries by employing base classifiers that search different regions of the global classifier parameter space.
- Noise in the training data can be addressed by combining models that are trained using data sampled from the entire training data set. Combinations of models trained is such a way often reduce overfitting the data as well.

There are two general types of ensemble combination functions: fusion and selection. In fusion functions, the output from each of the base classifiers includes a weight on every

prediction made by the ensemble. A typical fusion function is the sum of the weighted predictions. Selection functions, on the other hand, allow the use of outputs from one or more of the base classifiers, not necessarily making use of the outputs of all of the predictions. Moreover, the base classifier output used in the selection function typically depends on the characteristics of the instance whose class is being predicted. For example, a selection function could be one that chooses the base classifier that performs best on the training data instances that are most like the testing data instance being considered for classification. We currently consider fusion methods only throughout the remainder of this paper.

The combinations functions we consider take as input one of three different types of output from the base classifiers.

- *Labels.* Functions that make use of the most likely class to which an instance belongs.
- *Label rankings.* Functions that make use of ranked lists of class label predictions (e.g., sorted by likelihood or probability that the instance belongs to the class).
- *Measurements.* Functions that make use of vectors of length k (i.e., the number of classes), where element j corresponds to a measurement of an instance associated with class j. These measurements are often intrinsic values computed within each classification method. In this work, the measurements are the probabilities that an instance belongs to a particular class.

A variety of fusion methods have been developed. We discuss several here that are used in the experiments described in Section 4. An example of a fusion function that uses label outputs is the majority voting function [19]. In majority voting, the labels output from each base classifier are used as votes on the predicted class, and the class with the most votes becomes the predicted class of the ensemble classifier. The sum rule is an example of a fusion function that uses measurement outputs from the base classifiers [18]. The sum rule first sums the measurement output vectors from the base classifiers for a given test instance, and then chooses the class corresponding to the largest sum as the predicted class. The linear combination rule is identical to the sum rule except that each of the base classifiers is assigned a weight, which is used to scale the measurement vectors before summing [26]. In our experiments, we determined the weights using the least squares method to maximize training accuracy. In stacking, the output vectors are treated as input to a new classification problem, where a classification algorithm, such as the SVM classification algorithm, builds a model to act as the fusion function. We have also been alerted, by a referee, to an ensemble method that is implemented inside of WEKA, called Ensemble Selection[8]. In the future we would like to explore this method and compare our results with it.

The process of creating ensemble classifier models involves two major steps. First, the base classifiers are created during a generation phase, and then the models are combined during a combination phase. The goal of the generation phase is to create a diverse, accurate set of base classifiers. A recent survey on several diversity measures [5] illustrates the challenges associated with creating a diverse set of base classifiers, and accuracy will be discussed in Section 3.1. Examples of methods for generating diverse base classifiers include sampling the training data (e.g., bagging [6]) and sampling the feature space (e.g., the random subspace method [15]). Another example method, the method of random forests [7], combines these two sampling strategies.

For some homogeneous ensemble models (e.g., models using a linear combination fusion function), several model parameters need to be learned or fit. Typically, the training of the ensemble model parameters is performed using the training data set. However, there are issues with such an approach [13], and more work in this area is required to better understand the implications of such a training strategy. Such work is beyond the scope of this paper, but

will be pursued in future work.

2.2. Heterogeneous Ensemble Classifiers. A heterogeneous ensemble is an ensemble with a set of base classifiers that consist of models created using different algorithms. The same combination functions that are used to create homogeneous ensembles can be used to create heterogeneous ensembles. The main difference lies in the methods used for creating the set of base classifier. The methods available for creating base classifier sets for homogeneous ensembles are modified so that models built from different classification algorithms can be combined to form a set of base classifiers. Currently, there is no clear choice on how to combine these base classifiers most effectively. Furthermore, there are open questions regarding which base classifiers to use and how they should be combined for optimal performance.

Motivation. Using different types of base classifiers leads to diversity in the same way that changing model parameters can in creating homogeneous ensemble classifiers. Different base classifier types can have different internal representations and may be biased in different ways. This leads to classifiers that will disagree with each other to some extent over a set of data instances covering a wide range of the feature space. This disagreement between the base classifiers is essential for the success of an ensemble classifier and is what we refer to as diversity. Without diversity in the base classifier models, there is no point in using an ensemble, as the output of the ensemble classifier will be identical to the output of each of the base classifiers. On the other hand, we would also like the base classifiers to be as accurate as possible. We do not want to force diversity in such a way that we end up with base classifiers that have too much error or that do not generalize well. Using a heterogeneous set of base classifiers, then, is a way to introduce diversity while keeping accuracy high.

Diversity. Table 2.1 illustrates how different base classifier algorithms can lead to diversity in an ensemble. These classifiers were trained using the same data set. They were then tested using a testing data set and their outputs were recorded. The differences in their outputs represent the extent to which they "disagree" about the probability distributions for the test instances.

	Naive Bayes		*Decision Tree*	
Instance	*Measurement*	*Label*	*Measurement*	*Label*
1	[0.99, 0.01]	1	[0.87, 0.13]	1
2	[0.00, 1.00]	2	[0.18, 0.82]	2
3	[0.04, 0.96]	2	[0.18, 0.82]	2
4	[0.01, 0.99]	2	[0.87, 0.13]	1
5	[0.00, 1.00]	2	[0.18, 0.82]	2
6	[0.00, 1.00]	2	[0.05, 0.95]	2
7	[0.99, 0.01]	1	[0.87, 0.13]	1
8	[1.00, 0.00]	1	[0.05, 0.95]	2

TABLE 2.1

Measurement outputs from two classifiers trained on the same data. The differences in these outputs for the same instances is what we refer to as diversity and is essential for creating ensemble classifiers. In this example the diversity is introduced by using different classification algorithms.

2.2.1. HEMLOCK. HEMLOCK has been designed to create and test heterogeneous ensemble models. As such it contains methods for creating base classifiers, applying fusion functions, and evaluating classifier models. HEMLOCK currently uses interfaces to WEKA classification algorithms for creating base classifiers. Input and output is passed to and from HEMLOCK via XML. The input, or experiment, files contain information about which models

to use, model parameters, evaluation methods, and data sets to be used in an experiment. Ranges of model parameters can be specified as well, leading to collections of experiments, each corresponding to a particular set of parameters allowed. Currently only full factorial experiments for a given set of parameter values (i.e., sets of experiments where all possible combinations of the given parameters values are tested) are supported. Several standard evaluation methods have been implemented in HEMLOCK as well: e.g., stratified k-fold cross validation, ROC curve generation, and generalization error calculations. Creation of base classifiers using a particular set of parameters or via random sampling can be specified as well. HEMLOCK currently includes the fusion methods of majority voting, the sum rule, and a linear combination rule (where the weights are computed using linear l_2 regression).

3. Evaluation of classification algorithms. The evaluation of performance or accuracy of a classification algorithm is not necessarily straightforward. Real world problems often have different sets of (potentially conflicting or competing) requirements. Hence, an algorithm (or more precisely, a particular parametrized instance of an algorithm) that may work well in solving one classification problem may perform poorly on other problems. Some issues that motivate different approaches to classifier evaluation include model interpretability, predictive ability of models created (including accuracy and data overfitting), and computational time/effort required during both the training and application of a model.

3.1. Evaluation Considerations.

Interpretability. Sometimes the models from classification algorithms need to be interpretable by a human being, e.g., for explanatory analysis of classes in addition to prediction. It may be that the user wants to ensure the sanity of the model, or it may be that he or she wants to learn something about the underlying classes by studying the models. Some classification algorithms create models that are easy and natural for humans to interpret. Ensembles models, however, are generally not easy to interpret, even when the base classifiers individually are easy to interpret.

Accuracy. Generalization error is a measure of how well a model predicts the classes for a set of instances it has never seen before (i.e., a testing data set). Training error is a measure of how well a model predicts classes for the same set of instances that were used to train the model. Training error is a very optimistic estimation of the true model error over the entire instance space, whereas generalization error is more pessimistic and a less biased estimate of the true error. Accuracy then usually refers to the complement of the generalization error.

Overfitting. One of the reasons that training error is regarded as optimistic is based on data overfitting. Training error does not incorporate the extent a classification model has overfit the data. If a model is too specific to the training data and does not generalize well to the entire instance space, then it has overfit the training data. There are various strategies that classification algorithms can employ to ensure that resulting models both perform well on the training data and generalize to unseen training instances. For example, many decision tree classification algorithms contain methods for pruning of tress to avoid singleton leaf nodes, which typically do not generalize well.

Computational Time. There are two considerations associated with the computational running time of a classification algorithm. One is the amount of time it takes to build the models, called training time. The other is the time it takes to predict the class of an instance, called prediction time. Some algorithms require very little time to train but more time to predict, and some vice versa. For example, artificial neural networks [22] (not currently part of HEMLOCK) can take a very long time to train, but the models they produce are a simple linear combination of the features, leading to constant prediction time complexity. On the other hand, a nearest neighbor classifier spends no time on training, but during classification

the training data set must be searched to find the nearest neighbors for each testing instance, which can be computationally expensive. A less extreme example are decision trees, which for many methods can be trained in $O(mn \log n)$ time and tested in $O(\log_s n)$ time, where s is the minimum number of splits allowed over all the attributes (e.g., $s = 2$ in the case of binary decision trees induced from data with only continuous attributes) [12]. Another consideration that impacts computational time is how much and to what extent the training and testing methods of a classification algorithm parallelizes. In general, ensemble methods can be parallelized well because the ensemble members can be trained in parallel, and the ensemble predictions can be executed in parallel as well.

3.2. Evaluation Measures. The following descriptions of evaluation measures assume we are solving a two-class classification problem and that the true class labels are known for the testing data instances. All of the following concepts can be generalized to multi-class problems but we do not, in this exposition, for the sake of clarity. When dealing with two-class problems the class labels are "positive" (typically referring to the class in which we are most interested in classifying) and "negative". Table 3.1 lists definitions used throughout this section.

Value	*Symbol*	*True Label*	*Predicted Label*
True positives	TP	positive	positive
True negatives	TN	negative	negative
False positives	FP	negative	positive
False negatives	FN	positive	negative

TABLE 3.1
Definitions used in measures for evaluating two-class classifier models.

Confusion Matrices. A confusion matrix can be created from a model f and a set of instances with known true class labels D_{te}, and reflects how well a classifier correctly classifies those instances per class. An example confusion matrix for the two-class classification problem is as follows.

		Predicted Class	
		Positive	Negative
True Class	Positive	*TP*	*FN*
	Negative	*FP*	*TN*

Rows of the matrix are associated with true labels, and columns represent the predicted labels. Hence, the sum of the row values equals the number of instances in D_{te} that have the true class corresponding to that row. Similarly the sum of the column values equals the number of instances in D_{te} where $f(x)$ has predicted the class corresponding to that column. It is easy to see that elements on the diagonal contain the counts of instances that the model correctly labeled for each class. With a two class classification problem, you always get a 2×2 confusion matrix where the values in the matrix correspond to TP, FN, FP, TN from left to right and top to bottom. TP and TN are on the diagonal entries and correspond to correctly classified instances.

Accuracy. We denote the accuracy of a classifier model, f, as

$$A(f) = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} . \tag{3.1}$$

As mentioned above, it is important to realize the importance of choosing a set of instances for calculating accuracy. Accuracy can be calculated using any set of instances, but a data set

containing instances different than those used in training the model begin evaluated will be less biased.

Receiver Operator Characteristic (ROC) Curve. ROC curves depict the potential of a model for correct classification in two-class problems. To generate an ROC curve for a classifier model, the set of continuous outputs relative to the positive class—i.e., the values $f(\mathbf{x}), \mathbf{x} \in \mathcal{D}_{te}$ are first sorted in descending order. The ROC curve is then a plot of the true positive rate, $TP/(TP+FN)$, as a function of the false positive rate, $FP/(FP+TN)$, computed using each of the sorted outputs as a cutoff threshold for labeling instances as positive. The true positive rate is also called the *sensitivity* of a classifier model, and the false positive rate is the complement of the *specificity* of a classifier model. Thus, an ROC curve is sometimes referred to as a plot of sensitivity versus (1 - specificity).

ROC analysis can also be used to compare models built for multi-class classification problems. In the simplest case, each model will correspond to one point on the graph. One can then easily compare different properties of the models based on their placement on the graph. [14]

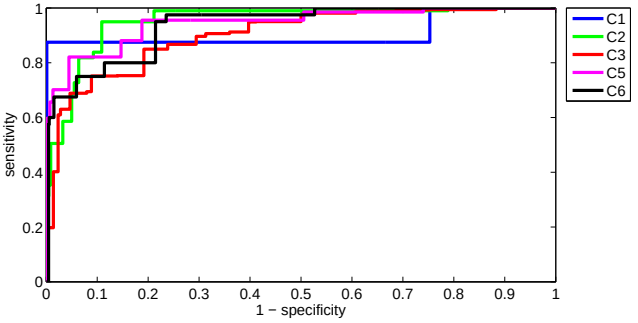


FIG. 3.1. *Example ROC curves for random tree classifiers trained on the "anneal" data set (6 classes, 5 classes represented in test data). The different curves correspond to ROC curves for the two-class problems of one class versus the rest.*

Area Under the Curve (AUC). The AUC measurement corresponds to the ROC analysis for two-class classification models. The area under the ROC curve described above can be calculated to give a scalar representation of the most important aspect of the plot: the potential of the classifier to perform well in separating regions of feature space by class.

3.3. Validation Methods. Methods for validating the performance of classifier models typically employ a scheme for choosing training and testing data sets and a process for determining the significance of the results [19, 23].

Holdout. The simplest method to validate a classifier model is called the holdout method and simply involves separating a given set of instances with known labels into two sets, $\mathcal{D}_{tr}$ and $\mathcal{D}_{te}$. The first set is used to train the model and the second set is used to test the model. This method is not often used because it is difficult to determine the significance of the tests results since it is based on a single split of the data only. However, the method is quick and does indicate a rough estimate of performance.

Stratified k-Fold Cross Validation. The method of k-fold cross validation is used to compute generalization error and determine the significance of the testing results. The methods starts by first dividing the training data into k partitions, or folds, where each fold contains instances with approximately equal class distribution (i.e., are stratified). If the folds are not stratified, the test results may poorly approximate the generalization error. Each of the

folds will be used as a testing set for exactly one model trained using the remaining folds as training data. Once the models are created, they are tested and the results are averaged to determine generalization error and then analyzed statistically to determine the significance of the results. A common value used for k is 10, but their may be advantages for using other values depending on the problem at hand. One disadvantage of using large values of k is that a large number of models will need to be created and tested; this may not be practical due to the computational time required.

Other related methods are the leave-one-out method and 5×2-fold cross validation. In the leave-one-out method, each of the instances is used individually as a testing set with the remaining instances used as the training set. With n instances, this is equivalent to n-fold cross validation. 5×2-fold cross validation consists of 2-fold cross validation performed five times, with the results being averaged across the five runs [10].

Bootstrapping (Random Fold) Validation. Bootstrapping is similar to k-fold cross validation in that different models are created and tested, with the performance results computed as averages over all the models. The training sets are created by randomly selecting instances with replacement. All of the instances that are not chosen for the training set are used as the test set. Because the instances are drawn with replacement, a larger training set can be created than with k-fold cross validation. However, many of the instances in the training set may be duplicates. The test results of this method should be meaningful, as the class distributions of the test set will likely match the class distribution of the original set of instances and the training set because of the random way the set is chosen. This method can be useful when only a small number of instances with known labels are available.

4. Numerical Experiments. In this section we describe the experiments we performed on base classifiers, homogeneous ensemble classifiers, and heterogeneous ensemble classifiers.

4.1. Data. The data we used in our experiments is the data from [3]. We specifically only used data that had no unknown attributes. See Table B.1 for a summary of the data sets used in the experiments.

4.2. Experiments. First we performed parameter sweeps for the base classifier algorithms to find sets of parameters that performed well. We measured the performance by using 2-fold cross validation to calculate accuracy and then taking the average of those accuracies across all of the data sets. We then used the parameter combinations to construct homogeneous ensemble classifiers. Each of these homogeneous classifiers used the top two-thirds of the parameter combinations of the base classifier algorithm. This meant that the homogeneous ensembles had different numbers of base classifiers as there were different numbers of total parameters combinations for each of the base classifier types. We also created heterogeneous ensembles by using all of the base classifier combinations that were used in the homogeneous ensembles. Each of the ensembles were combined using either voting or the sum rule.

We then repeated the same strategy for creating creating ensembles, with the additional use of bagging. Each of the bagged ensembles were constructed using 200 base classifiers trained on a bag with the same size as the original training set. We evaluated the ensembles by measuring the accuracy using 2-fold cross validation for each of the data sets and then averaging over the all of the data sets.

4.3. Results. As was expected, on average, the ensembles performed better than the base classifiers in terms of accuracy. For the averaged results corresponding to the ensemble classifiers, the only clear trend was the improvement due to the use of the sum rule over voting when not using bagging, as can be seen in Figure 4.1. Surprisingly, the results were

Classification		***Fusion***
Type	*Algorithm*	*Function*
Decision Tree	Random Tree (RT) [7]	Voting [19]
Probabilistic	Naive Bayes (NB) [16, 20]	Sum Rule [19]
Function	Support Vector Machine (SVM) [17, 21, 2]	
Instance Based	k Nearest Neighbor (KNN) [1]	
Rule Based	Ripper [9, 23]	

TABLE 4.1
The classification algorithms and fusion functions used in HEMLOCK *to create ensembles.*

not as clear for bagging. Figure 4.2 shows no clear correlation between bagging and accuracy. The difference in accuracies for the different ensemble methods is not clear when averaged over all the data sets either. The heterogeneous ensembles and the decision tree ensembles result in higher averaged accuracies, but the differences may not be significant. More work on analyzing this data is needed.

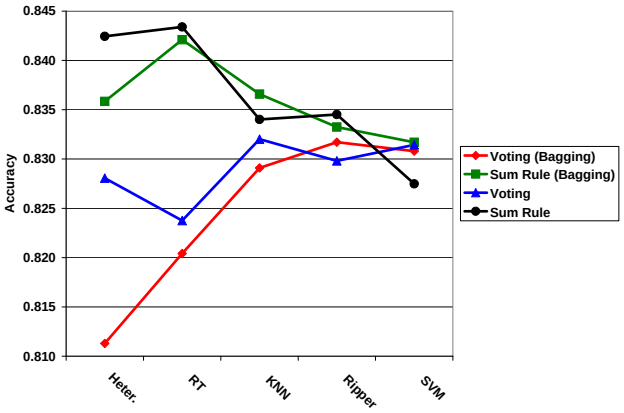


FIG. 4.1. *The average accuracies over all data sets showing that ensembles that use the sum rule do better than those that use voting.*

The accuracies for each data set and the averaged accuracies are presented in Appendix C. When the accuracies are averaged over all data sets, it is difficult to see any advantages over using one ensemble method over the other. However when looking at the details of how these methods perform on particular data sets, there are often large differences in accuracy. This suggests that exploring these different ensemble methods may lead to better understanding of the behavior of the different methods. However, the ensemble generation techniques need to be fine tuned to take advantage of information known about the data set for which we are building a classifier model.

We did notice that when the performance of a homogeneous ensemble was poor, relative to the other homogeneous ensembles trained on the same data set, then the heterogeneous ensemble also had lower performance. Examples of this can be seen in Figure 4.3. There are points in this figure where the random tree ensembles perform much worse than the average homogeneous ensemble, leading to mediocre heterogeneous ensemble performance. This suggests that if one of the homogeneous ensembles performs poorly, we should not include the associated base classifier type into the heterogeneous ensemble, or we should include less

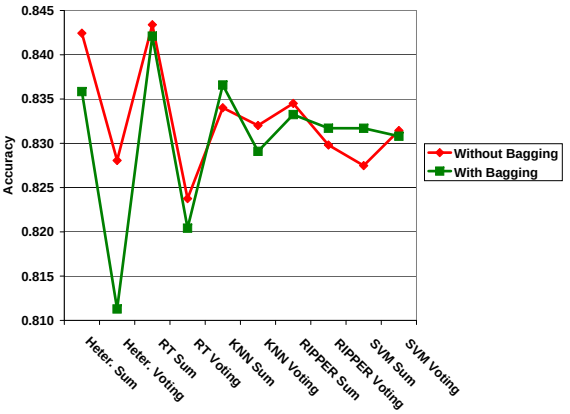


FIG. 4.2. *The average accuracies over all data sets, comparing the use of bagging for each ensemble method.*

base classifiers of that type. More work is needed to see how well this idea generalizes to other data sets.

Some of the results in Figure 4.3 indicate that all the ensemble methods performing equally well for a particular data set. However, the results for the *letter* data set illustrate that heterogeneous ensembles can outperform the homogeneous ensembles. In this work there was no systematic method used for selecting what balance of base classifiers to use, and this result indicates that paying closer attention to such a detail may be important.

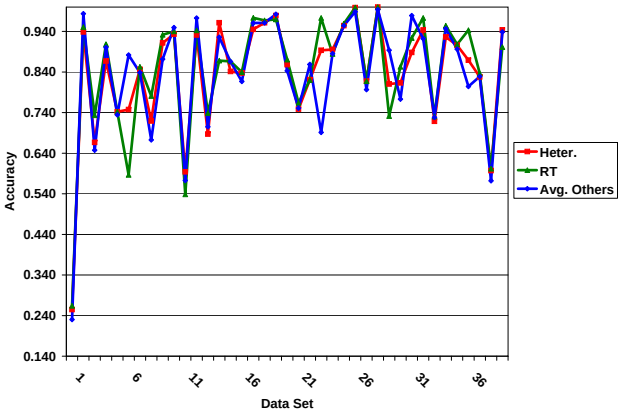


FIG. 4.3. *Accuracy of the heterogeneous ensemble, random tree ensemble, and the average for the rest of the homogeneous ensembles, for each data set.*

5. Future Work. This work was an exploratory experiment in learning more about creating heterogeneous ensembles and how they can be used effectively. The majority of the work went into creating the HEMLOCK framework, which allows us to experiment with these methods in future work.

We need to create better methods for training our ensembles for the data sets in which we are interested. The training sets may include indicators which will allow us to fine tune the way the base classifiers are generated or to help determine which ensemble techniques to use. This will be the main focus of future work. We would also like to determine, in

general, whether the strategies used for training homogeneous ensembles should be used for training heterogeneous ensembles as well. It will be important to explore which types of base classifiers lead to better ensemble performance. This could result in either general guidelines or a set of new methods.

We would also like to explore novel methods that take advantage of the unique nature of heterogeneous classifiers. The different base classifier algorithms all have their own strengths, and it would be interesting to try to use each of the base classifiers in ways that take advantage of those strengths. For example, if a base classifier works better with nominal features, then it could be trained on a subspace of the original training set with just the nominal features.

6. Conclusions. Our initial attempt at creating heterogeneous ensembles did not lead to significant gains in classifier performance. However, the results presented here will serve as a good benchmark for evaluating performance of ensemble classifiers. We have demonstrated that heterogeneous ensembles perform better than homogeneous ensembles in some cases, but more work is needed to better understand under what circumstances the use of heterogeneous ensemble classifiers leads to this improvement. Using the Hemlock software framework developed in this work, we can now investigate and evaluate new methods for heterogeneous ensemble classification, and we intend to follow up on the many questions identified in work presented here.

We also found that the sum rule performs better than voting on average. We expected this because the sum rule has more information from the base classifiers, since it uses measurement values. This result has lead us to believe that it is worth while to continue exploring fusion functions that use measurement values.

7. Acknowledgments. We would like to thank Philip Kegelmeyer for providing the data sets for our testing and for helpful suggestions throughout the project. We also thank the developers of the Weka and Jama libraries used in Hemlock.

REFERENCES

[1] D. Aha and D. Kibler, *Instance-based learning algorithms*, Machine Learning, 6 (1991), pp. 37–66.
[2] M. Aizerman, E. Braverman, and L. Rozonoer, *Theoretical foundations of the potential function method in pattern recognition learning*, Automation and Remote Control, 25 (1964), pp. 821–837.
[3] R. Banfield, L. Hall, K. Bowyer, and W. P. Kegelmeyer, *A comparison of decision tree ensemble creation techniques*, IEEE Trans. Pat. Recog. Mach. Int., 29 (2007), pp. 173–180.
[4] J. Basilico, D. Dunlavy, S. Verzi, T. Bauer, and W. Shaneyfelt, *Yucca mountain LSN archive assistant*, Tech. Rep. SAND2008-1622, Sandia National Laboratories, 2008.
[5] S. Bian and W. Wang, *On diversity and accuracy of homogeneous and heterogeneous ensembles*, Intl. J. Hybrid Intel. Sys., 4 (2007), pp. 103–128.
[6] L. Breiman, *Bagging predictors*, Machine Learning, 24 (1996), pp. 123–140.
[7] L. Breiman, *Random forests*, Machine Learning, 45 (2001), pp. 5–32.
[8] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes, *Ensemble selection from libraries of models*, in Proc. ICML, 2004.
[9] W. W. Cohen, *Fast effective rule induction*, in Twelfth International Conference on Machine Learning, Morgan Kaufmann, 1995, pp. 115–123.
[10] T. G. Dietterich, *Approximate statistical tests for comparing supervised classification learning algorithms*, Neural Comput., 10 (1998), pp. 1895–1923.
[11] T. G. Dietterich, *Ensemble methods in machine learning*, in Proc. International Workshop on Multiple Classifier Systems, 2000, pp. 1–15.
[12] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, Wiley-Interscience, 2nd ed., 2000.
[13] R. P. W. Duin, *The combining classifier: To train or not to train*, in Proc. International Conference on Pattern Recognition, vol. 2, 2002, pp. 765–770.
[14] T. Fawcett, *An introduction to roc analysis*, Pattern Recogn. Lett., 27 (2006), pp. 861–874.
[15] T. K. Ho, *The random subspace method for constructing decision forests*, IEEE Trans. Pattern Anal. Mach. Intell., 20 (1998), pp. 832–844.

[16] G. H. John and P. Langley, *Estimating continuous distributions in bayesian classifiers*, in Eleventh Conference on Uncertainty in Artificial Intelligence, 1995, pp. 338–345.
[17] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy, *Improvements to platt's smo algorithm for svm classifier design*, Neural Comput., 13 (2001), pp. 637–649.
[18] J. Kittler, M. Hatef, R. P. W. Duin, and J. Matas, *On combining classifiers*, IEEE Trans. Pattern Anal. Mach. Intell., 20 (1998), pp. 226–239.
[19] L. I. Kuncheva, *Combining Pattern Classifiers: Methods and Algorithms*, Wiley-Interscience, 2004.
[20] T. M. Mitchell, *Machine Learning*, McGraw-Hill, New York, 1997.
[21] J. C. Platt, *Fast training of support vector machines using sequential minimal optimization*, in Advances in Kernel Methods: Support Vector Learning, MIT Press, Cambridge, MA, USA, 1999, pp. 185–208.
[22] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice-Hall, Englewood Cliffs, NJ, 2nd ed., 2003.
[23] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*, Addison Wesley, May 2005.
[24] W. Wang, D. Partridge, and J. Etherington, *Hybrid ensembles and coincident failure diversity*, in Proc. International Joint Conference on Neural Networks, 2001.
[25] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*, Morgan Kaufmann, June 2005.
[26] D. H. Wolpert, *Stacked generalization*, Neural Netw., 5 (1992), pp. 241–259.

Appendix

A. Base Classifiers. A base classifier refers to a single classifier model whose output is used as input for an ensemble classifier. The base classifiers used in this work consist of several typical learning models used in ensemble classifier models. The base classifiers chosen are representative of the major strategies currently used in solving many classification problems: decision trees, probabilistic models, functional models, instance-based models, and rule-based models.

Decision Trees. Decision tree algorithms create models using a divide and conquer strategy to recursively partition the feature space along feature axes until "pure" partitions (i.e., partitions containing data from a single class only) are found. Choices for a partitioning strategy (i.e., how to choose features and split values), stopping criteria for growing trees (i.e., if and when to stop growing a tree before pure partitions are found), and a partition aggregation strategy (i.e., how to prune a tree to avoid overfitting the data) lead to different decision tree methods. Also, sampling from feature space at each decision node leads to *random decision trees*. For small data sets, the resulting decision trees are often easy to interpret and analyze, thus making decision trees a popular choice for data analysts [7].

Bayesian Classifiers. Bayesian classifiers consist of a probability model for each class combined with a decision rule for choosing the class for a given data instance. The probability model is a conditional model that is estimated using Bayes' Theorem:

$$p(y_i|\mathbf{x}_i) = \frac{p(y_i)\ p(\mathbf{x}_i|y_i)}{p(\mathbf{x}_i)} \equiv \text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}} \ . \tag{A.1}$$

The naive Bayes classifier is a particular Bayesian classifier which includes the assumption that the features are conditionally independent:

$$p(\mathbf{x}_i|y_i) = p(\langle a_1(\mathbf{x}_i), \ldots, a_m(\mathbf{x}_i)\rangle|y_i) = \prod_{j=1}^{m} p(a_j(\mathbf{x}_i)|y_i) \ . \tag{A.2}$$

A common decision rule for Bayesian classifiers is to choose the class that is most probable (i.e., the *maximum a posteriori*, or MAP, decision rule) [16, 20].

Support Vector Machines. Support vector machines (SVMs) [17, 21] are linear classifiers designed to find a hyperplane which simultaneously minimizes classification error and maximizes the distance, or margin, between the hyperplane and data instances from two different classes. Extensions of SVMs used in the work presented here include methods for handling misclassifications (using soft, or relaxed, margins) and for embedding data into higher dimensional feature spaces in order to estimate nonlinear decision boundaries in the original feature space (i.e., the *kernel trick* [2]).

Nearest Neighbor Classifiers. Nearest neighbor classifiers [1] are examples of classifiers that do not need training. To label a data instance, a nearest neighbor classifier examines the instances in the training set that are closest to it in terms of feature similarity or distance in some metric space and predicts a label based on the labels of those "neighboring" instances. The class boundaries are therefor implicit, often leading to more flexible decision boundaries than some of the other explicitly formed boundaries discussed in this section. Choices for the number of neighbors, the similarity/distance measure, and voting/weighting schemes for combining information (attributes, labels, etc.) from neighbors lead to different variants of nearest neighbor classifiers.

Association Rules. Models created from these classifier methods are sets of rules consisting of logical conjunctions of decision boundaries along feature axes. An example of a

rule is

$$\{a_1 = \text{“scalar”}\} \wedge \{a_2 > 1.0\} \implies y_1 \,. \tag{A.3}$$

These rules are often built in a general to specific manner, where new conditions are added to the conjunction as long as the rule continues to improve classifier performance (see Section 3.2 for more information on classifier performance). An opposing approach, a specific to general rule building strategy, starts with a specific rule targeting some instance in the training set and then proceeds by removing conditions from the conjunction while the rule continues to improve classifier performance [9, 23].

B. Data Sets Used in Experiments.

#	*Name*	*Instances*	*Classes*	*Continuous Attributes*	*Nominal Attributes*
1	abalone	4177	29	7	1
2	anneal	898	6	6	32
3	bupa	345	2	6	0
4	car	1728	4	0	6
5	credit-g	1000	2	7	13
6	dna	3186	3	0	180
7	ecoli	326	8	7	0
8	glass	214	6	9	0
9	ion	351	2	34	0
10	iris	150	3	4	0
11	krk	28056	18	6	0
12	krkp	3196	2	0	36
13	led-24	5000	10	0	24
14	letter	36000	26	16	0
15	lrs	530	10	93	0
16	lymph	148	4	3	15
17	nursery	12961	5	0	8
18	page	5473	5	10	0
19	pendigits	10993	10	16	0
20	phoneme	5404	2	5	0
21	pima	768	2	8	0
22	promoters	106	2	0	57
23	ringnorm	300	2	20	0
24	sat	6435	6	36	0
25	segment	2310	7	19	0
26	shuttle	58000	7	9	0
27	sonar	208	2	60	0
28	soybean-small	47	4	0	35
29	splice	3190	3	0	60
30	threenorm	300	2	20	0
31	tic-tac-toe	958	2	0	9
32	twonorm	300	2	20	0
33	vehicle	846	4	18	0
34	vote	435	2	0	16
35	vote1	435	2	0	15
36	vowel	528	11	10	0
37	waveform	5000	3	21	0
38	yeast	1484	10	8	0
39	zip	9298	10	256	0

TABLE B.1
Meta information for each data set used in the experiments.

C. Numerical Results.

Data Set	*Heter.*	*RT*	*KNN*	*RIPPER*	*SVM*
abalone	0.258	0.267	0.200	0.211	0.264
anneal	0.958	0.967	0.983	0.982	0.986
bupa	0.698	0.733	0.638	0.681	0.623
car	0.900	0.911	0.916	0.862	0.924
credit-g	0.734	0.735	0.735	0.733	0.743
dna	0.626	0.589	0.801	0.936	0.921
ecoli	0.834	0.859	0.840	0.853	0.847
glass	0.771	0.793	0.676	0.700	0.635
ion	0.929	0.920	0.846	0.900	0.889
iris	0.940	0.933	0.947	0.933	0.955
krk	0.550	0.538	0.687	0.747	0.287
krkp	0.946	0.940	0.962	0.992	0.970
led-24	0.736	0.741	0.624	0.744	0.750
letter	0.914	0.871	0.982	0.954	0.845
lrs	0.864	0.861	0.877	0.847	0.885
lymph	0.858	0.864	0.819	0.758	0.824
nursery	0.965	0.971	0.978	0.972	0.930
page	0.966	0.968	0.960	0.971	0.951
pendigits	0.972	0.971	0.993	0.974	0.980
phoneme	0.868	0.868	0.893	0.861	0.772
pima	0.747	0.766	0.724	0.754	0.768
promoters	0.906	0.858	0.811	0.821	0.908
ringnorm	0.953	0.963	0.563	0.733	0.717
sat	0.888	0.885	0.910	0.883	0.864
segment	0.963	0.965	0.964	0.965	0.930
shuttle	0.999	0.999	0.999	1.000	0.966
sonar	0.769	0.841	0.851	0.761	0.727
soybean-small	1.000	1.000	1.000	0.981	1.000
splice	0.742	0.726	0.813	0.950	0.924
threenorm	0.853	0.860	0.783	0.657	0.830
tic-tac-toe	0.902	0.896	0.977	0.979	0.983
twonorm	0.960	0.973	0.950	0.827	0.963
vehicle	0.729	0.719	0.691	0.701	0.752
vote	0.938	0.945	0.936	0.956	0.952
vote1	0.906	0.899	0.903	0.901	0.920
vowel	0.922	0.956	0.975	0.752	0.684
waveform	0.844	0.837	0.800	0.814	0.865
yeast	0.621	0.606	0.551	0.595	0.590
zip	0.924	0.899	0.967	0.907	0.949
mean	0.842	0.843	0.834	0.835	0.827
std	0.148	0.149	0.166	0.151	0.171

TABLE C.1

Accuracy results for ensembles without bagging using the sum rule as the fusion function.

Data Set	*Heter.*	*RT*	*KNN*	*RIPPER*	*SVM*
abalone	0.270	0.262	0.221	0.188	0.261
anneal	0.941	0.915	0.979	0.982	0.986
bupa	0.733	0.730	0.612	0.684	0.641
car	0.840	0.807	0.920	0.863	0.928
credit-g	0.721	0.717	0.737	0.727	0.752
dna	0.569	0.542	0.790	0.931	0.924
ecoli	0.859	0.849	0.840	0.807	0.864
glass	0.762	0.754	0.665	0.686	0.644
ion	0.917	0.920	0.843	0.900	0.866
iris	0.953	0.939	0.961	0.913	0.966
krk	0.464	0.467	0.695	0.715	0.284
krkp	0.884	0.861	0.953	0.994	0.949
led-24	0.660	0.631	0.648	0.744	0.747
letter	0.828	0.782	0.970	0.943	0.845
lrs	0.870	0.863	0.870	0.845	0.888
lymph	0.838	0.840	0.839	0.737	0.845
nursery	0.959	0.967	0.979	0.969	0.931
page	0.966	0.967	0.958	0.972	0.950
pendigits	0.970	0.964	0.993	0.972	0.980
phoneme	0.852	0.850	0.882	0.860	0.758
pima	0.762	0.759	0.734	0.758	0.779
promoters	0.820	0.802	0.792	0.803	0.906
ringnorm	0.937	0.967	0.547	0.754	0.730
sat	0.882	0.881	0.908	0.880	0.867
segment	0.951	0.952	0.956	0.968	0.934
shuttle	0.999	0.999	0.999	1.000	0.966
sonar	0.788	0.822	0.822	0.755	0.765
soybean-small	1.000	1.000	1.000	0.981	1.000
splice	0.848	0.814	0.780	0.947	0.919
threenorm	0.830	0.843	0.850	0.694	0.854
tic-tac-toe	0.872	0.874	0.979	0.973	0.983
twonorm	0.960	0.977	0.940	0.817	0.960
vehicle	0.736	0.723	0.705	0.686	0.748
vote	0.945	0.952	0.929	0.954	0.963
vote1	0.922	0.910	0.897	0.894	0.915
vowel	0.848	0.920	0.913	0.769	0.710
waveform	0.838	0.828	0.807	0.821	0.871
yeast	0.601	0.608	0.573	0.582	0.596
zip	0.900	0.866	0.963	0.897	0.950
mean	0.828	0.824	0.832	0.830	0.831
std	0.151	0.155	0.161	0.154	0.169

TABLE C.2
Accuracy results for ensembles without bagging using voting as the fusion function.

Data Set	*Heter.*	*RT*	*KNN*	*RIPPER*	*SVM*
abalone	0.255	0.264	0.210	0.211	0.270
anneal	0.939	0.951	0.983	0.987	0.981
bupa	0.667	0.733	0.629	0.684	0.629
car	0.867	0.909	0.915	0.852	0.933
credit-g	0.742	0.742	0.745	0.717	0.742
dna	0.748	0.587	0.803	0.923	0.919
ecoli	0.846	0.853	0.844	0.807	0.864
glass	0.720	0.781	0.691	0.678	0.649
ion	0.912	0.932	0.852	0.889	0.874
iris	0.933	0.940	0.961	0.935	0.953
krk	0.594	0.539	0.686	0.748	0.285
krkp	0.931	0.943	0.958	0.992	0.969
led-24	0.687	0.739	0.626	0.742	0.747
letter	0.962	0.868	0.983	0.951	0.843
lrs	0.841	0.866	0.874	0.834	0.889
lymph	0.838	0.840	0.846	0.746	0.858
nursery	0.945	0.973	0.978	0.972	0.932
page	0.961	0.967	0.960	0.972	0.950
pendigits	0.979	0.970	0.993	0.974	0.980
phoneme	0.858	0.870	0.894	0.862	0.774
pima	0.749	0.764	0.720	0.758	0.775
promoters	0.821	0.820	0.821	0.830	0.925
ringnorm	0.894	0.973	0.567	0.780	0.727
sat	0.895	0.885	0.907	0.884	0.864
segment	0.954	0.959	0.966	0.965	0.932
shuttle	0.999	0.999	0.999	1.000	0.966
sonar	0.817	0.817	0.851	0.774	0.764
soybean-small	1.000	1.000	1.000	0.981	1.000
splice	0.810	0.732	0.810	0.947	0.924
threenorm	0.813	0.853	0.800	0.663	0.856
tic-tac-toe	0.888	0.924	0.973	0.981	0.983
twonorm	0.943	0.973	0.960	0.840	0.967
vehicle	0.719	0.730	0.714	0.698	0.773
vote	0.926	0.954	0.926	0.959	0.954
vote1	0.906	0.908	0.906	0.890	0.894
vowel	0.869	0.943	0.972	0.759	0.682
waveform	0.828	0.837	0.802	0.818	0.866
yeast	0.597	0.602	0.534	0.591	0.592
zip	0.944	0.901	0.966	0.902	0.948
mean	0.836	0.842	0.837	0.833	0.832
std	0.143	0.150	0.165	0.151	0.169

TABLE C.3

Accuracy results for ensembles with bagging using the sum rule as the fusion function.

Data Set	*Heter.*	*RT*	*KNN*	*RIPPER*	*SVM*
abalone	0.261	0.264	0.226	0.186	0.265
anneal	0.914	0.918	0.971	0.978	0.980
bupa	0.649	0.730	0.591	0.643	0.649
car	0.841	0.819	0.917	0.867	0.933
credit-g	0.725	0.718	0.742	0.718	0.758
dna	0.614	0.543	0.786	0.930	0.923
ecoli	0.847	0.841	0.863	0.811	0.862
glass	0.715	0.780	0.638	0.654	0.659
ion	0.900	0.931	0.843	0.903	0.878
iris	0.947	0.940	0.960	0.960	0.967
krk	0.452	0.465	0.693	0.706	0.283
krkp	0.880	0.849	0.952	0.990	0.951
led-24	0.625	0.643	0.653	0.740	0.743
letter	0.876	0.785	0.970	0.946	0.845
lrs	0.862	0.863	0.871	0.848	0.890
lymph	0.791	0.843	0.830	0.770	0.819
nursery	0.946	0.969	0.978	0.968	0.931
page	0.958	0.967	0.957	0.973	0.951
pendigits	0.964	0.965	0.992	0.973	0.980
phoneme	0.849	0.857	0.884	0.865	0.757
pima	0.714	0.740	0.743	0.738	0.766
promoters	0.735	0.698	0.736	0.886	0.935
ringnorm	0.867	0.953	0.550	0.760	0.720
sat	0.881	0.880	0.910	0.878	0.865
segment	0.935	0.959	0.957	0.964	0.935
shuttle	0.998	0.999	0.999	1.000	0.966
sonar	0.789	0.808	0.812	0.793	0.765
soybean-small	1.000	1.000	1.000	0.982	1.000
splice	0.837	0.809	0.767	0.951	0.920
threenorm	0.820	0.830	0.833	0.694	0.846
tic-tac-toe	0.860	0.872	0.976	0.980	0.983
twonorm	0.920	0.947	0.953	0.823	0.963
vehicle	0.714	0.721	0.695	0.693	0.741
vote	0.943	0.949	0.926	0.952	0.954
vote1	0.910	0.903	0.903	0.908	0.924
vowel	0.792	0.939	0.917	0.710	0.688
waveform	0.821	0.827	0.810	0.813	0.870
yeast	0.592	0.605	0.563	0.587	0.590
zip	0.899	0.868	0.961	0.894	0.946
mean	0.811	0.820	0.829	0.832	0.831
std	0.152	0.155	0.163	0.157	0.170

TABLE C.4
Accuracy results for ensembles with bagging using voting as the fusion function.

TOWARDS SCALABLE PARALLEL HYPERGRAPH PARTITIONING

AYDIN BULUÇ* AND ERIK G. BOMAN†

Abstract. This is a work in progress report describing our attempts on developing a more scalable and robust multilevel hypergraph partitioning algorithm. Current hypergraph partitioners are not as scalable as multigrid solvers for linear systems of PDEs. We first identify the challenges that are unique to hypergraphs, which motivated the development of our algorithms. Then, we present two new algorithms for hypergraph coarsening that are based on aggregation of vertices instead of matching. Our algorithms are amenable to efficient parallelization, though we present them as serial methods here.

1. Introduction. Hypergraphs are generalizations of graphs where the interactions between vertices can be beyond pairwise. They have been used to accurately model component connectivity in VLSI circuits [1], communication volume in parallel sparse matrix-vector multiplication [4], yeast protein complex networks [12], feature-object relationships in data mining (such as term-document matrices) [16], and dependencies in distributed databases [13]. Partitioning hypergraphs is a fundamental operation that is used for load balancing, minimizing inter-processor communication, and clustering. The goal is to partition the vertices of a hypergraph into a certain number of disjoint sets of approximately the same size so that a cut metric is minimized. This problem is known to be NP-Hard [11].

A hypergraph $\mathbb{H} = (\mathbb{V}, \mathbb{N})$ is defined by a set of *vertices* $\mathbb{V}$ and a set of *nets* (*hyperedges*) $\mathbb{N}$. Each net represents a relationship among a subset of vertices, which are also called the *pins* of that net. The size of a net is given by the number of pins it has. Both nets and vertices may have weights associated with them. For ease of reference, we call the weight of a vertex its *volume*. We represent a hypergraph with a sparse matrix $\mathbf{A}$, where columns correspond to vertices and rows correspond to nets. An example hypergraph along with its sparse matrix representation is given in Figure 1.1.

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & 0 & 0 & a_{15} \\ a_{21} & a_{22} & 0 & 0 & 0 \\ a_{31} & 0 & 0 & a_{34} & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} \end{pmatrix}$$

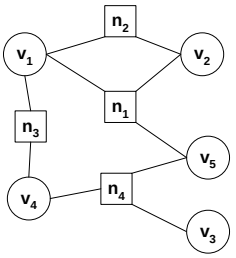


FIG. 1.1. *A hypergraph and its sparse matrix representation in the row-net model. Circles represent vertices and squares represent nets.*

Among all the methods for partitioning hypergraphs, multilevel approaches proved to be both more efficient and more robust [10]. Multilevel algorithms use a "V-cycle" implementation where the input hypergraph is successively approximated by coarser and coarser hypergraphs until it becomes small enough so that a direct partitioner/solver can be applied to it. Then, the coarse partitioning is projected back to get finer hypergraphs in the *refinement* phase. The challenge is to create good coarse approximations, because coarsening is crucial for the overall quality of the partitioning. Coarsening also has to be efficient as it is the most time consuming part of the V-cycle.

*Department of Computer Science, University of California, Santa Barbara, aydin@cs.ucsb.edu
†Scalable Algorithms Department, Sandia National Laboratories, egboman@sandia.gov

Our multilevel hypergraph partitioning approach is inspired by ideas from Algebraic Multigrid (AMG). In AMG, a sequence of "coarser" grids are constructed to solve a linear system. Thus, AMG solvers also have a V-cycle implementation similar to multilevel hypergraph partitioners. They have proved to be successful for large problems. Scalable parallel software, such as Sandia's ML [8] and LLNL's Hypre [6], has been developed.

Two notions are crucial for AMG [3]:

1. The notion of *strong influence* that determines which variables depend on each other. This drives both the coarse node selection and the mapping between coarse and fine levels.
2. The notion of *algebraic smoothness* where any error that is not reduced by relaxation on the fine grid is defined to be algebraically smooth.

AMG first fixes the relaxation operation, for example, Gauss-Seidel, then defines the restriction/interpolation operations and selects coarse variables. In hypergraph partitioning, the relaxation operation is most suitably Fiduccia-Mattheyses (FM) [7], but it is only applied when going up in the V-cycle in current implementations of multilevel hypergraph partitioning. This is one of the fundamental differences in the execution of multilevel graph/hypergraph partitioning and AMG. In AMG, there are two relaxation steps: pre-smoothing and post-smoothing. The former is applied to smooth the error in the fine grid and the latter is applied to smooth the error in the coarse grid. On the other hand, in graph/hypergraph partitioning, the relaxation is applied only as a post-smoothing operation. Therefore, coarsening is more important for graphs/hypergraphs.

For choosing coarse vertices, we rely on the notion of strong influence as AMG does. On the other hand, quantifying what constitutes as smooth error in hypergraph partitioning is not known. The $(k-1)$-cut metric is an ideal candidate except that we do not have access to it in the coarsening phase. It would have been necessary to define a variable similar to the residual in AMG, if we were to rely on the notion of algebraic smoothness. Since relaxation steps are not applied during the coarsening phase of the V-cycle in hypergraph partitioning, algebraic smoothness is not applicable.

For describing algorithms, we use the colon notation of MATLAB®. That is to say, $\mathbf{A}(:, \mathbf{v})$ denotes the submatrix indexed by the $v_1{}^{\text{st}}, v_2{}^{\text{nd}}, ..., v_n{}^{\text{th}}$ columns, where $\mathbf{v}$ is a vector of indices with size n. The row-wise notation is similar. For a vector with possible range $\{1, ..., m\}$, $\texttt{Comp}(\mathbf{v})$ denotes the complement of $\mathbf{v}$, i.e. a vector of size $m - n$ that contains all the values in $\{1, ..., m\}$ except for the values indexed by $\{v_1, ..., v_n\}$. The elementwise operations are denoted by a dot before them, such as ./ for elementwise division and .$*$ for elementwise multiplication.

1.1. Our contributions. Our main contributions are in the coarsening phase but the refinement step is also modified to work together with the new coarseness scheme. Although there are many variations, the common existing methods for coarsening can be roughly classified as:

- *Matching-based coarsening*, which involves computing a maximal matching on the similarity graph given by $\mathbf{A}^T\mathbf{A}$, and then contracting pairs of matched vertices [4, 5].
- *Hyperedge coarsening*, which involves choosing a subset of hyperedges, and then merging all vertices in each of those selected hyperedges [10].

Instead, our methods are *aggregative*, where fine vertices are aggregated around coarse seed vertices. Our first method, called *approximate weighted aggregation (AWA)*, uses weighted aggregation where each vertex can be divided into fractions that are mapped to different aggregates in the coarse hypergraph. Our second method is called *randomized strict aggregation (RSA)*, which uses strict aggregation and the rules of interpolation are determined in a randomized way. In weighted aggregation, *interpolation order* is defined as the maximum

number of fractions a vertex can be divided into at every level of the algorithm. We have a conservative approach with using fractions due to the issues explained in the next section. Therefore, the interpolation order is kept low.

To the best of our knowledge, neither a strict aggregative method, nor a weighted one has been applied to hypergraph coarsening. Our work generalizes the approach by Safro et al. [14] from graphs to hypergraphs. Even though our work is presented in serial, it is amenable to efficient parallelization as it is either readily vectorized or composed of independent loops that can execute in parallel. The data from the experiments are not available yet, but we believe that our algorithms will show promise since they are based on AMG solvers that proved to successful and scalable.

2. Aggregative Coarsening. Since the early days of multilevel graph partitioning, coarser graphs were constructed by matching pairs of vertices and contracting edges that connects matched pairs [9]. Although this approach initially allowed only pairs of vertices to be clustered, more general approaches that enable multinode clustering has also been also developed later.

Aggregative clustering, on the other hand, chooses a certain number of coarse vertices that serve as seeds for aggregates and lets fine vertices aggregate around those seeds. This distinction of seeds versus non-seeds lets the algorithm decouple the seed selection phase with the interpolation phase where it decides the aggregation of fine vertices around seeds. Aggregation can be either strict or weighted. In the former, each fine vertex is a member of only one aggregate, where in weighted aggregation each fine vertex can be divided into fractions and become members of multiple aggregates. We differentiate between matching and strict aggregation due to this distinction of seeds versus non-seeds in the latter. For coarsening graphs, Safro et al. uses AMG based weighted aggregation [14]. We first summarize their approach in Section 2.1, and then explain why a direct application of their method is impractical for hypergraphs in Section 2.2. The set of coarse vertices (seeds) are donated by $\mathbb{C}$, and the set of fine vertices (non-seeds) by $\mathbb{F}$, where $\mathbb{V} = \mathbb{C} \cup \mathbb{F}$.

2.1. Coarsening Graphs with Weighted Aggregation. The aggregative coarsening process in multilevel graph partitioning involves three steps: seed selection for aggregates, determining the rules of interpolation for non-seed vertices, and finally establishing the connections between aggregates. Seeds are chosen according to their ability to attract other vertices around them. Initially all vertices are considered to be in $\mathbb{F}$ and nodes are sequentially transfered to $\mathbb{C}$ if they are likely to aggregate large volumes of fine vertices around them. Once the seeds are fixed, the rules of interpolation for non-seed vertices are determined by running a similar procedure, except that the actual volumes are computed since $\mathbb{F}$ and $\mathbb{C}$ are known. The interested reader should consult the original paper [14].

Setting up the strength of connections between aggregates is done in such a way that the cut size is preserved in a probabilistic sense. Every edge from vertex i (v_i) to vertex j (v_j) in the fine graph is essentially partitioned into (possibly many) pieces depending on the distribution of the fractions of the vertices that it connects. The weight of the edge connecting aggregates k and l is given by

$$w^c_{kl} = \sum_{i \neq j} P_{ik} w_{ij} P_{jl}, \tag{2.1}$$

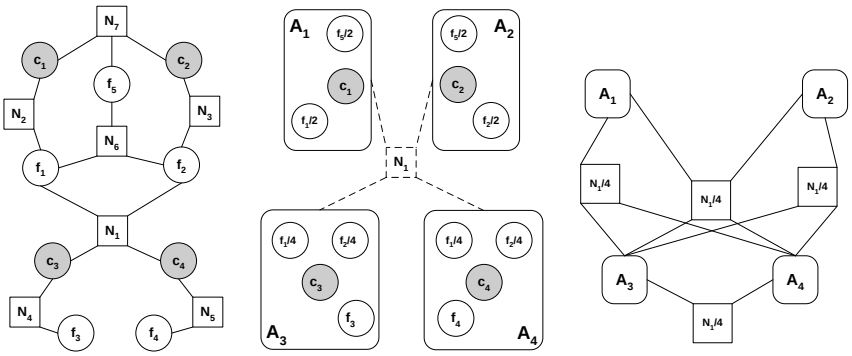
where w_{ij} is the weight of the edge between v_i and v_j in the fine graph, P_{ik} is the fraction of v_i's volume that contributes to aggregate k (A_k), and P_{jl} is the fraction of v_j's volume that contributes to aggregate l (A_l). We can also write the interpolation operation in matrix form: $\mathbf{A}^c = \mathbf{P}^T\mathbf{A}\mathbf{P}$. Please note that the matrix $\mathbf{A}$ here is the adjacency matrix of the graph. Since

these fractional volume assignments are going to be resolved during the refinement phase, we can also think of P_{ik} as the probability of v_i being a member of aggregate A_k. The crucial property of this weight assignment is that for any partitioning of the coarse graph, the cut size is preserved in a stochastic manner when injected back to the fine graph.

2.2. Issues Specific to Hypergraphs. Hypergraphs are conceptually generalized graphs in the sense that connections of vertices need not be pairwise. A hyperedge can connect any number of vertices. This makes them more expressive than graphs but operations on them are usually more computationally expensive than operations on graphs. We explain some of these issues in this section, whereas Section 3 includes the algorithms we propose in response to those issues.

The notion of "influence" on graphs, is defined in terms of the ability of a node to attract other nodes around it. The metric used by Safro et al. relies on the degrees of vertices [14]. In a graph, this means that a vertex with volume v_i, and degree d_i (i.e. it has d_i vertices that are immediately connected to it through a single edge), contributes v_i/d_i volume to each of its neighboring vertices.

For hypergraphs, we specifically avoided imitating the vertex-vertex influences approach used in graphs and opted for a diffusion-based approach. Our first motivation is economical. Finding a connectedness measure between all pairs of vertices requires computing the matrix product $\mathbf{A}^T\mathbf{A}$. We consider doing a full sparse matrix-matrix multiplication to be an overkill for a heuristic with no guarantees. It has also been observed that the product may be quite dense to store, so computing it may require multiple passes [5] that makes the operation too expensive for large problems. The second motivation is that our diffusion-based approach implicitly gives preference to smaller hyperedges, helping to reduce the exposed hyperedge weight quickly [10].



(a) Seed Selection (b) Interpolation rules (c) Connection Setup

FIG. 2.1.

Another complication arises when we try to preserve the probabilistic hyperedge cut size in weighted aggregation. Figure 2.1(a) shows the fine graph after seeds are chosen according to some criteria. The seed nodes $(c_1, ..., c_4)$ are gray, and non-seeds $(f_1, ..., f_4)$ are white. Figure 2.1(b) shows the aggregates formed around seed vertices following the rules of interpolation described in Section 3.2. Here, f_i/n means $1/n$ fraction of fine vertex f_i.

The net N_1 connects at least some fraction of vertices in each of the four aggregates $(A_1, ..., A_4)$. However, by letting N_1 connect all four aggregates, we cannot assign it an appropriate uniform weight that preserves the cut size in a stochastic manner. Consider the following possible partitionings (bisections) :

1. $P = \{\{A_1, A_2, A_3\}, \{A_4\}\}$. The cut size of this bisection is guaranteed to be 1 since N_1 connects the seeds of A_3 and A_4, namely it connects c_3 and c_4
2. $P = \{\{A_1, A_2\}, \{A_3, A_4\}\}$. As said before, N_1 is definitely connected to both A_3 and A_4. The probability that it connects neither A_1 nor A_2 is $1/2 \cdot 1/2 = 1/4$. Therefore, the expected cut size is $1 - 1/4 = 3/4$.
3. $P = \{\{A_1\}, \{A_2, A_3, A_4\}\}$. The expected cut size of this bisection is $1/2$ as the probability of N_1 connecting A_1 is equal to the probability of f_1 being a member of A_1, which is $1/2$.

The moral of this example is that it demonstrates the challenges of capturing the topology right in the case of hypergraphs. The correct modelling of N_1 in the coarse hypergraph is shown in Figure 2.1(c), where each of $N_1/4$ gets 1/4 of the total weight of N_1. However, the number of possible net combinations that would capture the semantic correctly can be exponential. Such a problem did not exist for graphs since edges are always pairwise, avoiding any combinatorial explosion. Also, a graph can only get so dense ($|\mathbb{E}| = O(|\mathbb{V}|^2)$, but the number of possible nets in an hypergraph has the same cardinality as the power set.

3. Aggregative Hypergraph Coarsening. We present our two algorithms together and point out differences whenever necessary. Our randomized strict aggregative method, RSA, has the advantage of preserving the cut size but it may be less expressive than AWA, our weighted aggregative method. AWA uses weighted aggregation with approximate net (hyperedge) weights. It requires more effective limiting of interpolation order to ensure that successively coarser hypergraphs indeed get smaller. Both algorithms has the same seed selection stage, but they differ in interpolation and connection setup stages.

3.1. Seed Selection. We choose seeds using a similar idea from aggregative graph coarsening [14]. However, our method is specifically tailored to hypergraphs and achieves low computational complexity by not examining vertex-vertex influences. We use the *future volume* concept where the future volume of an aggregate seeded by vertex v_i is the amount of volume it can accumulate from other vertices assuming that all other vertices are non-seeds. The algorithm is iterative in nature. Every iteration involves two sparse matrix-vector multiplication (matvec) operations. During the first iteration, every vertex is assumed to be in $\mathbb{F}$. With the first matvec, we calculate how much of its volume is going to disseminated among its neighboring hyperedges. In our scheme, a vertex with volume v_i and n_i hyperedges connected to it contributes v_i/n_i of its volume to each of those hyperedges. Then, the volumes accumulated in hyperedges are disseminated equally to vertices connected to them.

The full algorithm is given in Figure 3.3. Recall that the the input $\mathbf{A}$ is a sparse matrix that represents the hypergraph according to the row-net model. In the pseudocode, the nonzero values in $\mathbf{A}$ are volumes of vertices in the hypergraph. In real implementations, weights are represented as separate arrays to avoid repetitions and to save space.

By the time we have calculated the future volumes at line 6, total volume is preserved, i.e. the average future volume is 1. The $\texttt{Top}(\mathbf{v}, p)$ function returns the indices of p elements of vector $\mathbf{v}$ with highest values. We could have just picked vertices with larger than average future volumes but that would unnaturally favor highly connected clusters. As we have no knowledge about seeds in this step, we can incorrectly choose all the vertices in a highly connected cluster as seeds. That would leave many loosely connected clusters with no seeds at all.

An example hypergraph is given in Figure 3.1, which can be bisected with an hypergraph cut size of only 1. The values inside the vertices are their (undamped) future volumes computed by the initial iteration. This is an hypothetical example that is bisected with at most 25% imbalance, strict-aggregation process, and two aggregates only. The difference between a direct (one step) seed selection and an iterative one can be seen in Figures 3.2(a) and 3.2(b).

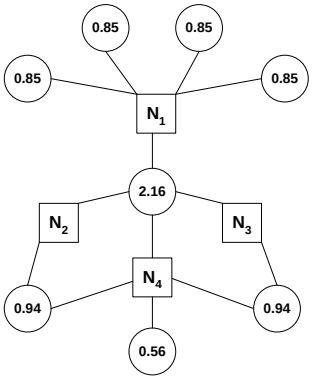
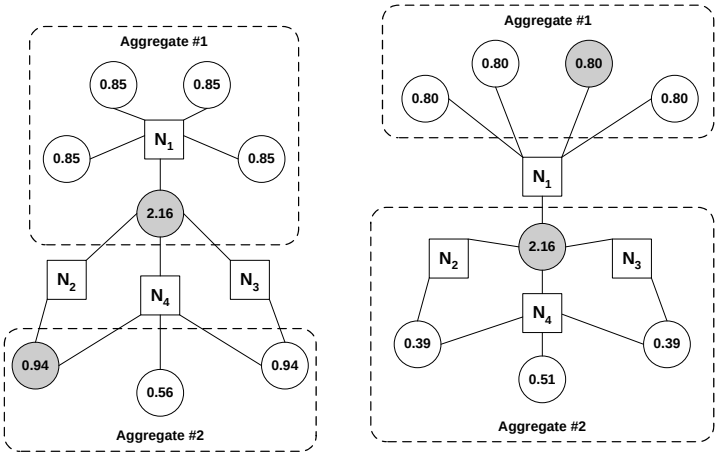


FIG. 3.1. *A hypergraph after initial future volume calculation*



(a) Direct seed selection (b) Iterative seed selection

FIG. 3.2.

Again, seed vertices are marked with gray. We see that the iterative version avoids formation of clustered seeds by correctly damping the future volumes as new vertices are added to the set of seeds.

A fully sequential algorithm that chooses seeds one by one (also called *agglomerative*) seems to be more effective, but it is also very hard to parallelize. On the other hand, a one-step algorithm offers the highest inherent parallelism, but does not perform as well. Here, we make a compromise between those two extremes by opting for an iterative yet parallelizable method that chooses seeds in batches. Each iteration adds another p% of all vertices to the set of seeds (p=10 in the code), so that we iterate just 4-5 times until we have enough seed vertices. For those who are familiar with the BSP model [15, 2], this also means the algorithm halts in 4-5 parallel BSP steps.

Therefore, we first pick the most obvious candidates for being seeds. For the remaining seeds we use an iterative approach that damps the future volume of a node according to its coupling to an already chosen seed vertex. It is crucial to note that this damping is not

seeds : $\mathbb{R}^{|\mathbb{C}|\times 1}$ = CHOOSE-SEEDS(**A** : $\mathbb{R}^{|\mathbb{N}|\times|\mathbb{V}|}$)

```
 1  vsize ← ColSizes(A)
 2  hsize ← RowSizes(A)
 3  x ← 1./vsize
 4  y ← A · x
 5  y ← y./hsize
 6  vol ← A^T · y
 7  seeds ← Top(vol, 0.1 · |vol|)
 8  while (|seed| < limit)
 9        do nonseeds ← Comp(seeds)
10           ncoarse ← ExcNetCoarseness(A)
11           vcoarse ← A(:, nonseeds)^T · ncoarse
12           damp ← vcoarse./vsize(nonseeds)
13           ind ← Top(vol(nonseeds). ∗ (1 − damp), 0.1 · |vol|)
14           seeds ← [seeds; ind]
```

FIG. 3.3. *The algorithm to choose seed vertices*

cumulative. In other words, the $(i+1)^{\text{th}}$ damping is applied to the initial future volumes, not to the output of the i^{th} damping.

Coarseness of a net is the ratio of the number of coarse vertices attached to it to the total number of vertices attached to it. Exclusive coarseness, which is defined with respect to a vertex, is the coarseness of a net when we ignore the reference vertex. For example, if a net has 2 seeds and 3 nonseeds connected to it, its exclusive coarseness with respect to one of its neighboring seeds is $1/4$ whereas it is $2/4$ with respect to a neighboring nonseed. Calculating the number of coarse neighbors of each net can done in potentially sublinear time (with an appropriate O(flops) matvec implementation) by multiplying **A** with a binary vector of size $|\mathbb{V}|$ that has 1 for every coarse vertex, and 0 for every fine vertex. The element-wise ratio of this vector and the **hsize** vector gives the net coarseness values. The coarseness values of nets are then diffused into vertices connected to them, using a matvec with the matrix representing the hypergraph induced by the vertices still in $\mathbb{F}$ and the net coarseness vector.

Pure weighted aggregation, however, may create quite dense vertex mappings since is has no way of controlling the interpolation order. As the size distribution of the nets gets skewed, the computational problem is exacerbated. For an example, think about a big hyperedge. The hypergraph induced by the vertices attached to a net with size 10 is shown in Figure 3.4. From the perspective of the fine vertex f_2, all coarse vertices have the same similarity metric, so that its volume is divided equally into 5 during the coarsening. When the size of the net is significantly big, it becomes a computational burden to keep track of many fractional mappings.

The flexibility of the iterative seed selection can be used to mitigate this problem as well. This is done by limiting the number of vertices chosen as seeds at each iteration. We can simply label a net "frozen" after any iteration. The fine vertices connected to a frozen net are not eligible for being seeds in the subsequent iterations, thus implicitly limiting the interpolation order. The freeze operation could have easily been implemented by setting the coarseness values of nets that already has more coarse vertices that a prespecified threshold manually to 1. However, we opted not to limit the interpolation order this way as it was not respecting the topology of the underlying hypergraph, and even an iterative approach might not be enough to keep the level mapping matrix ($\mathbb{F} \Rightarrow \mathbb{C}$) sparse enough.

One commonly used method to get around this problem is to truncate the interpolation

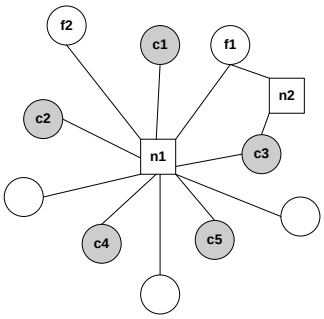


FIG. 3.4. *An hyperedge with 5 coarse vertices attached to it*

operator by ignoring all entries in a column that are less than the mean of that column, and distributing their values to the remaining entries in that column. By looking at Figure 3.4, we see that f_2 is a member of 5 different aggregates, in an equal manner. Therefore, we cannot truncate any entries. However, f_1 is strongly connected to c_3 through two different nets and relatively weakly connected to remaining seeds. Our truncation is going to ignore all less than average couplings, thus deleting any mappings from f_1 to c_1, c_2, c_4, and c_5. However, remember that we opted for a diffusion-based algorithm in order to avoid computing expensive vertex-vertex influences. For the same reason, we also do not want to explicitly construct the $\mathbb{F} \Rightarrow \mathbb{C}$ mapping. Instead, we represent the mappings $\mathbb{C} \Rightarrow \mathbb{N}$ and $\mathbb{N} \Rightarrow \mathbb{F}$ in sparse matrices $\mathbf{P}_{\mathrm{C}}^{\mathrm{N}}$ and $\mathbf{P}_{\mathrm{N}}^{\mathrm{F}}$. The actual fractional mapping from $\mathbb{F}$ to $\mathbb{C}$ is implicitly stored as the product of those two relatively sparser matrices:

$$\mathbf{P}_{\mathrm{N}}^{\mathrm{F}} \cdot \mathbf{P}_{\mathrm{C}}^{\mathrm{N}} = \mathbf{P}_{\mathrm{C}}^{\mathrm{F}} \in \mathbb{R}^{|\mathrm{F}| \times |\mathrm{C}|} \tag{3.1}$$

We control the sparsity of $\mathbf{P}_{\mathrm{C}}^{\mathrm{N}}$ and $\mathbf{P}_{\mathrm{N}}^{\mathrm{F}}$ matrices when we determine the interpolation rules in the next phase.

3.2. Rules of Interpolation. We first determine the $\mathbb{C} \Rightarrow \mathbb{N}$ mapping by letting each coarse vertex choose the set of fine vertices that are going to be members of its aggregate. Letting the coarse vertices initiate the mapping is more amenable to efficient implementation and it allows us to control sparsity without explicitly forming $\mathbf{P}_{\mathrm{C}}^{\mathrm{F}}$. Each seed chooses a constant (`MAXNEIGH`) number of nets to invade/claim among its neighbors, based on the following criteria:

- Select the net with lowest exclusive coarseness value. We assume $h_i > 1$, as nets with size 1 does not contribute to the cut metric, thus should have been pruned before.
- In the case of ties, choose the net with smaller size (not shown in code).

It is worth noting that the number of nets a seed can claim is constant only in that level of the algorithm. It adaptively changes as the density of the hypergraph changes on different levels of coarsening. The number of coarse vertices that claim a given net are naturally limited. Even when a fine vertex chooses a big net to disseminate its volume, the mapping is not going to be too scattered since that big nets are not likely to be claimed by many seeds. $\mathbf{P}_{\mathrm{C}}^{\mathrm{N}}$ is guaranteed to have at least one nonzero per column after the invasion step, but there might possibly be completely empty rows that represent unclaimed nets. This is not a problem unless there are some fine vertices that are connected to those unclaimed nets only. In that case, we add those orphaned fine vertices to our set of seeds, but they won't have any fine vertices attached to them as they have not invaded any nets, i.e. they form their

$\mathbf{P}_C^N : \mathbb{R}^{|N|\times|C|}, \mathbf{P}_N^F : \mathbb{R}^{|F|\times|N|} = \textsc{Interpolate}(\mathbf{A} : \mathbb{R}^{|N|\times|V|}, \mathbf{seeds} : \mathbb{R}^{|C|\times 1})$

```
1  ncoarse ← ExcNetCoarseness(A)
2  for (cvtx ∈ seeds)
3      do nnets = NeighborNets(cvtx)
4         fnnets ← Bottom(ncoarse(nnets), INVLIMIT)
5         P_C^N(fnnets, cvtx) ← 1./fnnets
6  seeds = [seeds; EmptyRowIndices(P_C^N)]
7  for (fvtx ∉ seeds)
8      do nnets = NeighborNets(fvtx)
9         cnnets ← Top(ncoarse(nnets), MAXNEIGH)
10        P_N^F(fvtx, cnnets) ← 1./cnnets
```

Fig. 3.5. *The algorithm to determine the rules of interpolation*

own aggregates. Our experiments show that less than 5% of all vertices are orphaned on the average.

Next, we have to determine $\mathbb{N} \Rightarrow \mathbb{F}$ mapping. We do this by examining the fine vertices and letting each fine vertex choose `MAXNEIGH` of its coarsest neighboring hyperedges. Both loops that determine mappings can easily be vectorized or implemented in an embarrassingly parallel way because there are no dependencies in subsequent iterations. In each iteration, we operate on a different column of the corresponding matrix.

Both `Top`($\mathbf{v}$, p) and `Bottom`($\mathbf{v}$, p) functions can be implemented to run in near linear time by using a priority queue of size of $p + 1$. In the case of `Top`, we use a min-heap and insert the first $p + 1$ elements. For the rest of the elements, we do an extract-min operation before we insert a new element. This way, we always keep the top p elements in the heap. This runs in $O(n \log p) \approx O(n)$ time, because p is a very small constant such as 3 or 4.

3.3. Setting-up Coarse Connections. Setting up net (hyperedge) connections for the coarse hypergraph is arguably the most important and the most computationally demanding phase. The inputs of this phase are the mapping sparse matrices $\mathbf{P}_C^N$ and $\mathbf{P}_N^F$. For example, assuming that each seed can invade at most two nets, the mapping matrices of the hypergraph in Figure 2.1(a) are

$$\mathbf{P}_C^N = \begin{pmatrix} 0 & 0 & 1/2 & 1/2 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 \end{pmatrix}, \mathbf{P}_N^F = \begin{pmatrix} 1/2 & 1/2 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

For each net in the fine hypergraph, we need to find its *uncertain connections*. If a net is connecting an aggregate with probability less than 1, then it is said to have an uncertain connection to that aggregate. For example, in Figure 2.1(b), N_1 has uncertain connections to both A_1 and A_2, with probability $1/2$ each. We will come back on the details of finding those uncertain connections, but for now, assuming that they exist, the question is how to treat them.

For reasons explained in Section 2.2, it is computationally infeasible to split the net to capture the semantics of those uncertain connections. We propose two different solutions to overcome the computational problem.

So far, we have been keeping the probabilities and postponing the decisions in order to capture the global picture better, at the expense of increasing complexity. At some point (usually during refinement), however, an indicator random variable needs to be drawn to realize the actual connection. The RSA algorithm makes this realization for edges early in the algorithm (during coarsening) to keep the complexity manageable while preserving the cut size in a stochastic manner. Concretely, for each uncertain connection with probability p, we generate a real random number $r \in [0.0, 1.0)$ and keep that connection if and only if $r \leq p$. This way, we avoid any combinatorial explosion on the number of nets of the coarser hypergraph.

On the other hand, the AWA algorithm just approximates the edge weights by keeping the weight of a net intact in the coarser graph, yet having it connect all the aggregates that contains at least a fraction of its pins from the fine graph.

After interpolation, to actually determine uncertain connections of a net N_i, we first determine its fine pins, $\{pins(N_i) \in \mathbb{F}\}$, that are not members of an aggregate whose seed N_i connects. This set is likely to be composed of fractional values since some fraction of a fine pin may be a member of an aggregate whose seed is connected by N_i whereas the remaining fraction may be a member of an aggregate whose seed is not connected by N_i. This uncertainty set is found by:

$$\mathbf{Uset}(N_i) = \texttt{ones}(\mathbf{A}(N_i, \mathbf{nonseeds})) - \mathbf{P}_{\mathrm{N}}^{\mathrm{F}}(:, N_i)^T \tag{3.2}$$

We can treat the whole set of **Uset** values as a matrix where the N_i^{th} row gives information about the fine vertex fractions that N_i has to connect. For example, for the hypergraph in Figure 2.1(b), **Uset** is:

$$\mathbf{Uset} = \begin{pmatrix} 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \Rightarrow \mathbf{Uset}' = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Our two methods differ in the way they treat the **Uset** matrix. While RSA draws an indicator random variable for each fractional value, AWA rounds up all nonzero values to 1. They both output a binary matrix **Uset′** after that stage, yet they output essentially different matrices (the example above shows a possible **Uset′** using RSA).

More concretely, both algorithms construct $\mathbf{A}^c$ by scanning pins of nets. For every pin of the currently examined net N_i in $\mathbf{A}$:

(a) If it is a seed vertex, just add it to $pins(N_i)$ in $\mathbf{A}^c$
(b) If it is a nonseed vertex, say v_i, first check if it has been determined before. If not, then determine the row $\mathbf{P}_{\mathrm{N}}^{\mathrm{F}}(v_i, :)$.

The determination is done by rounding up to zero in the case of AWA and by drawing a random variable in the case of RSA. The result of the determination gives us a set of coarse

nets. We ignore N_i if it is a member of that set. For every other member net $N_{j\neq i}$, we find the set of seed vertices that invades it by looking at $\mathbf{P}_C^N(N_j, :)$. AWA adds every seed in $\mathbf{P}_C^N(N_j, :)$ to the set of $pins(N_i)$ in $\mathbf{A}^c$, whereas RSA again draws a random variable to determine whether that aggregate should be connected by N_j. In order to perform constant time membership queries, we use a dense vector of size $|seeds|$. It is also important to mention that we do not determine $\mathbf{P}_C^N$, i.e. we will draw another random variable next time the same row of $\mathbf{P}_C^N$ is examined. This is done to prevent RSA to degenerate into net (hyperedge) coarsening. $\mathbf{P}_N^F$ and $\mathbf{P}_C^N$ matrices should be represented in CSR format since we only access them by rows.

4. Conclusions and Future Work. We proposed two new hypergraph coarsening algorithms, both of which use the new aggregative hypergraph coarsening paradigm. Our first method uses randomization to lower the computational costs of hypergraph coarsening whereas the second one relies on weighted aggregation with approximate net weights. We are currently implementing prototypes for our methods so that they can be evaluated in a multilevel framework. Our experiences from the implementation phase are likely to suggest minor modifications to our algorithms.

Acknowledgements. We wish to thank Bruce Hendrickson, Ümit Çatalyürek, and Cedric Chevalier for stimulating discussions throughout the period of this work. This work was funded by the US Department of Energy's Office of Science through the CSCAPES Institute and the SciDAC program.

REFERENCES

[1] C. J. Alpert and A. B. Kahng, *Recent directions in netlist partitioning: A survey*, Integration: The VLSI Journal, 19 (1995), pp. 1–81.

[2] R. H. Bisseling, *Parallel Scientific Computation: A Structured Approach Using BSP and MPI*, Oxford University Press, 2004.

[3] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A multigrid tutorial: second edition*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[4] U. V. Catalyurek and C. Aykanat, *Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication*, IEEE Transactions on Parallel and Distributed Systems, 10 (1999), pp. 673–693.

[5] K. Devine, E. Boman, R. Heaphy, R. Bisseling, and U. Catalyurek, *Parallel hypergraph partitioning for scientific computing*, in Proc. of 20th International Parallel and Distributed Processing Symposium (IPDPS'06), IEEE, 2006.

[6] R. D. Falgout and U. M. Yang, *hypre: A library of high performance preconditioners*, in ICCS '02: Proceedings of the International Conference on Computational Science-Part III, London, UK, 2002, Springer-Verlag, pp. 632–641.

[7] C. M. Fiduccia and R. M. Mattheyses, *A linear-time heuristic for improving network partitions*, in DAC '82: Proceedings of the 19th conference on Design automation, Piscataway, NJ, USA, 1982, IEEE Press, pp. 175–181.

[8] M. Gee, C. Siefert, J. Hu, R. Tuminaro, and M. Sala, *ML 5.0 smoothed aggregation user's guide*, Tech. Rep. SAND2006-2649, Sandia National Laboratories, 2006.

[9] B. Hendrickson and R. Leland, *A multilevel algorithm for partitioning graphs*, in Supercomputing '95: Proceedings of the 1995 ACM/IEEE conference on Supercomputing, New York, NY, USA, 1995, ACM, p. 28.

[10] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, *Multilevel hypergraph partitioning: applications in VLSI domain*, IEEE Trans. Very Large Scale Integr. Syst., 7 (1999), pp. 69–79.

[11] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, John Wiley & Sons, Inc., New York, NY, USA, 1990.

[12] E. Ramadan, A. Tarafdar, and A. Pothen, *A hypergraph model for the yeast protein complex network*, IPDPS, 10 (2004), p. 189b.

[13] D. ren Liu and S. Shekhar, *Partitioning similarity graphs: A framework for declustering problems*, Information Systems Journal, 21 (1996), pp. 475–496.

[14] I. Safro, D. Ron, and A. Brandt, *Graph minimum linear arrangement by multilevel weighted edge contractions*, J. Algorithms, 60 (2006), pp. 24–41.

[15] L. G. Valiant, *A bridging model for parallel computation*, Commun. ACM, 33 (1990), pp. 103–111.

[16] D. Zhou, J. Huang, and B. Schölkopf, *Learning with hypergraphs: Clustering, classification, and embedding*, in Advances in Neural Information Processing Systems 19, B. Schölkopf, J. Platt, and T. Hoffman, eds., MIT Press, Cambridge, MA, 2007, pp. 1601–1608.

PROBLEM-SPECIFIC CUSTOMIZATION OF (INTEGER) LINEAR PROGRAMMING SOLVERS WITH AUTOMATIC SYMBOL INTEGRATION

NICOLAS L. BENAVIDES *, ALESSIO CAROSI †, WILLIAM E. HART‡, VITUS J. LEUNG§, AND CYNTHIA A. PHILLIPS‡

Abstract. We describe the Solver Utility for Customization with Automatic Symbol Access (SUCASA), a mechanism for generating (integer) linear programming solvers derived from PICO that integrate algebraic modeling constructs. SUCASA allows application developers to access parameters, constraints, and variables from the application algebraic model within PICO. This allows rapid development of problem-specific incumbent heuristics and cutting planes. We briefly describe SUCASA and illustrate its use in two applications: generating graphs with specific degree sequence and scheduling the movements of mobile data collection units in wireless sensor networks.

1. Introduction. Algebraic Modeling Languages (AMLs) are essential tools for solving complex, large-scale optimization applications. AMLs are high-level programming languages for describing and solving mathematical problems, particularly optimization-related problems [10]. AMLs like AIMMS [1], AMPL [3, 8] and GAMS [9] have programming languages with an intuitive mathematical syntax that supports concepts like sparse sets, indices, and algebraic expressions. They provide a mechanism for defining variables and generating constraints with a concise mathematical representation, which is essential for large-scale, real-world problems that involve thousands of constraints and variables.

AMLs interface to solvers to analyze algebraic models. Typically, an AML system translates a problem into a standard format, such as a matrix-based representation for a mixed-integer linear program (MILP). The solver returns solution information that the AML system then makes available to the user within the model. For example, AMPL communicates to solvers via files, so it will work with any solver that can read and write files in the format AMPL expects. AML systems like ILOG's OPL are tightly coupled to a single solver.

For integer programming problems, a developer may know of some exploitable problem structure that would lead to more efficient ways to find heuristic solutions, better cutting planes, etc. It would be convenient for the developer to write such problem-specific code using the variables, parameters, sets, etc, from the model. However, we are aware of no AML system that provides this facility. Some systems provide callbacks from an API in C++, JAVA, or other language to augment the solver (e.g. to add a cutting plane). However, the user must know the location of their variables in the solver's linear variable order. For example, the user's model might have two types of variables, a two-dimensional variable y and a three-dimensional variable z. To write a cutting plane, the user must know that $y_{3,20}$ is variable x_{1003} and that $z_{1,24,541}$ is variable x_{246981}.

In this paper, we describe an example of a tool that makes symbols from an AML model available to a MILP solver. This allows users to customize the search within the solver using their problem's natural representation. Specifically, we describe how we automatically customize the PICO MILP solver for AMPL models. PICO's software library uses a flexible C++ class hierarchy for MILP solvers which allows users, and hence SUCASA, to extend PICO's MILP solvers through class inheritance. Though we use an AMPL solver as part of SUCASA, we are working toward a system that provides this capability using only open-

*Santa Clara University; NBenavides@scu.edu

†Dipartimento di Informatica, Università di Roma "La Sapienza"; carosi@di.uniroma1.it

‡Sandia National Laboratories, Discrete Math and Complex Systems Department, PO Box 5800, Albuquerque, NM 87185; wehart,caphill@sandia.gov

§Sandia National Laboratories, Computer Science and Informatics Department, PO Box 5800, Albuquerque, NM 87185-1318; vjleung@sandia.gov

source tools. Though SUCASA currently supports only AMPL, the methodology we describe could be adapted to other AMLs, as well as other solvers that support an extensible solver interface.

Given an AMPL model, the Solver Utility for Customization with Automatic Symbol Access tool automatically derives new problem-specific classes for the major PICO search-related classes and creates a driver routine. The new classes include definitions of the AMPL parameters, constraints, and variables. Specifically, the new classes can automatically map from an AMPL constraint name to a constraint number and from an AMPL variable name (of currently up to 7 dimensions) to a variable number.

The main impact of SUCASA is that it enhances a user's ability to customized PICO to leverage application-specific features. We describe two applications that illustrate the impact of this capability. Both use SUCASA to create cutting planes for separating exponential-sized families of constraints. The first is a graph generation problem where the separation algorithm is a customized C++ code. The second is a wireless sensor network management problem where the separation algorithm is an integer program.

The remainder of the paper is organized as follows. In Section 2, we give a simple knapsack example to show the simple syntax of using AMPL symbols in a SUCASA-derived C++ class. In Section 3, we describe the steps SUCASA takes to build the customized solver and describe how a user invokes SUCASA. Sections 5 and 6 summarize our example applications. Finally Section 7 offers a few final remarks.

2. Generating a Solver. In this section, we give a simple example to illustrate how a user can reference AMPL symbols within the C++ classes that SUCASA generates. To begin the user must have either created or prepared both a *.mod and a *.dat file.

We use the AMPL model knapsack.mod:

```
param n;
param capacity;
param value{1..n};
param weight{1..n};
var y{1..n} binary;
maximize obj: sum{i in 1..n} value[i]*y[i];
subject to constraint:
  sum{i in 1..n} weight[i] * y[i] <= capacity;

# SUCASA SYMBOL: n capacity value weight
```

Suppose the user, given a fractional solution to this model, wants to add some cuts to enforce additional constraints, and suppose that generating a new cut requires computing the sum of the weights of the elements that are at least half-selected by the solution. If the current fractional solution is in the vector called solution, the following code will compute the required sum:

```
int sum_some;
for (int i = 0; i < n(); i++)
  if (solution[y(i)] > 0.5)
    sum_some += weight(i);
```

The user can access the value of parameter n via the function call $n()$, and the value of the ith weight via weight(i). The call $y(i)$ returns the variable number of y_i. SUCASA also provides iterators for stepping through all valid indices of a variable or a parameter.

3. Customizing a Solver. PICO's **sucasa** command coordinates the generation of a customized PICO MILP solver. SUCASA invokes AMPL to generate a model instance and data-related files needed at runtime. It then post-processes the AMPL information to generate customized C++ PICO software. The user then compiles the new solver. This section involves accessing symbols, cutting planes, incumbent heuristics, and custom parameters.

In this section, we describe this process in more detail. With the version we describe here, the user must rerun **sucasa** and recompile the few application-specific files to run the model with a new data set. We have a different version that creates an executable that can handle arbitrary data files at runtime, but that does not yet support parameter export; it only makes constraints and variables available to the user for customization.

To begin the process, the user creates an AMPL model and data file. If the user created a data file knapsack.dat corresponding to the knapsack.mod model, he can invoke the **sucasa** command via:

```
scripts/sucasa -m --acro=\$HOME/acro-pico -g knapsack.mod
```

This will first create the `knapsack.map` file which is used to store sets and values. If the `*.map` file already exists then the user can skip to the next step. The next step utilizes **sucasa** to run the command.

```
sucasa -k --solver-options="--debug 10" knapsack.mod knapsack.dat
```

This launches AMPL to generate an MPS file for the MILP problem and model information that is used to customized PICO. In the simplest case, the model information is simply the row and column labels of the MILP problem. However, a user often needs additional problem data like the sets and parameters AMPL used to generate the MILP problem. A few key commands should be noted:

- `-h, --help` - Display the help message and exit.
- `-k, --keepfiles` - Keep temporary files.
- `--acro=ACRO` - The directory of the acro installation that will be used to build the customized PICO optimizer.
- `-m, --generate-mapfile` - Create the mapfile, even if it already exists.
- `--solver-options=SOLVEROPTIONS` - Options for the solver run by SUCASA.
- `-g, --generate` - Generate the customized IP source files.

There are other possible commands that can be using by using the `-h, --help` command.

The **sucasa** command interprets lines in an AMPL model file that contain `SUCASA SYMBOL` as declarations of parameter data to be exported from AMPL.

For example, the model `knapsack.mod` contains the line:

```
# SUCASA SYMBOL: n capacity value weight
```

to request that values for the symbols `'n'`, `'capacity'`, `'value'`, and `'weight'` be available in the customized PICO solver. In addition, one could use the line:

```
# SUCASA SYMBOL: *
```

to import everything. To date we have not needed the capability of exporting sets, as parameters are almost always sufficient and there are ways to make set data available through extra, artificial parameters. It should be noted that there is an issue with ambiguity in set supersets because the parser may interpret sets as literals. There are efficiency advantages in the `*.mod`

file if a user defines a set as a set over integers.

In our example, SUCASA creates the following files in this step:

- `knapsack.mps` - The MPS file for this problem.
- `knapsack.row` - Labels for the constraints and objectives.
- `knapsack.col` - Labels for the variables.
- `knapsack.val` - Values for parameters explicitly exported by **sucasa**.

The **sucasa** command then executes COOPR's command to process the AMPL output files. This program parses the variable and constraint names to identify their index sets, and infers relationships between these sets if the indices are symbolic. If index sets are integer, it cannot in general tell if index sets are shared for all instances or just coincidental in the current instance. So it assumes that all integer index sets are unique. At this point it will search for a user generated `knapsack.map` file. If one does not exist it then generates the file `knapsack.map`, which summarizes the model parameters, variables and constraints.

For reference, `SUCASA` uses an efficient indexing scheme to map tuples from param to values, constraints to row numbers, and variable symbols to variable numbers.

In our example, the `*.map` file contains:

```
Set0 ( INT ) ;
Set1 ( INT ) ;
Set2 ( INT ) ;
n ;
capacity ;
value [ Set0 ] ;
weight [ Set1 ] ;
constraint ;
y [ Set2 ] ;
```

The `value`, `weight` and `y` logically use the same index set. The user can edit this map file to indicate this. For example, they can declare `Items (INT)` and then specify `value[Items]`. Because `sucasa` does not regenerate the map file if it exists, this allows more efficient code that shares a single index set.

4. Running a Solver. The `sucasa` code generates classes in the `example_info.h` and `example_info.cpp` files that encapsulate the model data. Every parameter, constraint name, and variable in the map file is exposed in the `example::MILPSymbFunctions` class. These classes contain methods to support the flexible use of this data. For example, the following data and methods are available for variable 'y':

```
//
// Returns true if the given tuple or index is valid
//
y_isvalid( tuple )
y_isvalid( index )
//
// Returns the index of variable y for the specified index
//
y( tuple )
y( index )
//
// Returns the set of valid indices for B
```

```
//
y_valid()
```

The methods for parameters like 'weight' are similar except that function `weight(int i)` returns the value of the *i*th weight. Because by default, SUCASA invokes AMPL's preprocessor, there may be "holes" in the index set the user specified in the model file. That is, AMPL may have removed some variables in preprocessing because it inferred the variables' values. The various methods for checking for valid elements, or the iterators over the valid indices, allow the user to avoid referring to non-existant variables.

The **sucasa** command generates code for derived PICO classes that integrate these model data objects. Specifically, it generates the following PICO classes:

- **Problem** - The problem instance solved by the MILP solver
- **MILP** - Coordinates the branch-and-bound process
- **MILPNode** - Defines a node of the branch-and-bound tree; computes bounds and applies incumbent heuristics
- **parMILP** - Coordinates the parallel branch-and-bound process
- **parMILPNode** - parallel equivalent of MILPNode

These classes inherit from corresponding PICO classes, as well as the `MILPSymbFunctions` class. Further, instances of these classes share an instance of the underlying model data. Consequently, the model data and methods described above are available throughout the derived MILP solver. To summarize the `MILPSymbFunctions` class is an object of `Symbinfo` and the PICO classes summarized above are objects of `MILPSymbFunctions`.

The **sucasa** command constructs a makefile that leverages the PICO makefiles to support the construction of the derived MILP solver. It also provides a method for cleaning up files that **sucasa** generates. After executing **sucasa**, the derived solver can be built with:

```
make
```

This generates an executable `knapsack_milp`. The user runs this solver by calling

```
knapsack_milp [PICO options] knapsack.mps
```

Since this is a PICO solver, it recognizes all of PICO's options.

5. Application Example: Graph Generation. Our first example is a graph generation problem, motivated by Li et al's [12, 2, 7] model for the structure of some infrastructure networks. Those who study link prediction, for example, would like to generate graphs with such structure, or closely related.

The specific problem is as follows: given a number of nodes n and vector of vertex degrees d_i (for $i = 1 \ldots n$), construct a connected graph with the given set of vertex degrees that maximizes the sum of the edge degree products, $d_i d_j$, over all selected edges e_{ij}. Li et al give a heuristic solution for this problem.

We use decision variables c_{ij} that are 1 when e_{ij} is selected and zero otherwise. The integer program for this problem is straightforward except for the exponential number of constraints required to enforce that the graph is connected. However, there exists a separation algorithm to enforce that the graph is connected. We compare a shell script solution using AMPL, CPLEX, and a C program [11] implementing the separation algorithm with a SUCASA-based solution.

In the scripted AMPL, CPLEX, and C program solution, we first use AMPL and CPLEX to solve a base linear program with only the degree constraints. We then use a C program to find the connected components, where we consider two nodes connected if the edge joining

them is sufficiently highly selected. We then add constraints as necessary to require a (fractional) total of at least one edge leaving each connected component and a (fractional) total of at least enough edges to create a spanning tree between all connected components. We iterate until we (fractionally) have a single connected component.

We then use the Concorde [5] cut library to ensure that the global minimum cut is at least one. We add constraints as necessary in this second phase of the separation algorithm. We iterate through the combined two phases of the separation algorithm until the global minimum cut is at least one.

At this point, we may still have fractional edges. In our initial scripted prototype, we cannot use CPLEX's branch and cut mechanism to find an optimal integer solution because each linear programming relaxation requires the separation mechanism. We could potentially do this if we wrote a C or C++ code to use CPLEX's call back mechanisms and managed variable mappings.

The SUCASA solution avoids the file I/O and cold start of CPLEX after the base solve. The SUCASA solution is four times faster than the scripted solution. In addition to these speed gains, the SUCASA solution is easier to implement and can implement branch and cut.

6. Application Example: Scheduling Data Collector Movements in Wireless Sensor Networks. In the second application example, we consider a problem in managing wireless sensor network using Basagni et. al.ś method [4]. Unattended wireless sensor networks monitor for events of interest in military, environmental, industrial, and household settings. The sensors are small, usually disposable, and static (i.e. they do not move). They function only until their batteries run out of power. Sensors send packets with status or event information to one or more *sinks*. Sinks are not energy constrained. They can move, compute, and can communicate to a central data collection point. Sensors can only communicate within a fixed tranmission radius. They relay packets through other sensors until the packets reach a sensor that can communicate directly to a sink. Relaying packets requires energy. Because nodes near the sink must relay packets from every sensor in the network, they use energy at a greater rate than those more distant from the sink. Thus if the sinks did not move, nearby sensors would die quickly, disconnecting the sink, while other sensors still had plenty of battery power. The problem, which we will not describe formally, is to schedule movements of the sinks to balance energy consumption, maximizing network lifetime, which is the time between network deployment and the death of the first sensor.

Basagni et. al. [4] solve a linear-programming relaxation of the scheduling problem. The LP selects a set of configurations (placements for the sinks on a subset of legal sink locations) and selects a time for each configuration to hold. But it does not handle movement between configurations and other technical details. Basagni et. al. use the LP solution to find a feasible solution that is always within 1.4% of optimal for the realistic test cases they have run. The LP has a variable for each of the exponential number of possible configurations, however. Therefore, they solve the dual, using separation. The separation algorithm is a p-median-like integer program, which is tractable in practice for problem sizes consistent with current wireless sensor networks.

Basagni et. al. solved this LP using a PERL script [6] and AMPL with CPLEX. The PERL script driver calls AMPL iteratively, running the LP model and the IP separation model on each iteration. It creates the LP and IP ampl model files at each step. It interprets the IP solution, and, if there is a violated constraint, adds the new constraint to the LP for the next round.

This system leads to a considerable waste of time. At each step AMPL rebuilds the LP and IP matrices from scratch. CPLEX has to solve the LP from scratch each iteration even though the PERL script only adds a single constraint to the previous LP problem. Since that

work, we have reimplemented the system in AMPL only. AMPL does not support general data structures, and therefore cannot provide the full programming environment for separation algorithms that SUCASA does, but for this example, AMPL's looping mechanism is sufficiently powerful. In fact, the AMPL-only (plus CPLEX) implementation was extremely easy to write. Furthermore, since AMPL is aware of the iterations, it warm starts CPLEX on each LP solve using the previous basis. However, AMPL still rebuilds the LP matrix at each step.

Table 6 compares the PERL script with the AMPL-only solution for some test problems. These problems had 400 sensors with a $k \times k$ grid of feasible sink locations. There are s sinks. Each table entry shows the time in seconds and the number of iterations in parentheses. These results are for a 64-bit machine with two Intel Xeon CPU at 3.60GHz and 8GB of RAM, linux kernel *2.6.9-1.667smp* and CPLEX version 10.0.0.

(k,s)	PERL/AMPL	AMPL
(16,2)	4401.57 (46)	4951.97 (47)
(16,3)	9407.67 (66)	8956.41 (66)
(16,4)	12120.1 (73)	11219.1 (73)

TABLE 6.1
Solution time in seconds and number of iterations (in parentheses).

Although the AMPL-only solution is somewhat faster, a solution using SUCASA should be faster still, since it keeps a solver environment open, allowing it's own warm-starting decisions, and there is no need to rebuild the LP matrix at each step.

7. Discussion. Our experience with application examples demonstrates that developing applications with SUCASA can be quick and intuitive. Furthermore, these applications can solve problems faster than using scripts because PICO makes incremental changes to a solver environment that is always open. This avoids rebuilding matrices and allows the solver to warm start on the next iteration.

The standard PICO release includes an example SUCASA application: scheduling jobs on a single machine with precedence constraints. This example uses the derived classes to implement an incumbent heuristic.

The code that SUCASA generates includes useful stubs and comments for adding incumbent heuristics or separation algorithms.

Acknowledgements. We thank Jonathan Eckstein for his feedback on SUCASA and for help extending PICO to support this capability. We thank Jon Berry for his help with SUCASA support. We thank Randall Laviolette for suggesting the graph generation problem and Robert Carr for discussions about the graph generation problem. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

REFERENCES

[1] *AIMMS home page*. `http://www.aimms.com`.
[2] D. L. ALDERSON AND L. LI, *Diversity of graphs with highly variable connectitivity*, Physical Review E, 75 (2007).
[3] *AMPL home page*. `http://www.ampl.com/`.
[4] S. BASAGNI, A. CAROSI, C. PETRIOLI, AND C. A. PHILLIPS, *Coordinated and controlled mobility of multiple sinks for maximizing the lifetime of wireless sensor networks*. Submitted.

[5] *Concorde home page*. http://www.tsp.gatech.edu/concorde.html.
[6] H. M. Deitel, P. J. Deitel, T. R. Nieto, and D. C. McPhie, *Perl How to Program*, Prentice Hall, Englewood Cliffs, N.J., 2000.
[7] J. C. Doyle, D. L. Alderson, L. Li, S. Low, M. Roughan, S. Shalunov, R. Tanaka, and W. Willinger, *The "robust yet fragile" nature of the internet*, PNAS, 102 (2005), pp. 14497–14502.
[8] R. Fourer, D. M. Gay, and B. W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming, 2nd Ed.*, Brooks/Cole–Thomson Learning, Pacific Grove, CA, 2003.
[9] *GAMS home page*. http://www.gams.com.
[10] J. Kallrath, *Modeling Languages in Mathematical Optimization*, Kluwer Academic Publishers, 2004.
[11] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice Hall, Englewood Cliffs, N.J., 1978.
[12] L. Li, D. Alderson, W. Willinger, and J. Doyle, *A first-principles approach to understanding the internet's router-level topology*, in Proc. Conference of the ACM Special Interest Group on Data Communication, 2004.

Architecture and Systems Software

Articles in this section discuss advances in high–performance computing architectures and systems software that enhance performance of real–world scientific and engineering applications.

Reiss et al. address the challenges associated with the lack of scalability and reliability in I/O systems on modern supercomputers. One proposed solution has been to dedicate compute nodes as a staging I/O proxy, which acts as a large buffer and sends output during an application's compute phase. The authors present the design and implementation of a transparent wrapper for the parallel netCDF library that provides this service. *Martin et al.* present a detailed performance analysis of the SiCortex SC072 high–performance computing cluster. SiCortex SC072 is a balanced cluster which makes use of low–power MIPS processors and a custom interconnect in an effort to avoid many of the bottlenecks plaguing most modern systems. The results of the study indicate that, albeit outpaced by modern commodity clusters in terms of pure processing power per node, the SiCortex approach to high–performance computing leads to consistent scalability and significantly greater performance per watt. *La Fratta and Rodrigues* address the problem of intelligently utilizing on–chip cache resources in future generations of CPUs, with the objective to enhance both computational capabilities and memory access patterns. They present an unconventional approach of augmenting each cache level with distributed co–processors, which are utilized by forming small groups of computations. The approach is evaluated using an extension of the Structural Simulator Toolkit (SST) on a set of important Sandia applications. *Curry et al.* discuss the performance of a general Reed–Solomon encoding and decoding library that is suitable for use in RAID–like disk drive systems that utilize GPUs. Reed–Solomon coding is a method for generating arbitrary amounts of checksum information from original data via matrix–vector multiplication in finite fields. The authors generalize, broaden, and optimize a previously developed Reed–Solomon coding library and report encouraging performance results.

D. Ridzal
S.S. Collis

December 11, 2008

IMPLEMENTATION AND EVALUATION OF A STAGING PROXY FOR CHECKPOINT I/O

CHARLES A. REISS*, JAY LOFSTEAD†, AND RON. A. OLDFIELD‡

Abstract. As supercomputers have increased in processing power and decreased in reliability, their I/O systems are not scaling similarly. One proposed solution has been to dedicate compute nodes as a staging I/O proxy. These nodes act as a large buffer for output and send output during applications' compute phase without causing interference. This report presents the design and implementation of a transparent wrapper for parallel netCDF that provides this service. The wrapper's performance was evaluated on the Cray XT3 Red Storm. Based on the observed performance, we show that the wrapper should provide better overall performance for large checkpointing applications even after taking into account computation that could otherwise be done with the staging nodes.

1. Introduction. As new supercomputers have more and more cores and thus more and more processing power, reliability and I/O capabilities are not scaling similarly. Unfortunately, traditional supercomputing applications rely on I/O capacities scaling as complexity increases. The most common means of ensuring long-term progress is periodic checkpointing. But, as reliability decreases, more and more frequent checkpointing is required to obtain maximum utilization. When the I/O system also does not scale according to compute power, each checkpoint also takes longer. With increasing checkpoint overhead, the application only runs marginally better with the extra cores.

Without substantially changing the applications themselves, one potential workaround for the poor scaling of I/O is staging. Instead of communicating directly with the filesystem, applications communicate with a staging proxy. The staging proxy can sit on an arbitrary number of the plentiful compute nodes and be resized and adapted for particular application's needs. The proxy hides the overhead of the I/O system by buffering and consolidating the I/O requests. Buffering allows I/O to be overlapped with the application's computation phases, achieving similar effects to asynchronous I/O when it would not otherwise be possible. If the staging nodes are considered reliable enough, one can even stop viewing it as a proxy: instead they can be a fast, in-memory filesystem that supplements the more permanent filesystems.

We developed a staging application, which is a wrapper around the parallel netCDF library [8]. Our wrapper works with unmodified parallel netCDF applications by translating their library calls into RPC calls for the staging servers. The staging servers then buffer the data (if they have memory to do so) and perform the specified netCDF operations after the application has finished its I/O phase.

We demonstrate that for sufficiently large applications, our approach will improve the performance of unchanged applications which use parallel netCDF. The application size required is within what may be encountered on current supercomputers, and we anticipate that this approach will become more applicable to future machines. Although the staging nodes are unavailable for computation, the drastically decreased checkpoint times mean that applications should make more progress with the same overall number of nodes.

2. Design.

2.1. Platform Limitations.

Our staging servers ran entirely on normal compute nodes on Sandia's Cray XT3, Red Storm, which use the lightweight operating system Catamount [7]. This imposed some restrictions on our staging application: Catamount does not have support for threading or multiprocessing and the only available internode communication

*Georgia Institute of Technology, creiss@cc.gatech.edu
†Georgia Institute of Technology, lofstead@cc.gatech.edu
‡Sandia National Laboratories, raoldfi@sandia.gov

mechanism is through Portals [2], which provides an RDMA interface, and through a Portals-based MPI implementation.

We also limited ourselves to performing I/O using the parallel netCDF API, rather than handling the file format ourselves. On Catamount, this meant we had no asynchronous write support, so we had no way of overlapping writes with communication. The netCDF format itself has some limitations; for example, datatypes large than a byte require byte swapping, which uses extra buffer space and slows down the staging node's I/O phase. (Since we performed our tests with IOR [6], which only writes arrays of bytes, this was not necessary in our measurements.)

2.2. Parallel netCDF Semantics. The parallel netCDF library's API provides applications some guarantees that are not ideal for I/O performance. To obtain good I/O performance, we relaxed guarantees which we felt were unlikely to be relied upon in any real application.

Primarily, we relaxed the implicit assumption that when a file is closed, it has been written to non-volatile storage. We, however, provide a similar guarantee that, as long as staging nodes and storage remain up for the time it takes to write the file, it will be written to non-volatile storage. Even if the staging nodes are unreliable, at worst, the application will be an additional checkpoint behind. Since shorter checkpoint times (somewhat non-intuitively) decrease the optimal checkpoint interval, we would expect often to be recovering from a more recent version in spite of the added delay.

Many collective parallel netCDF operations act as barriers. Even an application that does not use collective reads or write could take advantage of these implicit barriers since opening, closing, and creating variable definitions are collective in parallel netCDF. Coordinating most operations is not useful for the staging nodes; it is more important that the staging nodes evacuate the application nodes quickly, and filesystem-related coordination can be delayed until the application's compute phase. So, while we can easily make these operations a barrier in our wrapper, doing so results in substantial performance degradation because the staging nodes cannot keep their pipeline of requests full.

We do not, however, entirely desynchronize collective operations. Writing out a file still acts as a barrier for all processes involved, which is all we believe that real applications are likely to assume.

Commonly used parallel netCDF write functions assume that memory regions passed to them cannot be accessed until after the call returns. Although there are many cases in practice when they can as the application is passing arrays that will not be modified until the next compute cycle begins, we cannot distinguish this situation from when a temporary buffer is passed. We avoided allocating buffer space on application nodes assuming that applications would not leave us with a substantial amount of memory. Parallel netCDF does not have any explicit contract about how much memory it uses internally, but besides header information, parallel netCDF buffer space does not remain allocated after a synchronous write call returns. If we could relax these assumptions, we could provide implicit asynchronous I/O, removing the overhead of waiting for staging nodes to finish fetching each datum before preparing or sending new data.

2.3. Staging Node Placement. To obtain full performance in data transfers from application nodes to staging nodes, the selection of staging nodes is important. The minimum bisection bandwidth of Red Storm is on the order of terabytes per second, larger than the interconnect-to-memory bandwidth of hundreds of nodes, and even the bandwidth of the physical links is larger than the interconnect-to-memory bandwidth. Thus, one might expect that the interconnect-to-memory bandwidth would determine the overall bandwidth we could provide, and the interconnect bandwidth itself would be largely irrelevant. Unfortunately, although there is plenty of overall interconnect bandwidth, the routing scheme is fairly static

and so cannot take into account changing communication patterns. When the routes for several pairs of nodes that are transferring data intersect, these shared interconnects thus limit the total bandwidth.

Figure 2.1 illustrates this problem. With the ratio between staging and application nodes increased, bandwidth can decrease substantially as the number of staging nodes is increased. We believe this results from network contention.

Naive allocations are much more susceptible: if one allocates the staging nodes and the application nodes as separate job invocations, the job launcher, by default, tends to allocate contiguous blocks of nodes as would be expected when intrajob communication speed is the primary concern. But with such an allocation, it is inevitable that there will be many shared links on paths between the application nodes and their corresponding staging nodes. Thus, as seen in the figure, performance decreases to half the maximum bandwidth with only 32 staging nodes (288 nodes in total).

We, however, achieved more consistent performance by spreading out the staging nodes. Using the MPI ranks of each node over the entire allocation, we assigned staging nodes to every kth rank (with appropriate rounding, setting k to the number of staging nodes divided by the total number of staging and application nodes). Since application nodes are assigned to staging nodes in groups of neighboring rank, this should ensure that the distance between application and their corresponding staging nodes is reasonable.

Even with this allocation policy, node placement was still responsible for most of the variance in bandwidth between application and staging nodes. We tested this by allocating segments of 600 compute nodes and choosing random 534 node suballocations from it, and observed the performance of the every kth rank selection on these suballocations. We had 22 staging nodes, the number required to buffer 32 MiB each from 1024 cores. Although performance of the each suballocation was consistent over time, we observed a great deal of variance between performance of different suballocations. Figure 2.2 illustrates the results of this test over 52 trials.

Running the same test with a single 534 node allocation yields very consistent performance (less than 5% difference in observed bandwidths) even over hours. Thus, the most variance observed in our tests was probably not caused by inconsistent interconnect performance or by interference from other applications on the machine.

FIG. 2.1. *Staging node input performance with default allocation versus random allocation of staging nodes.*

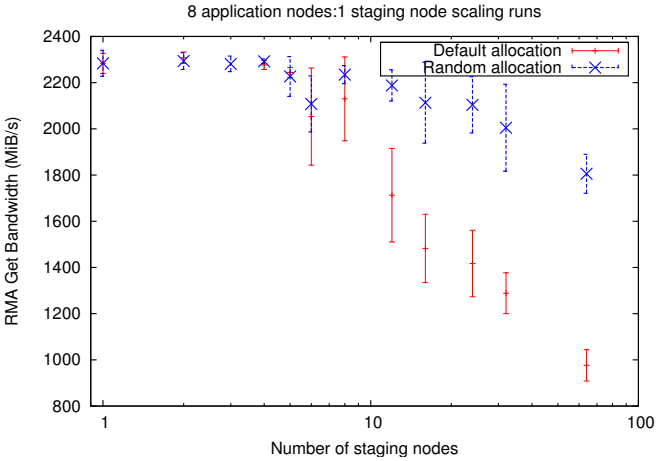
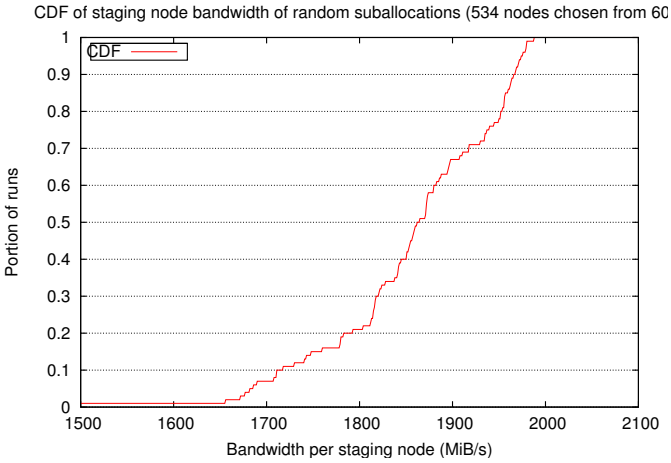


FIG. 2.2. *CDFs of performance of random suballocations of larger allocations.*



3. Evaluation.

3.1. Platform Characteristics. We tested our staging library and made performance predictions based on Red Storm, a Cray XT3 located at Sandia. At the time of testing, Red Storm had 12960 dual-core compute nodes. The compute nodes are arranged in a regular three-dimensional grid, connected with a hypertorus topology. Each node has an interconnect with a custom Cray SeaStar networking chip and an dedicated PowerPC chip. The interconnect is coupled to the processor using a HyperTransport link, which has a theoretical (excluding wire protocol overhead) bandwidth of 2.8GB/s [3]. Each of the six links from each node can support 2.5GB/s, after protocol overheads [1]. Low-level software access to the interconnects is provided through the Portals library [2], which provides a connectionless RDMA-based interface.

According to prior the maximum node-to-node unidirectional bandwidth through Portals is around 2.1GB/s [3], but in microbenchmarks, we observed that total bandwidth when receiving from multiple nodes simultaneously can exceed 2.3GB/s. Because all communication between compute nodes and service nodes takes place over these interconnects, our results could include some interference from other jobs.

All experiments detailed in this article that wrote to disk used `/scratch1` on Red Storm, a Lustre [9] version 1.4 filesystem which had 160 OSTs (Object Storage Targets) distributed over 80 OSSs (Object Storage Servers). The staging server's writes were performed through the real parallel netCDF library, which uses MPI-IO internally. The performance of writing through parallel netCDF and writing directly with POSIX or MPI-IO in tests with IOR were nearly identical on Red Storm. Based on past observed performance, the maximum I/O bandwidth to this filesystem should be around $176\text{MiB/s} \cdot 160 \approx 28\text{GiB/s}$ [5], but values observed in practice (except for carefully coordinated tests) are much lower.

We did not use MPI collective I/O for writes because our tests indicated that collective I/O consistently performed worse or no better than independent I/O on this filesystem. Except where otherwise noted, for all tests we set the stripe count of the directory where our file was written to the maximum and wrote one shared file. The stripe size was kept at the default of 2 MiB.

All I/O was done synchronously, as the Red Storm's MPI implementation does not sup-

port asynchronous I/O[1]. For staging I/O, as long as staging nodes had sufficient free memory, we prevented the latency of synchronous I/O from interfering with transfers from the application by delaying disk I/O until the application closed the file. Except when otherwise noted, the staging nodes did, in fact, have enough memory to buffer the data.

Although some nodes on Red Storm have 3GiB of memory, we tested as if all staging nodes had only 2GiB available. When running IOR [6] to gauge I/O performance to the staging nodes, one instance of IOR ran on each of the two cores of each application node. The staging nodes, however, always ran in single-core mode so that more memory was available to them.

3.2. Staging to Storage Performance. From our tests, it appears our staging node's I/O phase was competitive in speed with the speed directly from IOR with a similar number of clients, despite the different ordering and total I/O sizes. We found that Lustre generally gives much poorer performance for smaller (1 MiB) writes compared to larger (16 MiB) ones in our tests, even with large numbers of clients.

Although the bandwidth between staging nodes and storage was comparable to the direct bandwidth directly from IOR, Lustre did not in our tests provide full performance until it had on the order of 500 clients. When each application core writes 32 MiB, there are not nearly enough staging nodes in use to saturate the filesystem. We believe that it should be possible to achieve full performance with the number of nodes we used by carefully setting Lustre settings and with better arrangement of compute and staging nodes relative to their corresponding OSSs.

There is a great deal of variance in I/O performance to storage. Likely causes for this variance include competition with other jobs and the placement of our jobs relative to the OSSs. A more carefully designed test could allocate nodes in a manner that was aware of the OSTs and thus obtain better and more reproducible performance. However, past large I/O benchmarks on Red Storm that have controlled for these factors have observed considerable variance, with differences of well over 10% between maximum and average I/O performance [5].

3.3. Application to Staging Performance. The limiting factor in application to staging node transfers is the I/O bandwidth of the staging nodes. Under optimal conditions, RDMA fetches proceed at an aggregate average speed of 2200MiB/s, the maximum speed we have seen for such transfers even in small-scale tests. This speed is only achieved by overlapping at least two RDMA transfers. Under normal conditions, we measured the RDMA transfer speed at 1800 or 1900 MiB/s. Other overheads, most likely the round-trip latency between the server and the clients, can further reduce the effective bandwidth to 1600 to 1700 MiB/s, but this is as low as we have consistently observed with reasonable node allocations.

The net effective I/O bandwidth is much higher than that of the regular I/O system: for example, we observed effective bandwidth of more than 375GiB/s in a run with 249 staging nodes. Effective bandwidth begins exceeding what can usually be obtained with the I/O system directly with as few as 512 application cores writing 32 MiB each, a data size that the storage servers should be able to cache completely.

4. Conclusions.

4.1. Modeling Efficiency. Staging is only worthwhile if the compute resources used for staging would not be more productive as part of the application proper. Only when the

[1]MPI-IO's asynchronous I/O functions are provided, but tests showed that they were apparently synchronous. There is theoretically asynchronous I/O support in Catamount on Red Storm, for example using the Cray `iwrite` function, but we did not test this.

application is larger than the filesystem can easily handle and failures are relatively common, faster checkpoints are worth more than those compute resources. An important question is at what exactly size the benefits of the faster I/O outweigh the loss of the nodes.

To address this question, we need a model of how much work is accomplished in the presence of checkpointing and failures. Parameters of this model are:

- B_S, the bandwidth to each staging node
- B_F, the bandwidth of the filesystem
- M_1, the mean time to interrupt (MTTI) of a single node
- d_S, the amount of data a staging node can buffer
- d_A, the amount of data an application core will write
- C, the number of cores per node
- A, the total number of application nodes

For the parameters determined by the machine, we have low and high estimates for Red Storm from our experimental results:

Parameter	Low	High
B_S	1400MiB/s	1600MiB/s
B_F	4GiB/s	25GiB/s
d_S	1600MiB	1700MiB

Some useful derived quantities are:

- δ, the checkpoint time
- I, time after checkpoint completes before it is committed to disk
- τ, the checkpoint interval; optimal value $\tau_{opt} \approx \sqrt{2\delta(I + M + R)} - \delta$ (based on [4]; see 6)
- S, the number of staging nodes required
- $M = M_1/(A + S)$, the overall MTTI of the entire job (we assume node failures are independent)
- $d_{tot} = CAd_A$, the total size of the checkpoint

4.2. Evaluating Efficiency. Daly's derivation[4] of an approximation for the optimal checkpoint interval is obtained by minimizing an estimate of an application's time to completion that we will use here to estimate performance, modified to include a term for I. Under this model, every τ-unit long compute phase requires on average

$$T = \tau + \delta + \left(\frac{1}{2}(\tau + \delta) + R + I\right) e^{\frac{\tau+\delta}{M}}$$

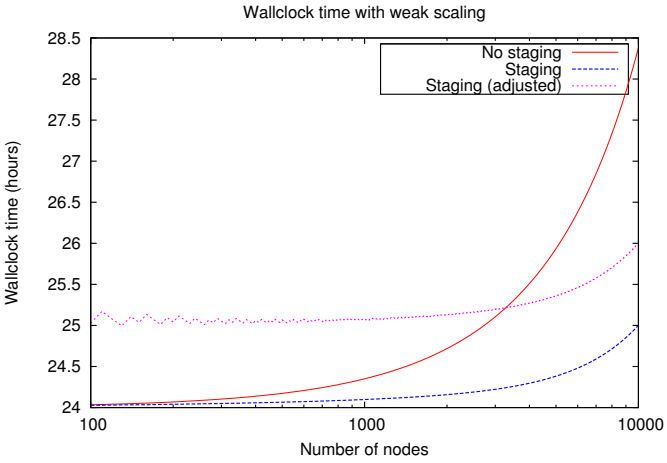
units of wallclock time. The latter term is the estimate rework time, assuming failures are a Poisson process.

We arbitrarily estimate that $R = 2d_{tot}/B_F$ (since we know that that recovery requires reading a checkpoint, we know that at least $R > d_{tot}/B_F$, the time it takes to write a checkpoint to disk).

The other inputs we can compute. With staging, $S = \lceil d_{tot}/d_S \rceil$, $\delta = S \cdot B_S$, and $I = d_{tot}/B_F$. Without staging, $\delta = d_{tot}/B_F$ and $S = I = 0$.

For example, consider an application where $d_A = 32$MiB and which takes one day of computation to complete. Its wallclock time with checkpointing overhead will be (1 d)T/τ. Figure 4.1 shows a scaling plot of its this wallclock time using our low estimates of Red Storm I/O performance. Execution time with an appropriate number of staging nodes added is easily better, but this is an unfair comparison: with staging, the application uses $S/A \approx 4\%$ more nodes, so we should adjust the staging wallclock time accordingly for comparision. The adjusted staging times in the figure are thus about 4% more than unadjusted staging time.

FIG. 4.1. *Scaling plot of a hypothetical application which needs 1 day of computation and writes 32 MiB per core (using low estimates of Red Storm performance). Adjusted staging numbers account for the extra nodes devoted to staging.*



With this adjustment, there is an application size below which staging is not useful because of the work that could be done with the extra nodes. We call this point the *crossover point*, and it determines when staging is or is not appropriate.

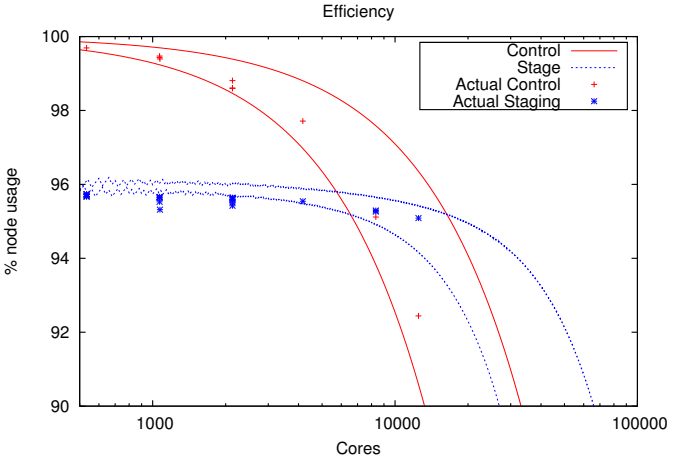
Using the adjustment for the number of nodes in use, we compute a quantity we call *efficiency*, which is $\frac{A}{A+S} \cdot \frac{T}{\tau}$. The efficiency is the portion of available compute time used for computing the application's result, so if an application did not have devote resources to outputting and did not experience failures, its efficiency would be 100%. Our computation of efficiency is motivated by an assumption that making small adjustments in the number of nodes devoted to an application does not substantially change work performed per unit compute time. Since most applications show slightly sublinear scaling at interesting sizes, we probably overestimate the number of nodes required for staging to be worthwhile.

In this model, the failure of any staging or application node requires a full restart. But, ideally, a failure of an application node would allow recovery from the staging nodes, avoiding any problems with the time disk I/O takes and waiting for a deferred write to disk to complete, and a failure of a staging node would allow recovery by restarting the staging nodes separately in between checkpoints. The case where low-overhead recovery would not be possible is when at least one staging node and one application node fail in close proximity. If we assume that all failures are single node failures and that each node fails according to a Poisson model, then such 'double failures' are relatively unlikely. In practice, however, problems with interconnects and other shared resources often cause multiple nodes to fail simultaneously.

The model also assumes that the data size per application node is fixed, regardless of the number of staging nodes. It might be more reasonable to assume that the problem size of the application should instead be determined by the total number of nodes. Nevertheless, staging is still advantageous under these assumptions if the amount of data per application node is determined by the total nodes allocated. Since this 'strong scaling' increases the total data size by what should be a relatively small amount — the proportion of nodes devoted to staging — the difference in the crossover point under these assumptions is small. A real application would, of course, have limits on how it can distribute the work and non-I/O scaling issues that would complicate this picture.

4.3. Crossover Points. From actual measurements, when each application core must write out 32 MiB in 1 MiB chunks, the crossover point lies somewhere between 6000 cores and 10000 cores. Figure 4.2 shows estimated efficiencies in this case with and without staging given our low and high parameters for our model in terms of the total number of nodes (staging and application). The figure also shows the efficiency estimates based on values of δ, I, A, and S we measured in actual runs. Staging runs for small numbers of nodes are below our model's estimates because not enough nodes are performing I/O to use the full bandwidth of the filesystem. Observed performance to Lustre being well less than its potential performance, using our high estimates of Red Storm's performance puts the crossover point near 16000 cores.

FIG. 4.2. *Efficiency of a linearly scaling application for 32 megabyte writes in 1 megabyte transfers, shown with efficiency values from measured I/O times. Staging performance is less than the model predicts for small runs because there are not enough staging nodes to achieve reasonable I/O performance to Lustre. Sawtooth patterns in estimates of staging performance for small runs are due to rounding when choosing the number of staging nodes to allocate.*

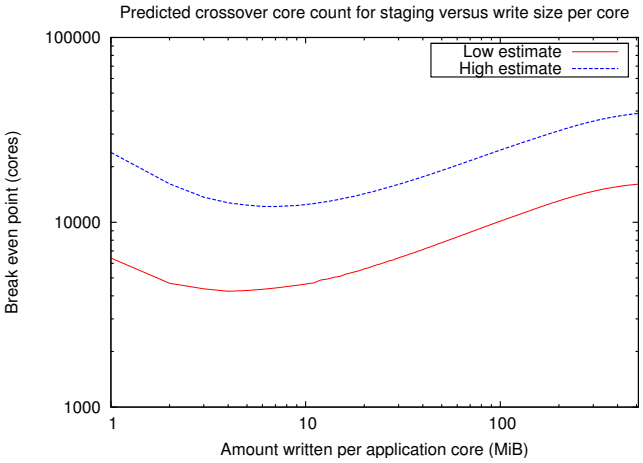


For larger write sizes, the crossover point occurs at a much higher number of nodes as shown in figure 4.3. For example, if each application core writes 128 megabytes, we predict the crossover point to be somewhere between 10000 and 25000 cores under the same assumptions. This is since the portion of nodes that need to be devoted to staging that buffers the write completely is approximately the same as the portion of memory that is written out. Fortunately, staging I/O scales well enough that relatively little time beyond the fixed cost of the lost nodes is lost, so we can expect staging to be practical for most I/O sizes.

4.4. Varying Checkpoint Intervals. With staging, the computed optimal interval between checkpoints can be very small. A small checkpointing time δ implies a small optimal checkpoint interval, as can be seen from the formula $\sqrt{2\delta(I + M + R)} - \delta$. This may seem non-intuitive, but it is natural that when doing a checkpoint is not a burden, one can gain more from making them more often. However, these small checkpoint intervals may be difficult to achieve in practice as the time spent in the compute cycle cannot be precisely adjusted. To estimate the effects of this imprecision, we examined the estimated efficiency after varying the checkpoint time away from the optimum.

In most cases, the optimal checkpoint interval is several minutes with staging, but an order of magnitude larger without. Fortunately, for non-small data sizes, the benefit lost by exceeding the checkpoint time is relatively small. Figure 4.4 shows how changing the

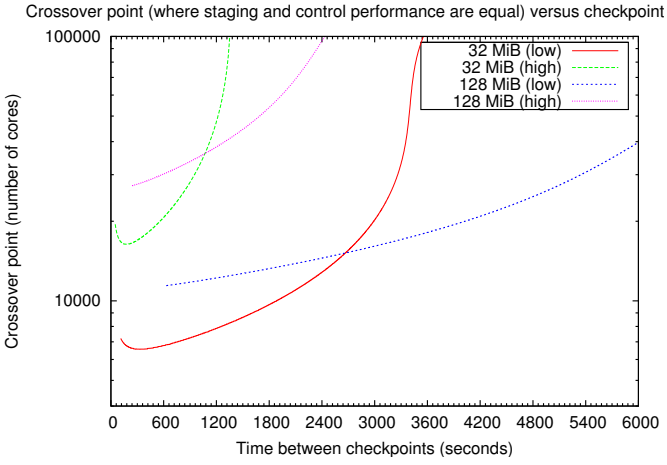
FIG. 4.3. *Crossover points when each core writes different amounts.*



checkpoint interval in the staging case affects the crossover point in some example cases. With small data sizes not only is the optimal checkpoint interval exceptionally small to begin with, but deviating from it has a more severe effect. With larger data sizes, the optimal checkpoint interval is several times larger, determined by the output time to disk already and relatively large deviations from it have relatively little effect.

For example, when each core writes 128 MiB with our model of Red Storm, we estimate that though the optimal checkpoint interval is between 5 and 10 minutes, using an interval of 20 minutes increases the number of nodes required to benefit from staging by less than 40%. With smaller data sizes, the optimal checkpoint times are smaller, close to 3 minutes for 32 MiB written per core. Unsurprisingly, the effect of increasing the checkpoint time is much greater; in the 32 MiB per core case, the number of nodes required to benefit increases by somewhere between 25% and 250% (depending mostly on how fast the normal filesystem is).

FIG. 4.4. *Change in crossover point as the checkpoint interval for staging is changed. No data is provided at the checkpoint intervals below the actual I/O time to the filesystem.*



5. Future Issues.

5.1. Large Memory Nodes. Modeling the efficiency of staging can estimate the expected improvement of potential hardware improvements. To support staging, one obvious hardware improvement is increasing the amount of memory per node, possibly only in selected "jumbo" nodes. To test the effects of such an improvement, we performed scaling tests with small memory (2GiB) nodes, which were configured to retrieve more data than they had memory for, silently discarding the excess data (producing a corrupt file) rather than stalling to write the data. These tests showed that staging nodes gave consistent I/O performance even when their number of clients increases dramatically.

The availability of such 'jumbo' nodes would substantially increase the viability of staging. For example, we estimate that given a sufficient number of nodes with 8GiB of memory, the break-even point when 128 MiB was written per core would not be between 10000 and 25000 cores but between 3000 and 8000 cores. Given that it may be easier and cheaper to supply such jumbo nodes than to provide a traditional parallel filesystem with appropriately scaled speed, large memory nodes scattered throughout a supercomputer may be an good way for future supercomputers to scale their I/O systems.

5.2. Proxies for Caching. When staging nodes do not have enough memory to hold the entire dataset, staging is theoretically still useful because it provides an effectively faster I/O system. Tests done when staging nodes only had enough buffer space to buffer two-thirds of the data show that this can provide better performance. These tests were small scale, run with 512 cores feeding into 27 staging servers which were writing to a Lustre file limited to 8 stripes. These provided an effective I/O speed of around 3000 MiB/s instead of the actual filesystem speed of 1000 MiB/s. As we expected, with memory constrained the actual speed of the filesystem was the limiting factor and not the speed at which staging nodes can read.

Extrapolating and assuming linear scaling and 5 year/node MTTI, on full-scale runs, we would expect this sort of I/O speedup to become worthwhile somewhere between 8 and 20 thousand application cores, but we did not get an opportunity to run experiments to demonstrate scaling to these sizes.

5.3. Combining Writes and Asynchronous I/O. With our architecture, we could easily support asynchronous I/O from the staging servers to the filesystem if the underlying MPI-IO supported it. We would expect asynchronous I/O would overcome the problem of poor performance due to small writes as we would effectively always be writing as much as possible. Portals on the XT3 also has good performance with simultaneous puts and gets, so at least on the XT3, we may be overlap filesystem I/O with transfers to the servers, decreasing the effective I/O time and interference with applications' communication phases.

Even without asynchronous I/O support, we should be able to improve performance by performing larger writes. Since with our current arrangement, each staging node is likely to have data that is adjacent in the application's conceptual space and thus in the resulting file, they should be able to combine the application's writes into a larger contiguous write. Potentially, these writes could even be coordinated so the staging servers would spread the load intelligently across the servers of the parallel filesystem.

5.4. Application Interference. One unanswered question about staging is how the staging I/O phase will interfere with the compute phase of the application. If the staging servers are performing large transfers to the filesystem over the same network connections as the application uses for internal communication, we can expect diminished performance where these phases overlap. This is especially true as our placement of the staging nodes interleaves them with application nodes. This effect is likely to be highly dependent on application com-

munication patterns and how much of the interconnect bandwidth is required to saturate the I/O system.

6. Modified First-Order Approximation of Optimal Checkpoint Interval. We present here the derivation of the formula $\sqrt{2\delta(I + M + R)} - \delta$ presented in section 4.1. This is based on Daly's [4] derivation of the formula $\sqrt{2\delta(M + R)} - \delta$ when the I/O occurred entirely synchronously with the checkpoint.

Daly approximately minimizes

$$\begin{aligned}\text{time worked} = T_w(\tau) &= \text{solve time} + \text{checkpoint time} + \text{rework time} + \text{restart time} \\ &\approx T_s + (\frac{T_s}{\tau} - 1)\delta + \left[\frac{1}{2}(\tau + \delta) + R\right]\frac{T_s}{\tau}\left(e^{\frac{\tau+\delta}{M}} - 1\right)\end{aligned}$$

where τ is the checkpoint interval, δ is the checkpoint time, M is the mean time to interrupt, R is the recovery time, and T_s the total solve time. This assumes failures are a Poisson process.

To take into account the extra rework time from an incomplete I/O phase, we need to change are computation of the rework time for each failure, which is $\frac{1}{2}(\tau + \delta)$, representing the amount of computation that must be repeated. Here, if we are interrupted t units of time after a successful checkpoint, the lost time to be made up is $\tau+\delta+t$ if $t < I$ and t is $I \leq t < \tau+\delta$. Thus the average lost time is

$$\frac{1}{\tau + \delta}\left[\int_0^I (\tau + \delta + t)dt + \int_I^{\tau+\delta} tdt\right] = I + \frac{1}{2}[\tau + \delta]$$

Modifying $T_w(\tau)$ to include this yields:

$$T_s + \left(\frac{T_s}{\tau} - 1\right)\delta + \left[\frac{1}{2}(\tau + \delta) + (I + R)\right]\frac{T_s}{\tau}\left(e^{\frac{\tau+\delta}{M}} - 1\right)$$

which is equivalent to adding the I/O phase time to the recovery time.

REFERENCES

[1] R. Brightwell, T. Hudson, K. Pedretti, R. Riesen, and K. Underwood, *Implementation and performance of Portals 3.3 on the Cray XT3*, in IEEE International Conference on Cluster Computing, Boston, Massachusetts, September 2005.

[2] R. Brightwell, R. Riesen, B. Lawry, and A. Maccabe, *Portals 3.0: protocol building blocks for low overhead communication*, in 2002 Workshop on Communication Architecture for High-Performance Clusters, Fort Lauderdale, Florida, April 2002, pp. 164–173.

[3] R. Brightwell, K. D. Underwood, and C. Vaughan, *An evaluation of the impacts of network bandwidth and dual-core processors on scalability*, in International Supercomputing Conference, Dresden, Germany, June 2007.

[4] J. Daly, *A model for predicting the optimum checkpoint interval for restart dumps*, in Proceedings of the International Conference on Computational Science 2003 (ICCS 2003), Melbourne, Australia and St. Petersburg, Russia, June 2003, pp. 724–733.

[5] J. H. L. III, L. Ward, R. Klundt, S. Kelly, J. L. Tomkins, and B. R. Kellogg, *Red Storm IO performance analysis*, in 2007 IEEE International Conference of Cluster Computing (Cluster 2007), Austin, Texas, September 2007.

[6] *IOR (interleaved or random) HPC benchmark*. `http://sourceforge.net/projects/ior-sio/`.

[7] S. M. Kelly and R. Brightwell, *Software architecture of the light weight kernel catamount*, in Proceedings of the 2005 Cray User Group Annual Technical Conference, Albuquerque, New Mexico, May 2005.

[8] J. Li, W.-k. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale, *Parallel netCDF: A High-Performance Scientific I/O Interface*, Proceedings of the 2003 ACM/IEEE conference on Supercomputing, (2003).

[9] *Lustre parallel filesystem*. `http://www.lustre.org/`.

PERFORMANCE ANALYSIS OF THE SICORTEX SC072

BRIAN J. MARTIN[†], ANDREW J. LEIKER[‡], JAMES H. LAROS III[§], AND DOUG W. DOERFLER[¶]

Abstract. The world of High Performance Computing (HPC) has seen a major shift towards commodity clusters in the last 10 years. A new company, SiCortex, has set out to break this trend. They have created what they claim to be a balanced cluster which makes use of low-power MIPS processors and a custom interconnect in an effort to avoid many of the bottlenecks plaguing most modern clusters. In this paper, we reveal the results of preliminary benchmarking of one of their systems, the SC072. First, we ran a collection of microbenchmarks to characterize the performance of interprocessor communication. Next, we ran some real applications relevant to high performance computing and compared performance and scalability to a typical commodity cluster. Lastly, we examine and compare the performance per watt of the SiCortex system to a commodity cluster.

1. Introduction. Recently, the most popular high performance computing solution has been the commodity cluster, which employs a large number of commodity processors linked together with a commercially available interconnect. This trend has largely been fed by high performance, low priced processors available for the personal computing market, as well as the advancement of a variety of open source software. SiCortex [5], a relatively new entrant in the HPC market, recently introduced a line of all-in-one clusters. They seek to avoid the inefficiencies that arise from clustering a large number of commodity parts not built for high performance computing. SiCortex claims to be the first company to engineer a cluster "from the silicon up" to create a balanced system, balancing processor speed with power consumption and communication speed in order to maximize application performance per dollar, per watt, and per square foot [11]. We benchmarked the smallest system in the lineup, the SC072("Catapult"), a 72 processor machine with the form factor of a typical desktop tower. Although the SC072 is not their largest cluster, it is representative of their unique architecture and design philosophies. In this paper we analyze the performance of the SC072 on a series of micro-benchmarks and compare the application performance and scalability of the SC072 with a typical commodity cluster. To characterize the communication performance of the interconnect, we ran a series of microbenchmarks, both from the Pallas suite and Sandia. In addition, we employed several applications to reveal performance and scalability of the SC072 and contrast between the SiCortex system and a commodity cluster. We analyzed the data gathered from these applications in several different ways, including an examination of the claims made by SiCortex regarding better performance per watt compared to a typical commodity cluster. Section 2 outlines previous work done in this area. Section 3 discusses the architecture of the SC072 and what makes it unique as well as contrasts it with a commodity cluster. A discussion of the microbenchmarking tools used takes place in section 4, and a review of the applications used to measure performance and scalability is covered in section 5. Results are presented and analyzed in sections 6 and 7, and a look at the performance per watt takes place in section 8. Our conclusions based on the data collected are presented in section 9.

2. Related Work. Preliminary analysis of the SiCortex systems has mainly been performed by the engineers at SiCortex, due in part to their recent entrance into the HPC market. Publications and technical summaries provided by SiCortex can be found in a series of white papers [6]. In addition, analysis of the zero-copy remote direct memory access (RDMA) implementation has been done by SiCortex through the use of HPCC RandomRing and Ping-

[†]Sandia National Laboratories, bjmart@sandia.gov
[‡]Sandia National Laboratories, ajleike@sandia.gov
[§]Sandia National Laboratories, jhlaros@sandia.gov
[¶]Sandia National Laboratories, dwdoerf@sandia.gov

pong benchmarks [13]. Other than the RDMA analysis done by SiCortex and a few published white papers, analysis of the SiCortex systems has been largely non-existent. Therefore, this paper strives to provide an accurate and unbiased performance analysis of the SiCortex systems.

TABLE 2.1
Test Platform Summary

System	**SiCortex SC072**	**Generic Commodity Cluster**
Processor	500 MHz MIPS64	2.2 GHz AMD Opteron Processor x86-64
Single Core Peak Floating Point Rate	1 GFLOPS	4.4 GFLOPS
Interconnect	Custom	Myricom Myri10G
Interconnect Topology	Degree-3 Kautz Graph	Clos
Compiler	PathScale version 3	Gnu Compiler Collection 3.4.3
Power Consumption per Socket	15 Watts	85.3 Watts
MPI Implementation	MPICH2	MPICH-MX

3. Architecture. The SiCortex SC072 resides in a desk-side case, plugs into a typical 100-120V electrical outlet, and draws less than 300W of power. On the inside, however, it houses twelve compute nodes, each of which is a six-way symmetric multiprocessor, containing six low-power 500 MHz 64-bit MIPS® processor cores. Each core has a peak double precision floating-point rate of 1GFLOPS, giving the entire system 72 GFLOPS of peak performance [11]. To support these processors, the system houses 48GB of memory. One of the defining features of the SiCortex line of clusters is their unique interconnect [12]. The fabric topology in the SiCortex is a unique system based on a degree-3 directed Kautz graph [10]. This Kautz topology means that the diameter of the network grows logarithmically with the number of nodes even as the degree of the network remains fixed. The fabric links can support large message bandwidth of 2GBytes/second, and since there are a total of six fabric links per node, three exit links and three entrance links, bandwidth between nodes is up to 6Gbytes/second. The SC072 runs a custom build of Linux on each of it's compute nodes, placing it on equal footing with most commodity clusters in the availability of many open source software solutions and expertise with the system. For message passing, the SiCortex includes a custom message passing interface (MPI) implementation which is based on MPICH2 and optimized for the architecture. RDMA protocol takes effect at message sizes greater than 1024 bytes, and an MPI send-receive implementation is used for message transfers below 1024 bytes. The RDMA is essentially implemented through the DMA Engine; it is one of three interconnect components. SiCortex includes several compilers available for use on their system, including the PathScale and GNU compiler suites, both containing C, C++ and FORTRAN compilers. All applications in this paper used the PathScale compiler suite, as it is optimized to the SiCortex architecture. The resource management was taken care of on the SC072 through use of the Simple Linux Utility for Resource Management (SLURM) with the default production settings.

The cluster we used in comparison to the SiCortex is a typical commodity cluster which uses 256 2.2 GHz AMD Opteron processors linked together by a Myrinet network in a Clos

topology. For all of the data gathered, this cluster was limited to 72 cores or less in order to provide a core-to-core comparison between the two systems. The job management on this cluster was handled by the Portable Batch System(PBS) in a production environment.

4. Microbenchmark Overview. In our analysis of the SiCortex system, several microbenchmarks were used to characterize SiCortex's unique communication system. Of the microbenchmarks, two were developed at Sandia National Laboratories and three were obtained from the Pallas Microbenchmark tool suite. A brief description of each is presented below.

4.1. Pallas Microbenchmarks. The Pallas microbenchmarks (version 2.2.1) are a suite of tools capable of characterizing the message passing interconnect on high performance computers. They include point-to-point, collective, and parallel transfer benchmarks. We utilized the pingpong, allreduce, and sendrecv benchmarks as they are used extensively in Sandia applications. Performance analysis for point-to-point communications was accomplished with the pingpong microbenchmark, which measures the startup and throughput as a message is sent between two processors. MPI_Send() and MPI_Recv(), blocking communication functions, form the bulk of the pingpong microbenchmark. The allreduce microbenchmark is a collective benchmark that measures the average time to reduce a vector with MPI_Allreduce(). Lastly, the sendrecv tool was chosen as the parallel transfer benchmark. A chain like communication pattern and MPI_Sendrecv() are the underpinnings of the sendrecv tools analysis capabilities. Its bidirectional bandwidth results characterize the interconnects bandwidth capabilities for communication intensive work.

4.2. Sandia Microbenchmarks. Analysis performed on the SiCortex communication system with Sandia based microbenchmarks was done through the use of the Host Processor Overhead (HPO) analysis program [1] and a modified streaming analysis program; both benchmarks were originally developed at Sandia. The HPO microbenchmark provides a picture of the total overhead and application availability on a single processor while communication is taking place. Overhead is the total processor time spent on MPI related tasks during communication. Application availability, on the other hand, is the percentage of time available for computational work during communication. As noted in citation [8], high application availability and low overhead can remedy the negative affects of a high latency, low bandwidth interconnect. MPI_Isend() and MPI_Irecv(), both non-blocking, allow communication and application work to overlap in the HPO analysis, thereby producing a realistic estimate of MPI related overhead for the send and receive calls. The final microbenchmark used for analysis was the streaming bidirectional microbenchmark. It is much like the sendrecv benchmark; however, it characterizes the interconnect somewhat differently and gives a much better estimate of bisection bandwidth. It utilizes the MPI_Sendrecv() and floods the fabric links with messages between processes for one second. Bisection bandwidth is then found based upon the total number of bytes sent in that time period.

5. Application Overview. For performance analysis and scalability, three applications were used to compare the SiCortex to a generic commodity cluster. They are described below.

5.1. HPCCG: Simple Conjugate Gradient Benchmark Code. The HPCCG micro-application is a simple partial differential equation solver and preconditioned conjugate gradient solver that solves a linear system on a beam-shaped domain. It generates a 27-point finite difference matrix for a 3D chimney domain on an arbitrary number of processors. This open source software was designed to be scalable up to thousands of processors, making it a sound software choice for analyzing scalability of a system. This software was designed to be a weak scaling code, meaning that, given the same input, the problem size doubles as the

number of processors doubles. Benchmarking data in this paper was taken with version 0.5 of HPCCG, using the reported total MFLOPS from the output of the program. HPCCG is licensed under the GNU LGPL [2].

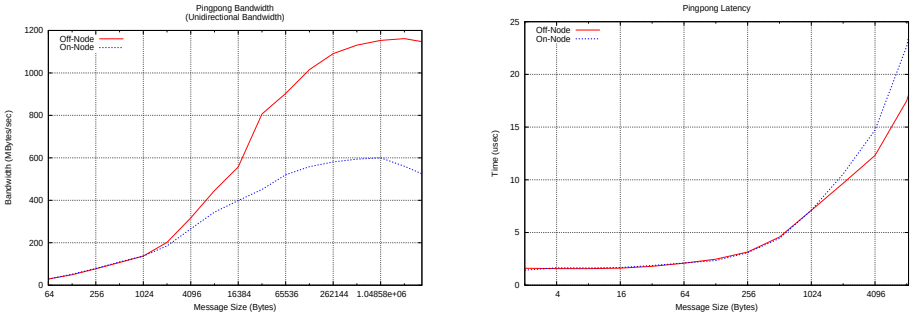
5.2. phdMesh. phdMesh is a micro-application designed to perform parallel geometric proximity search, dynamic load balancing, parallel synchronization, and other common operations on parallel, heterogeneous and dynamic unstructured meshes. The data analyzed was taken from the amount of time that the program spent performing the parallel geometric search per step [4].

5.3. LAMMPS. LAMMPS is an open source molecular dynamics simulator available under the GNU general public license. The May 21, 2008 release of LAMMPS is the version being used for our analysis [3]. Within the LAMMPS package, atomic and molecular models constitute the principal scientific tools; however, the package also has application benchmarks incorporated. The application benchmarks originate from the models themselves. As the models scale linearly, the benchmarks prove to be excellent scaling analysis tools for high performance computers. Of the five application benchmarks available for performance analysis, we chose two, the Lennard Jones liquid benchmark and the Rhodospin Protein benchmark. The two benchmarks were chosen for the dissimilarity in simulation methods and the difference in computational expense, as Rhodospin Protein is more computational and communication intensive. Both benchmarks allow for weak and strong scaling. The combination of benchmark and scaling type provide various pictures of the systems scalability, such as characteristics of the system's communication or computation scalability. Or more importantly, the overall balance of the system's scaling.

6. Microbenchmark Results. SiCortex microbenchmark results were attained through the treatment of the SiCortex cluster as a non-production environment, allowing microbenchmarking to take place without outside influence. In addition, all core allocations were handled by the SiCortex default implementation of SLURM, and all results are the average of multiple message transmissions.

6.1. Pingpong Results. Latency and bandwidth data are presented for both on and off node two processor core allocations. Both the on and off node results correlate with previous work published by SiCortex. However, previous work done by SiCortex exhibited erratic behavior at a message size of 64kB, see citation [13]; however, our results indicate SiCortex improved 64kB characteristics. For a message size of 1024 bytes, figures 6.1(a) and 6.1(b) demonstrate increased bandwidth performance and decreased latency. Performance changes at 1024 bytes are common among other communication systems, but the MPI to RDMA protocol change at this stage warrants inquiry as to its effect on the SiCortex's performance. Up to a 512 byte message size, latency values are under five microseconds, which is considerably good. Finally, on-node bandwidth performance is lower than off-node bandwidth performance; this can be attributed to the overload of on-node system memory due to the significant number of reads and writes, as noted in citation [13].

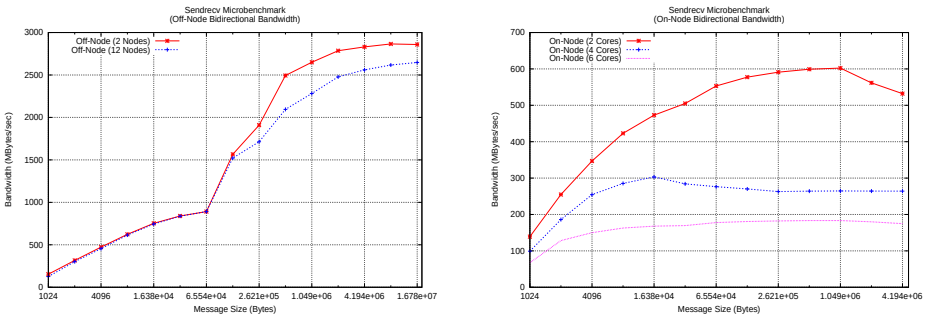
6.2. Sendrecv Results. The sendrecv benchmark was run both on and off node. Off-node jobs ranged from two to twelve nodes, where on-node jobs ranged from two to six cores. Figure 6.2(a) shows the results obtained from the two and twelve off-node jobs. Other node allocation results fall somewhere in-between these two lines. For all off-node allocation sizes, the distribution of messages across multiple fabric-links, rather than one, brought about an increase in performance at a message size greater than 64kB. Boosts in performance at this stage can be directly attributed to MPI protocol change; messages greater than 64kB are spread across fabric links in partitions of 64kB, while messages 64 kB and smaller are passed



(a) Bandwidth (b) Latency

FIG. 6.1. *Pingpong Microbenchmark Results*

on one fabric-link [13]. Interestingly, achievement of maximum bidirectional bandwidth for off-node jobs could only be obtained with message sizes greater than 4 MB. Lastly, figure 6.2(b) reveals an increase in the number of cores for an on-node job produces a predictable reduction in bidirectional bandwidth.



(a) Off-Node Performance (b) On-Node Performance

FIG. 6.2. *Sendrecv Microbenchmark Results*

6.3. Allreduce Results. Allreduce results are presented for power-of-two core allocations up to 64 cores, along with a 40 core job. Power-of-two core allocations exhibited excellent performance, while non-power-of-two core allocations did not. For example, the 40 core allocation performed 17000 microseconds slower than a 64 core allocation at a vector reduction size of 16 MBytes. The prime factor for this deviation is the inherent dependence of collective communication algorithms on power-of-two allocations. Consequently, core allocation size plays a key role in performance and a smaller job size doesn't necessarily signify optimum timings, as the 40 core allocation demonstrates. Figure 6.3 demonstrates the power-of-two dependence for collective communications on the Sicortex interconnect. Finally, increased performance with greater core allocations indicates the SiCortex's scalability for communication intensive programs.

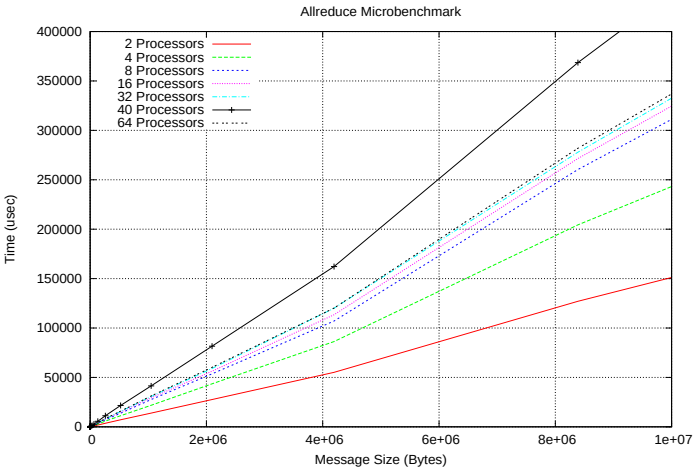
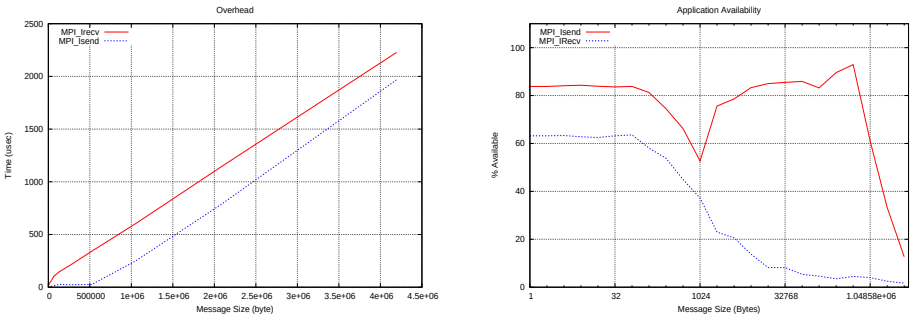


FIG. 6.3. *Allreduce Microbenchmark Results*

6.4. Host Processor Overhead. Overhead and application availability results were obtained for both an MPI Send and Receive function calls. Overhead data shown in figure 6.4(a), for both function calls, behaves linearly. In comparison to other systems, overhead performance on the Sicortex system is marginal at best [9]. Application availability for MPI_Isend in figure 6.4(b) exhibits erratic behavior, including a plunge at 1024 Bytes and a peak availability, 94.4%, at 512 KBytes. The unpredictable nature of the MPI_Isend application availability induced questions regarding the validity of the results; however, our analysis was performed multiple times and deviations were so small that results proved to be valid. Furthermore, a dependence on the RDMA protocol change is present at 1024 Bytes. In contrast to MPI_Isend, application availability for MPI_Irecv is unfavorable for all message sizes. Overall, high overhead and low application availability are common for message transfers greater than 1 MB, but performance yields below 1 MB are generally only good for the MPI_Isend function.

6.5. Bisection Bandwidth. The bisection bandwidth microbenchmark shown in figure 6.5 performed similarly to the Pallas sendrecv microbenchmark. As expected, the bisection bandwidth microbenchmark obtained a greater peak bidirectional bandwidth, a consistent 2.95 Gbytes to a sendrecv bandwidth of 2.86 GBytes. These bandwidth values place the interconnects bidirectional bandwidth capabilities under 3 GBytes, which is noteworthy. Lastly, maximum bandwidth was only obtained at message sizes greater than 4 MBytes, which also occurred in the sendrecv program; this tendency to not obtain high bandwidth until extremely large message sizes is not common among other interconnects.

7. Application Results. All applications presented in this section were tested in conditions consistent with a production environment. All applications were compiled with a high level of optimization consistent on both of the two platforms presented. All processor to MPI task core allocations were done using SLURM on the SiCortex system, and PBS on the x86 cluster.



(a) Overhead (b) Application Availability

FIG. 6.4. *Host Processor Overhead Microbenchmark Results*

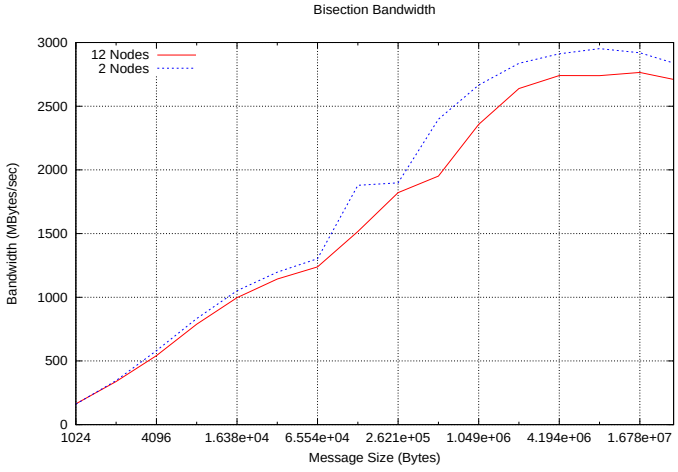
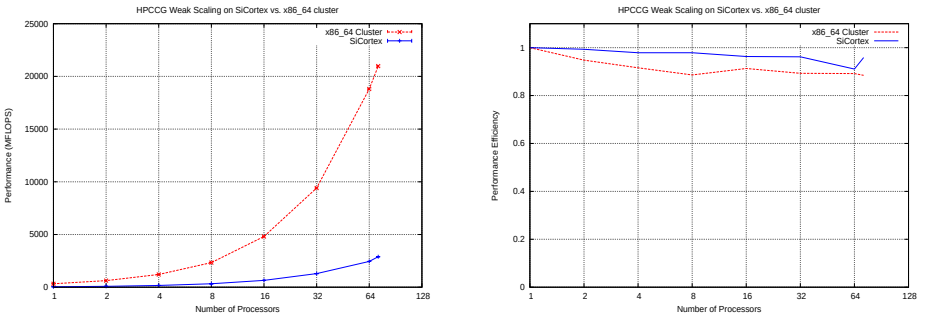


FIG. 6.5. *Bisection Bandwidth Microbenchmark Results*

7.1. HPCCG Results. A weak scaling study was accomplished using HPCCG. A problem size of 64 x 64 x 64 elements for each core was used. For each data point HPCCG was executed three times, and the results were averaged and plotted. The error bars on the plots represent the high and low point. Note the error bars are barely visible on the charts, showing consistency in the results.

Figure 7.1(a) shows how the clusters performed on HPCCG. It is clear that as far as performance goes, the Opteron cluster outperforms the SiCortex machine due to the large difference in processor speed. However, as we can see in figure 7.1(a), the SiCortex shows better scalability up to 72 cores, maintaining over 95% efficiency, whereas the Opteron cluster falls to 88% efficiency. It appears the balanced approach taken by SiCortex helps it maintain performance efficiency up to full capacity for HPCCG.

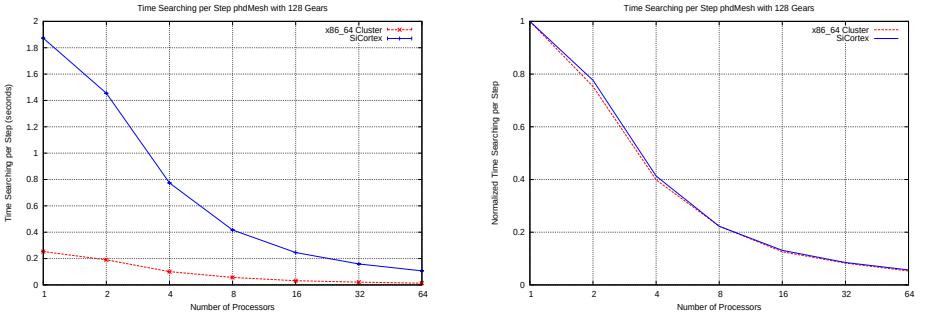


(a) Raw Data (b) Normalized Data

FIG. 7.1. *HPCCG Weak Scaling*

7.2. phdMesh Results. For phdMesh, both weak and strong scaling studies were accomplished. The data gathered for analysis comes from the amount of time performing a parallel geometric search per step. For strong scaling, a 4x8x4 mesh of 128 gears was used. Weak scaling was done with 2 gears per core arranged in an appropriate 3D mesh. For each data point, phdMesh was run three times and the average was plotted, with the high and low showing up as error bars on the plot. Again, the error bars are barely visible on the plot, showing consistent results.

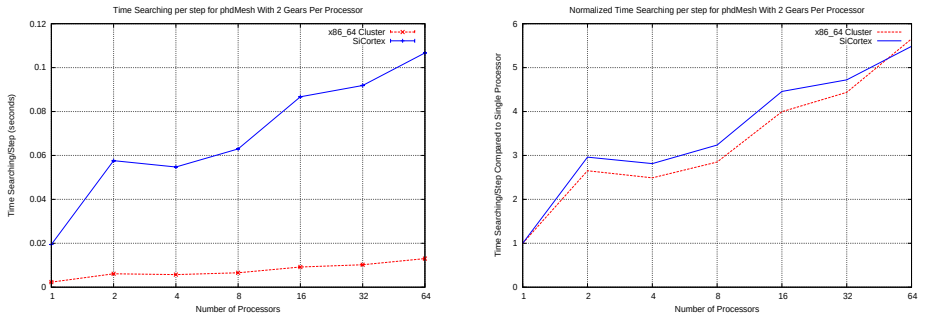
In strong scaling, as we can see in Fig. 7.2(a), the x86 cluster completed the task significantly faster due to it's greater per-core performance. Unlike HPCCG, though, the scaling was almost identical on the Opteron cluster as the SiCortex system. This can be seen in Fig. 7.2(b)(the plots are almost identical). Note that in Fig. 7.2(b), the graph is normalized so for each of the systems '1' represents the amount of time one core took searching, and all other times are relative.



(a) Raw Data (b) Normalized Data

FIG. 7.2. *phdMesh Strong Scaling*

Similar results to those seen in the strong scaling study are seen in the weak scaling study. The Opteron cluster performs better than the SiCortex in general, simply because of the difference in raw computational ability per processor. Again, the scaling of the two



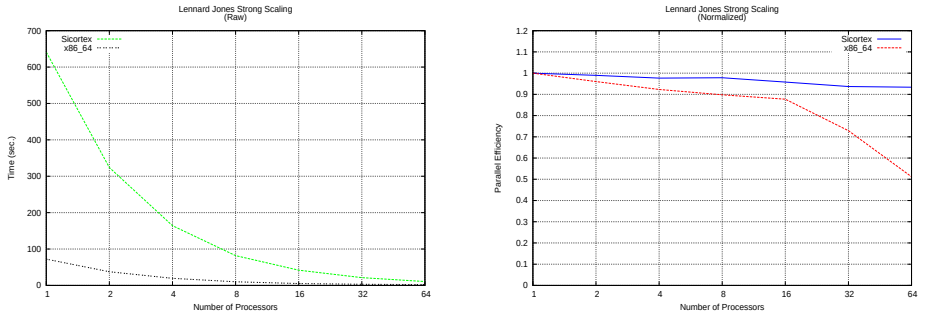
(a) Raw Data (b) Normalized Data

FIG. 7.3. *phdMesh Weak Scaling*

systems are nearly identical, as seen in Figure 7.3(b).

On phdMesh, unlike HPCCG, we see almost identical scalability between the x86 cluster and the SiCortex cluster up to 64 cores. Both systems scale fairly well and consistently up to 64 cores with phdMesh.

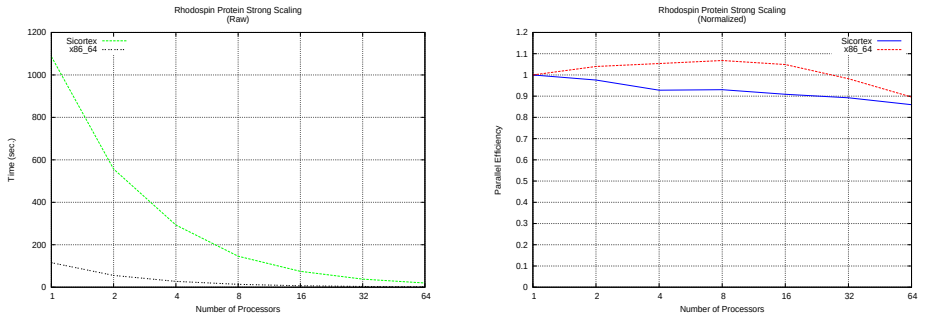
7.3. LAMMPS Results. Strong and Weak Scaling results are presented for the two LAMMPS benchmarks chosen, Lennard Jones and Rhodospin Protein. Three runs were utilized on both clusters in all LAMMPS evaluations; the final result is the normalization of the minimum time obtained for each run at each core allocation size, with normalization based on single core run times. Power-of-two core allocations up to 64 processors were selected as the job sizes.



(a) Raw Data (b) Normalized Data

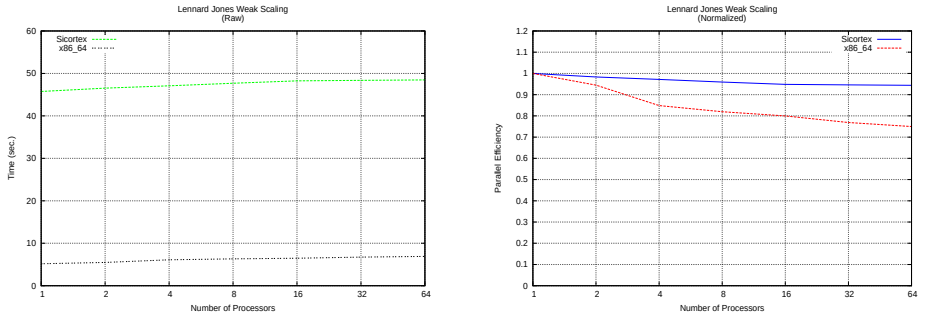
FIG. 7.4. *Lennard Jones Strong Scaling*

Problem size was set at 32000 atoms for Rhodospin Protein strong scaling. The same problem size was originally used for Lennard Jones strong scaling analysis; however, this problem size proved to be too small a problem for the Linux cluster which demonstrated poor and uncharacteristic performance. To combat the problem, a problem size of 4194304 atoms was implemented for LJ, "Lennard Jones", strong scaling on both clusters. Figure 7.4(b)



(a) Raw Data (b) Normalized Data

FIG. 7.5. *Rhodospin Protein Strong Scaling*

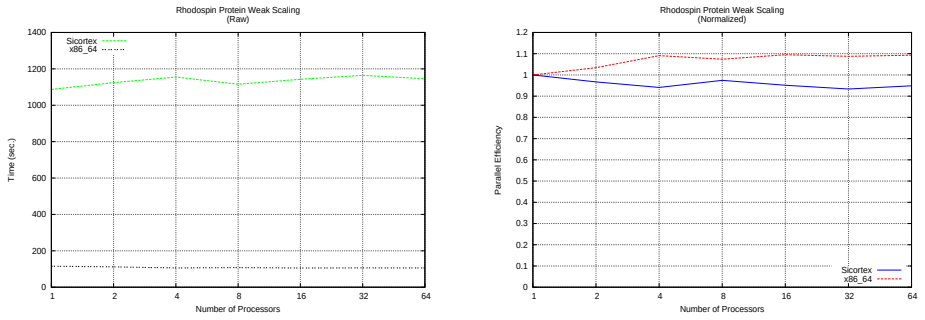


(a) Raw Data (b) Normalized Data

FIG. 7.6. *Lennard Jones Weak Scaling*

demonstrates the SiCortex's greater scalability versus those of a generic Linux cluster for LJ strong scaling, a non-computationally intensive benchmark. Correspondingly, the computationally intensive Rhodo, "Rhodospin Protein", strong scaling favored the faster generic Linux cluster for small core allocations; however, scaling performance on the generic cluster begins to fall as core allocations grow larger, which is shown in figure 7.5(b). Not to our amazement, the SiCortex scaled better than the generic Linux cluster for non-computationally intensive weak scaling (LJ microbenchmark) and the generic Linux cluster demonstrated good scaling for the computationally intensive weak scaling (Rhodo microbenchmark), see figures 7.6(b), 7.7(b). Weak scaling, in this instance, was performed by increasing the problem size proportionally to the number of processors allocated. Although the generic cluster clock times are consistently lower than the SiCortex for weak scaling, the SiCortex nearly catches the generic cluster at larger allocation sizes for strong scaling. Combined, the weak and strong scaling results demonstrate a tendency for the SiCortex to scale better for less computationally intensive programs.

8. Performance per Watt. The motivation behind the use of the underpowered MIPS64 processors found in SiCortex systems is the fact that they take a low amount of power to run



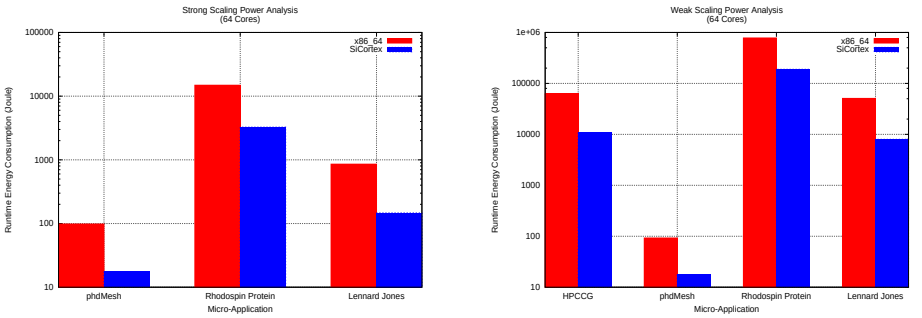
(a) Raw Data (b) Normalized Data

FIG. 7.7. *Rhodospin Protein Weak Scaling*

and provide a very high performance to watt ratio. Each processor in a SiCortex system consumes less than one watt, and each six-processor node, which includes memory and other components for those six processors, consumes less than 15 watts. In comparison, the commodity cluster's Opteron core used in this paper requires 85.3 Watts. Peak performance per watt on SiCortex systems is 322 MFLOPS per watt, an impressive number. In comparison, the average performance per watt of a system on the July 2008 version of the top500 list is 122 MFLOPS per watt [7].

As the performance per watt data for the x86 cluster was unavailable, the data presented here is based on the wattage of a single socket and the performance of the 4 applications presented in section 7. Wattage for the x86 cluster socket encompasses the NIC, memory (4 DIMMS), and the Opteron processor. The NIC and memory power contributions are estimates based upon previous work done by one of the authors; altogether, the generic cluster socket consumes approximately 115 watts. For the SiCortex, one socket power consumption is 15 watts, which is the equivalent of one node. Weak and strong scaling 64 core runs were used for our analysis. With 64 core allocations, the x86 cluster used 64 sockets and the SiCortex used 11 sockets. Considering power analysis results, it is quite apparent the energy consumption for the SiCortex system is low compared to the x86 cluster. It should also be noted that the SiCortex run time for equivalent core allocations was much higher. This would normally raise a dilemma as to whether or not the decreased power consumption is worth the increase in run time; however, the core allocation size on the SiCortex can be raised to levels that compete with lower allocation sizes on generic systems and still consume less power.

9. Conclusions. In order for SiCortex to be competitive in the HPC using the less computationally powerful MIPS processors, they need to show that the amount of cores needed to make up for computational ability in a large system will not cause a steep drop-off in performance. The results gathered largely reflect the claims that the system scales well to a reasonable number of processors. The microbenchmarks show that the communication system is capable of low-latency, high-bandwidth data transfer on par with many popular commercial interconnects. This interconnect capability coupled with the slower clock rate of the MIPS processors provide more balance than is typically seen in a commodity cluster, preventing many of the communication bottlenecks prevalent in the world of high performance computing. The application benchmarks show us that the system is consistently scalable to a reasonable number of nodes on all applications tested. As a result, more of the advertised



(a) Strong Scaling Power Analysis (b) Weak Scaling Power Analysis

FIG. 8.1. *x86_64-SiCortex Energy Consumption Comparison*

performance will be used at a high number of nodes. This was reflected in the fact that for all of the applications which analysis was performed on, performance efficiency per core never dipped below 87%. We compared these performance numbers to those of a typical commodity cluster in production, and we saw some advantages in scalability for the SiCortex system. The total performance of the commodity cluster's high-powered Opterons, however, outpaced the slower MIPS processors in the SiCortex. In some applications the commodity cluster showed the same or greater performance efficiency at a high number of processors than the SiCortex, but the SiCortex showed consistent scalability across all applications. The performance efficiency of the commodity cluster dropped as low as 51% on an application benchmarking study. In terms of performance per watt, we saw a huge advantage for the SiCortex, a big concern recently in the world of HPC due to operating costs and environmental impacts. In terms of pure processing power per node, the SiCortex MIPS64 nodes do not compete with today's modern consumer processors. However, our results demonstrate that their more balanced approach to HPC leads to consistent scalability and greater performance per watt than a typical commodity cluster.

10. Acknowledgements. We would like to express our gratitude to the Computer Science Research Institute (CSRI), the Student Internship Program (SIP), and our manager James Ang. In addition, we greatly appreciate the engineers at SiCortex for their help in preparing the SC072 for analysis.

REFERENCES

[1] *Host processor overhead (hpo)*. Available http://www.cs.sandia.gov/smb/overhead.html.
[2] *Hpccg*. Available http://software.sandia.gov/mantevo/download.html.
[3] *Lammps molecular dynamics simulator*. Available http://www.lammps.sandia.gov.
[4] *phdmesh*. Available http://www.cs.sandia.gov/ maherou/.
[5] *Sicortex website*. Available http://www.sicortex.com.
[6] *Sicortex whitepapers*. Available http://www.sicortex.com/products/white_papers.
[7] *Top 500 computer sites*. Available http://www.top500.org.
[8] D. DOERFLER, *An analsysis of the pathscale inc. infiniband host channel adapter, infinipath*, Tech. Rep. SAND2005-5199, Sandia National Laboratories, August 2005.
[9] D. DOERFLER AND R. BRIGHTWELL, *Measuring mpi send and receive overhead and application availability in high performance network interfaces*, in EuroPVM/MPI, 2006.
[10] W. KAUTZ, *"bounds on directed (d,k) graphs," theory of cellular logic networks and machines*, Tech. Rep. AFCRL-68-0668, Air Force Cambridge Research Laboratory, 1968. pp. 20-28.

[11] M. Reilly, L. Stewart, J. Leonard, and D. Gingold, *Sicortex technical summary*, April 2008. Available http://www.sicortex.com/whitepapers/sicortex-tech_summary.pdf.

[12] L. Stewart and D. Gingold, *A new generation of cluster interconnect*, April 2008. Available http://www.sicortex.com/whitepapers/sicortex-cluster_interconnect.pdf.

[13] L. Stewart, D. Gingold, J. Leonard, and P. Watkins, *Rdma in the sicortex cluster systems*, in EuroPVM/MPI, 2007.

INSTRUCTING THE MEMORY HIERARCHY WITH IN-CACHE COMPUTATIONS

PATRICK A. LA FRATTA* AND ARUN F. RODRIGUES†

Abstract. As technology advances, processing resources continue to increase in number and speed. However, memory speed continues to limit the utilization of these processing resources, especially for scientific applications that access off-chip data with relatively high frequency. On-chip cache sizes rise with new processor generations, and the intelligent use of these cache resources to mitigate slow off-chip accesses will have a direct impact on overall performance. This work enhances the ability of conventional cache hierarchies by augmenting each cache level with simple cores, called *Local Cache Processors* next to the cache lines. We define *In-Cache Computations* for utilizing these processors and present methods for forming them. Simulation results from a suite of eight scientific applications run at Sandia labs show that In-Cache Computations can offer significant performance improvements over a processor backed by a conventional cache hierarchy.

1. Introduction. The power and performance of the memory hierarchy are key factors in the design of computing systems. Efficient utilization of limited memory bandwidth is necessary for power efficiency and high utilization of compute resources. To avoid wasteful memory transfers, architectures must offer options for intelligently using available cache resources.

A large amount of research in prefetching [7, 21, 6, 1], cache partitioning [2, 15], advanced insertion policies [3, 14], miss-handling architectures [20, 5], and other areas [11, 10, 9] has addressed the problem of how to use cache resources intelligently. This work considers an unconventional system design which augments a conventional processing node by embedding simple cores, called *Local Cache Processors* (LCPs) throughout the memory hierarchy. The objective of such a design is to efficiently utilize these additional processing resources to exceed both the computational and caching capabilities of the conventional system. To do this requires the formation of small computations, which we term *in-cache computations* (ICCs), for execution on the LCPs. The process for forming ICCs must take into account not only the problem of *data placement* as in conventional cache hierarchies, but also the related problem of *computational partitioning*. This work addresses the problem by presenting two methods for ICC formation, and then evaluating their performance.

The second section offers an overview of the idea of ICCs and baseline architectural extensions. The third section presents our methodology for designing and evaluating ICC formation processes. The fourth section gives the design details of two ICC formation processes, and section 5 presents experimental results. The final section draws conclusions from these results.

2. An Overview of In-Cache Computations. Previous studies have considered the approach of augmenting a conventional processor with compute logic in either the memory controllers [4] or next to memory macros [19] using embedded DRAM technology [8]. An example of the latter is the concept of *Light-Weight Processors* (LWPs) [12], which reside in main memory, offering both parallelism and a high bandwidth memory interface to mitigate high power and performance costs of memory accesses. This work extends this approach by presenting *Local Cache Processors* (LCPs), which are simple processors embedded in the caches of conventional cores. One or more conventional cores make up the *Heavy-Weight Processor* (HWP), while the LWPs and LCPs together are both types of *Distributed Co-Processors* (DCPs). Figure 2.1 shows an example of this type of system.

*University of Notre Dame, plafratt@nd.edu
†Sandia National Laboratories, afrodri@sandia.gov

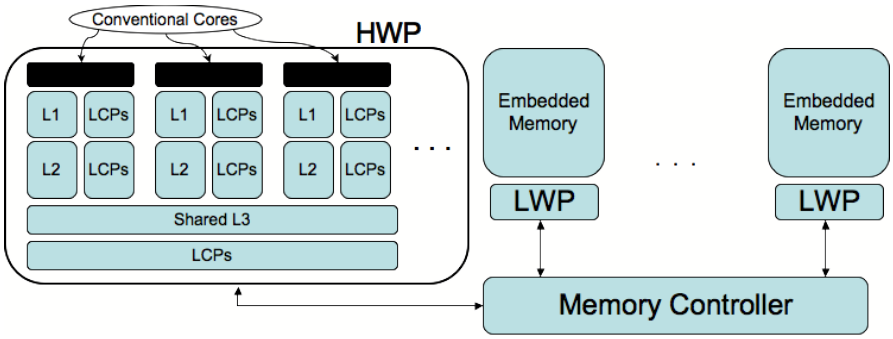


FIG. 2.1. *A Multicore Processor Backed by a DCP-Enhanced Memory Hierarchy. The HWP contains a set of cores with extensions for issuing computations to the LCPs embedded in the memory hierarchy. LWPs may also be embedded into main memory, as shown in the figure.*

An *In-Cache Computation* (ICC) that executes on an LCP is a computational unit that uses data in memory and is formed from a few instructions at the HWP. There are two primary objectives to ICC execution. The first is to increase parallelism, both system-wide and at the HWP, by offloading computations from the HWP to the DCPs. The second is to improve memory performance by reducing traffic in the memory hierarchy with better data placement.

Recent work has illustrated key characteristics of scientific applications that differ from those of benchmarks commonly used to evaluate microprocessor designs [13, 18, 16]. The design of architectures to run these scientific applications will have a significant effect on their performance, and architects should take these characteristics into account in the design process.

This work explores various architectural extensions for accelerating Sandia's scientific applications, while taking into consideration three important properties of these applications as established in part by Murphy et al. [13]:

- Large working set sizes with low temporal locality.
- High memory bandwidth requirements.
- Large basic block sizes.

The first and second properties tell us that conventional caches alone are likely insufficient for achieving acceptable memory performance. DCPs provide an alternative to bringing every piece of data into the cache for processing on a HWP. By moving computations out to the memory hierarchy, we have the potential of achieving the desirable results of cache amplification and increased parallelism. Part of the problem of utilizing DCPs is the partitioning of computations among the HWP and DCPs. The objective of this work is to propose and evaluate solutions to the problem of computational assignment. The ICC designs in section 4 will leverage our knowledge of the application properties illustrated by previous studies.

2.1. Baseline Microarchitecture. To achieve the parallelism benefits from ICC execution, the HWP must have the ability to fire an ICC efficiently. To achieve this, the HWP contains microarchitectural extensions for packaging, issuing, and committing ICCs. Much of this ICC-oriented activity taking place at the HWP will likely occur in parallel with the normal HWP actions.

An example of such microarchitectural extensions is shown in figure 2.2. In this work, we make no assumption regarding whether the ICC formation takes place at the compiler or at run-time. In the figure, the *ICC Filter* partitions instructions for execution at the HWP and DCPs. The *ICC Constructor* then packages the ICCs before passing them to the memory interface. The RUU must have extensions for handling the return of values from the ICCs.

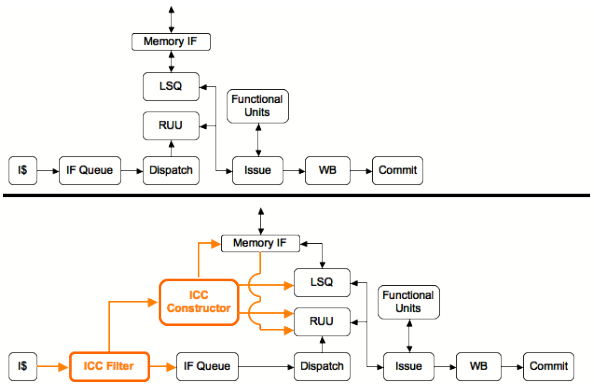


FIG. 2.2. *Microarchitectural extensions to a conventional pipeline for constructing and issuing ICCs. The top figure shows a pipeline for a conventional processor. The bottom figure shows the extensions, highlighted in orange, to this pipeline for supporting ICCs. Although the critical path will likely increase some (due to certain components such as the filter and the additional RUU logic for receiving values from the DCPs), it should be possible to perform much of the work in parallel with the conventional components.*

3. Methodology. This work uses a methodology consisting of two phases. The objective of the first phase is to present methods for ICC formation. Section 4 presents two of these methods. The second phase focuses on performance evaluation of these methods through simulation.

3.1. Applications. The simulations are runs of a suite of eight of Sandia's floating-point intensive scientific applications. The applications cover a wide range of purposes, including molecular dynamics (LAMMPS), shock mechanics of solids (CTH), radiation transport (ITS), and circuit simulation (Xyce). For evaluating ICCs, the simulations use traces of 100 million instructions of each application with varied inputs.

3.2. SST. For evaluating ICCs, we constructed a simulator based on the concept described in section 2. The simulator is an extension of the Structural Simulator Toolkit (SST) [17], which partitions simulation infrastructure between frontend and backend components. The frontend parses and feeds instructions to the backend which models the activity of the microarchitecture. For the simulations in this work, the frontend takes execution traces and feeds them to the backend which can gauge memory traffic and execution time. For baseline numbers, we simulate conventional architectures using a backend based on SimpleScalar. We then modified the frontend for translating the traces into ICC-enhanced code, and also added extensions to the SimpleScalar model in the backend for simulating the execution of ICCs. The simulations gather statistics regarding cache hit rates and ICC formation in addition to execution time to estimate performance improvements offered by ICCs.

4. ICC Design. Here, we define ICCs and explain their formation and execution in detail. Two types of ICCs are introduced: *In-Memory Operations* and *Graphlets*. The following sections define these ICC types, present methods for how a compiler or translator would form these ICC types, and describe the requirements and actions of the microarchitecture for ICC execution.

4.1. ICC Properties and Execution. DCPs execute small units of computation in the form of ICCs. ICC generation (performed by the compiler or microarchitecture or a hybrid of the two) takes groups of instructions, called the ICC's *Source Instruction Group* (SIG) from the HWP's task for offloading to the DCPs. In this work, the SIG has three requirements that

apply to ICC types:

- All external register inputs are ready at the time of issue.
- Their use does not require reordering of instructions in the code.
- The SIG must include at least one memory access.

The first requirement makes the execution of ICCs simpler. An ICC may use data at different physical locations in the system, but this data will always reside in memory and not within registers. Next, there is always a *clean* insertion point in the code for the ICC creation instruction, called the `iccci`. This means that the dependences between the ICC and other instructions can be indicated without major code modifications, such instruction reordering and register renaming. This greatly simplifies the translation process. Finally, as a computation that is fundamentally memory-centric, an ICC must write or read at least one data value to or from memory. Additional requirements will be imposed by specific ICC type definitions.

To execute an ICC, the HWP first issues the `iccci` to a functional unit, called the ICC constructor, for building the ICC. The ICC constructor bundles the SIG and any required register values into a package marked with the memory addresses containing data the ICC will use. The ICC then travels up the memory hierarchy, similar to a load or a store, for execution on a DCP. The destination DCP unpackages the ICC, and executes the computations. When the computation requires data at a remote location, the DCP may fetch the value or repackage the remaining computation into a new ICC for migration to a new location. ICC type definitions specify the details of how ICC synchronization, communication, and migration occur.

4.2. In-Memory Operations. An *In-Memory Operation* (IMO) is an ICC whose SIG contains exactly one store and no external register outputs. The address to which the IMO's store writes a value is called the *Target Memory Address* (TMA). A load may be included in the SIG only if it reads from the TMA. Hence, an IMO requires no communication due to data dependences, and produces a single value for storage.

The goal of IMO SIG extraction is to maximize the number of instructions in the IMO without violating the IMO SIG criteria. The instructions are grouped by basic block for the extraction process. A dependence graph, G, is then constructed from the instructions in this group. Algorithm 2 shows steps for forming the SIG for an IMO.

Algorithm 2: IMO SIG Extraction

Input: G, s
`/* G: dependence graph of a basic block.`
`s: A store in G.` `*/`
1 Remove loads to addresses other than the TMA.
2 Remove conditional control flow instructions.
3 Create a topological sort, T, of the remaining vertices.
4 Mark the node containing s. $i = |T|$.
5 **while** $i > 0$ **do**
6 | **if** *$T(i)$ has at least one child and all of $T(i)$'s children are marked* **then**
7 | | Mark $T(i)$.
8 | **end**
9 | $i = i$-1.
10 **end**
11 Remove all unmarked vertices, call the remaining subgraph H.
12 Replace all nodes in G corresponding to the nodes in H with a single `iccci` node.

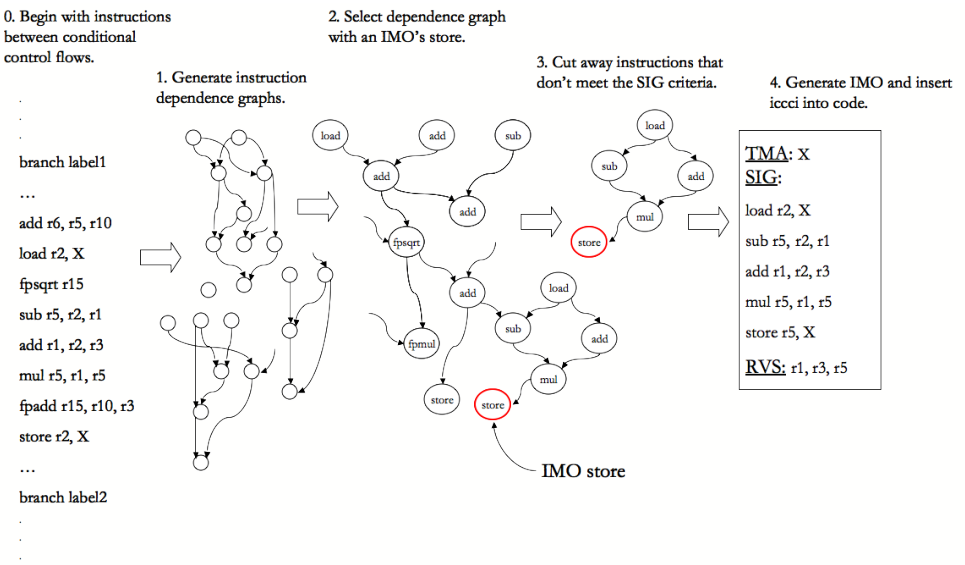


FIG. 4.1. *IMO Translation Process*

A compiler or translator contains an implementation of this algorithm which forms the IMOs. In this work, we implement the algorithm in a trace translator for the simulations. The translation process that implements this algorithm is illustrated in figure 4.1. The translator first forms the dependence graph of instructions in a basic block. Nodes in the graph are cut away using algorithm 2, and the remaining nodes are compressed into a single node that represents the `iccci` node in the HWP's executable. This new node contains three pieces of information: the TMA, the instructions taken from the SIG, and the set of register indices (marked *RVS* in figure 4.1) of external register inputs for the SIG.

The execution of an IMO, pictured in figure 4.2, is a multi-step process that the HWP initiates with the `iccci` instruction. When the HWP reaches an `iccci` in the code, it issues the instruction to the ICC constructor which packages the IMO and ships it out to the memory hierarchy. The IMO travels up the hierarchy similar to a load or a store. When the TMA hits a cache, the DCP that is nearest the physical memory holding the TMA unpackages the IMO for execution.

One primary strength of IMOs is that they require little communication among the processors. Because IMOs access exactly one address, this guarantees that its computation can execute on one DCP with no dependences on remote data. Additionally, there are no register dependences between the IMO's SIG and the instructions executing at the HWP. Hence, the IMO may return only an acknowledgement message to the HWP to indicate its completion.

4.3. Graphlets. The concept of graphlets leverages the third application property listed in section 2. This property of large basic block sizes indicates to us that frequently when a branch is taken, many instructions will follow with known control flow. In fact, if the processor has reached the branch target before, it can leverage the fact that it has seen the instructions that follow.

Further inspection of the basic blocks in Sandia's applications reveals that they frequently contain one large weakly connected component (WCC) accompanied by one or more small WCCs. Additionally, the WCCs often interact with memory. Therefore, we define a *graphlet* as an ICC whose SIG is a small WCC with at least one memory access in the data dependence

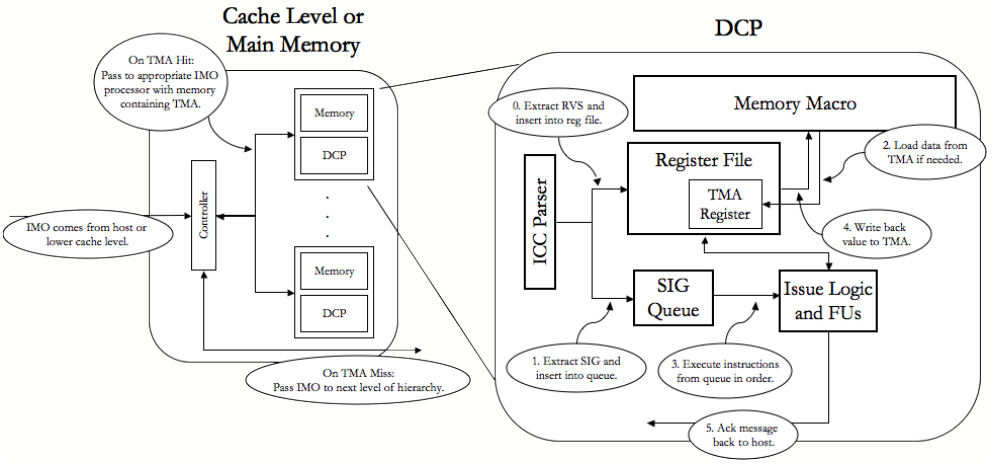


FIG. 4.2. *Execution of an IMO*

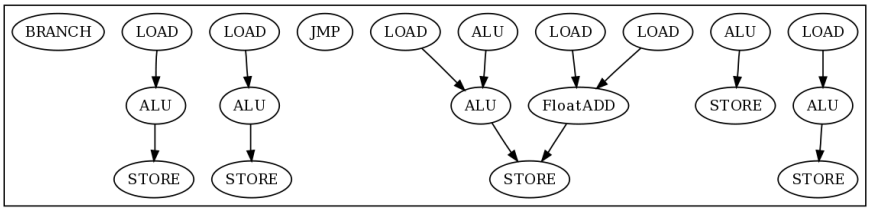


FIG. 4.3. *Example of a Basic Block Taken from CTH. In this example, the five WCCs containing memory accesses are the graphlets.*

graph of a basic block. Since the SIG is the entire WCC, it has no external register dependences with instructions in the basic block outside the SIG. An example of a basic block containing graphlets is shown in figure 4.3, taken from Sandia's CTH application. Algorithm 3 gives the steps used for graphlet SIG extraction.

Algorithm 3: Graphlet SIG Extraction

Input: G, SR
/* G: dependence graph of a basic block.
SR: an integer used in the construction of graphlets. */
1 Split the graph into a set, S, of WCCs.
2 Let B be the set of graphs with maximal number of nodes in S.
3 Let $|V(b)|=M$, for some $b \in B$.
4 **foreach** $s \in (S-B) \mid s$ *contains at least one load or store* **do**
5 | **if** $|V(s)| \leq (M/SR)$ **then**
6 | | Replace nodes of s in G with a single `iccci` node.
7 | **end**
8 **end**

The execution of graphlets is more complex than that of IMOs. First, the SIG of a graphlet may contain more than one memory access. These accesses may have different

addresses, implying that the graphlet may require data residing at more than one physical location in the memory hierarchy. Our design assumes that the graphlet is executed on a DCP at the lowest level of the memory hierarchy holding any data needed by the graphlet. Any additional memory values required are retrieved via a fetch message to remote DCPs. Another complication is that the SIG may contain indirect loads, which require a mechanism for virtual address translation for the DCPs.

Another design point of graphlets is that they may return register values to the HWP upon completion. This requires some extensions to the RUU or LSQ at the HWP that allow reception and handling of sets of graphlet-produced register values from the memory interface.

The primary advantage of the graphlets approach is that the partitioning of instructions leads to a natural load balancing between the HWP and the DCPs. While the graphlets – which are relatively high latency events from the HWP's perspective – are executing, the HWP stays busy executing the largest WCC of instructions in the basic block. The number of nodes in this WCC is divided by the *size ratio* value (SR in algorithm 3) to produce the maximum number of nodes a WCC may contain for it to qualify as the SIG for a graphlet. The size ratio can be varied to gauge the significance of load-balancing effects. SR is set to two for the experiments in this work.

5. Evaluation. This section presents a discussion of experimental results with ICCs. The simulations for baseline performance numbers use the SimpleScalar processor model with configuration shown in table 5.1. We then modified the processor and memory models to support execution of IMOs and graphlets. In the first set of experiments, we simulate the eighteen application traces using the SimpleScalar-based processor model in SST unmodified. The next set simulates the traces with IMO extensions. For every store in the trace for which it is possible to extract a SIG of size>0 using algorithm 2, the translator transforms these computations into an IMO. The speedup results, showing the ratio of execution time with IMOs to execution time without IMOs, are given in figure 5.1. For nine of the traces, IMOs offered noticeable improvement. For eight of the traces, there was an insignificant change in performance, and for only one case did IMOs have a negative effect on performance. In the next set of experiments, we ran the traces through a translator in SST's frontend implementing algorithm 3, and simulate graphlet architectural extensions in the backend. Figure 5.2 gives the speedup results for the graphlet simulations. These results show that graphlets offered improvement for eight of the traces, and were detrimental to performance in three cases. In the remaining seven cases, there was negligible change in performance.

5.1. Analysis. Performance improvements from ICCs can generally be broken into two components. The first is speedup of instructions executed at the HWP. This includes, for example, relaxed pressure on execution resources, such as the RUU and LSQ, at the HWP due to compiler-directed offloading of instructions to the DCPs. This also includes improved cache hit rates of accesses executed at the HWP. The other component is the increased parallelism achieved from the utilization of DCPs. The improvement from this component will be greater if data placement in the caches leads to reduced memory traffic due to ICC communication.

We incorporated mechanisms into the frontend and backend for gathering statistics regarding simulator activities that are relevant to these components. We now discuss the trends in these statistics to offer explanations of the positive, negative, and neutral performance results in figures 5.1 and 5.2.

Tables 5.2 and 5.3 include five values for each of the eighteen application traces. The first column, IR (instruction ratio), is the relative amount of instructions in the original trace that were offloaded from the HWP to the DCPs. This is a rough estimate of the potential parallelism that the DCPs may exploit.

TABLE 5.1
Baseline HWP Configuration

Register Update Unit Size	64 entries
Load-Store Queue Size	32 entries
Fetch Queue Size	4 entries
Decode Width	4
Issue Width	8
Functional Units	Integer ALU - 3 Integer Mult/Div - 1 FP Add - 2 FP Mult/Div - 1
Branch Predictor	Bimodal, 2048 entries
Branch Target Buffer	512 entries, 4-way
Branch Mispredict Penalty	3 cycles
Caches	DL1 - 2048 sets, 32-byte blocks, direct-mapped, 4-cycle latency DL2 - 32768 sets, 32-byte blocks, 4-way, 27-cycle latency
TLBs	ITLB - 16 entries, 4-way DTLB - 32 entries, 4-way 30-cycle miss penalty
Main Memory	Latency - 150 cycles Bandwidth - 64 bits/cycle

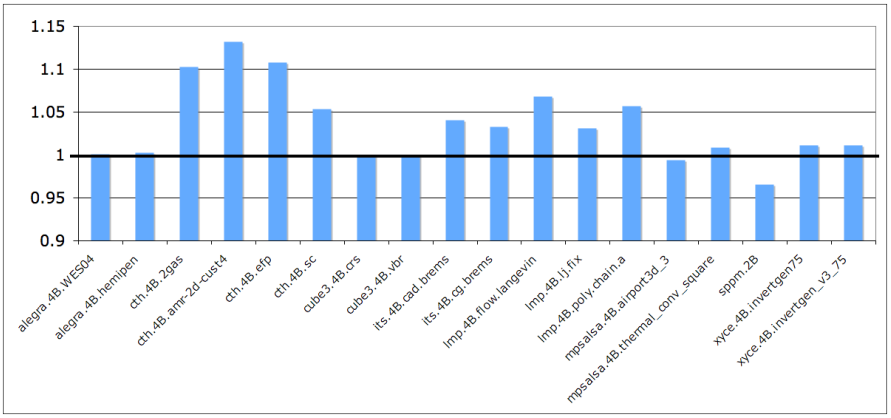


FIG. 5.1. *IMOs Speedup*

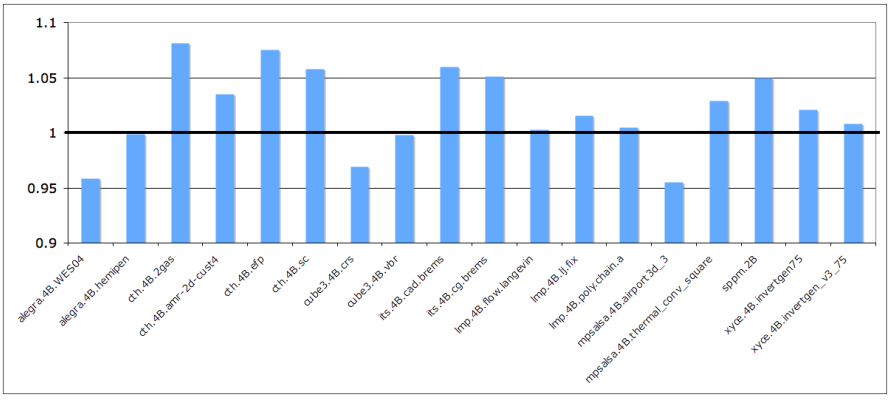


FIG. 5.2. *Graphlets Speedup*

The second column, *AR* (access ratio), is the percentage of the memory accesses in the original trace that were inserted into ICCs. This value offers an estimate of the performance impact of the *HWP's cache hit rates* versus the *ICC cache hit rates*. The HWP's cache hit rates are the hit rates of the caches from memory accesses *executed at the HWP* – that is, accesses that are *not* inserted into ICCs. The *ICC cache hit rates* are the hit rates of the caches from memory accesses inserted into the ICCs. Let A be the set of addresses that an ICC touches. For address $a \in A$, let L_a be the level of the lowest cache holding a. Let h be an address in A such that $L_h \geq L_a \ \forall \ a \in A$. The ICC will then impose a miss in all cache levels lower than L_h, and a hit at level L_h.

Hence, an *AR* of 0 indicates that the HWP is executing all memory accesses from the original program. The ICC hit rates will have no impact on performance, and the HWP hit rates will have significant impact on performance. An *AR* of 0.99 would indicate that almost all memory accesses have been inserted into ICCs, implying that the hit rates of accesses from the HWP will have little impact on overall performance.

The third column, *L2_HR* (L2 hit ratio), expresses the change (from baseline to with ICCs) in L2 hit rate of memory accesses executed at the HWP. *L2_HR* is calculated by dividing the HWP hit rate at L2 of the simulations with ICCs by the L2 hit rate of the baseline simulations. This ratio gives an indication of how the use of ICCs affects the L2 hit rate of the HWP's memory accesses. Ideally, the use of ICCs would increase HWP hit rate, and this ratio would be > 1. *L1_HR* (the L1 hit ratios) are not included in the table because they are 1 for almost every case and insignificant. The next two columns show the ICC hit rates for both the L1 and L2 caches. The last column is the performance improvement data shown in figures 5.1 and 5.2, given here to improve readability of the results. (Note that all data in tables 5.2 and 5.3 have slight rounding error.)

Considering the results in table 5.2, the first point of importance is that an *IR* of ≈ 0 indicates that IMOs will likely have negligible effect on performance. This means that algorithm 2, the IMO SIG extraction algorithm, was unable to create a significant number of SIGs given the computations in the trace. Hence, this immediately discounts five traces – *alegra.4B.WES*04, *alegra.4B.hemipen*, *cube*3.4*B.crs*, *cube*3.4*B.vbr*, *mpsalsa.4B.airport3d_3* – from benefiting from IMOs. The performance results agree with this hypothesis: all of these applications saw negligible performance effects from IMOs.

We will come back to the outlier, *sppm.2B*. For the remaining twelve traces, IMOs resulted in positive performance. For half of these traces, the improvement was >5%. The general trend is that performance follows *IR*. The six traces resulting in the greatest improvement had *IR*>8%, while the other six had *IR*<7%. None of these had a significantly detrimental effect on HWP cache hit rates, and the trace with the most improvement, *cth.4B.amr-2d-cust4*, resulted in an improvement in HWP L2 cache hit rate.

Now consider the outlier, *sppm.2B*. This trace has unusually high values of *IR* (53%) and *AR* (75%), yet showed negative performance results. To explain this, we consider *AR* along with the hit rates of the HWP and ICCs. Because of this very high *AR* value, the ICC hit rate will have a significant impact on performance. This offers an explanation of the negative results of this application, since the hit rates of both caches were ≈ 0. The reason for this is that IMOs, as defined here, do not move data to or from the caches. They use the data wherever it resides and return only an acknowledgement back to the host. Most of the memory accesses are inserted into IMOs, and the results show that most of the working set of this application remains in main memory.

Table 5.3 gives the statistics from the graphlets simulations. As is the case with IMOs, $IR \approx 0$ should imply that graphlets will have negligible effect on performance. Only one application in table 5.3, *cube*3.4*B.vbr*, follows this hypothesis. *mpsalsa.4B.airport3d_3*, with

TABLE 5.2
IMOs Simulation Statistics

Application	IR	AR	L2_HR	IHR_L1	IHR_L2	Impr.
alegra.4B.WES04	0.01	0.01	1.00	0.58	0.12	0.00
alegra.4B.hemipen	0.01	0.01	1.00	0.89	0.40	0.00
cth.4B.2gas	0.15	0.15	0.99	0.77	0.89	0.10
cth.4B.amr-2d-cust4	0.12	0.17	1.06	0.74	0.12	0.13
cth.4B.efp	0.14	0.15	0.97	0.86	0.74	0.11
cth.4B.sc	0.09	0.12	0.95	0.80	0.83	0.05
cube3.4B.crs	0.00	0.00	1.00	–	–	0.00
cube3.4B.vbr	0.00	0.00	1.00	1.00	–	0.00
its.4B.cad.brems	0.06	0.10	1.00	0.96	0.05	0.04
its.4B.cg.brems	0.06	0.10	1.00	0.94	0.04	0.03
lmp.4B.flow.langevin	0.09	0.06	1.00	0.43	0.07	0.07
lmp.4B.lj.fix	0.06	0.06	0.99	0.53	0.70	0.03
lmp.4B.poly.chain.a	0.08	0.07	1.00	0.70	0.79	0.06
mpsalsa.4B.airport3d_3	0.00	0.01	0.92	0.03	0.00	-0.01
mpsalsa.4B.thermal_conv_square	0.03	0.05	0.99	0.78	0.13	0.01
sppm.2B	0.53	0.75	1.77	0.00	0.00	-0.03
xyce.4B.invertgen75	0.04	0.04	1.00	0.67	0.99	0.01
xyce.4B.invertgen_v3_75	0.04	0.04	1.00	0.61	0.94	0.01

IR of only 1% and a 4% slowdown is an exception to this rule. Note that although only 2% (996 889) of all memory accesses in the original trace are inserted into graphlets, this is a significant portion (23%) of L2 accesses. Even a relatively small number of additional misses in higher cache levels can have a significant effect on performance. Hence, negative effects on L2 cache hit rate are particularly detrimental to performance when L1 cache hit rate is low. Note that one other application among graphlets exhibited both a low HWP L1 hit rate and low L2_HR (*cube*3.4*B.crs*), and this application also saw a drop in performance (3%). *mpsalsa*.4*B.airport*3*d*_3 also saw these hit rate changes for IMOs, but without the significant drop in performance. The likely reason is that IMOs are more tolerant of high latency accesses occurring within IMOs, since the accesses within IMOs are guaranteed, by definition, to have no output register dependences.

The remaining traces have high values of *IR* and *AR*, indicating high potential performance improvement from graphlets. However, three of these applications (*alegra*.4*B.hemipen*, *lmp*.4*B.flow.langevin*, and *lmp*.4*B.poly.chain.a*) showed no improvement, and another (*alegra*.4*B.WES*04) saw a 4% drop. *L2_HR*≈1 for all of these traces, and trends in the remaining statistics of ICC hit rate offer an explanation for the remaining results. Notice that the four applications mentioned above all have a mediocre ICC L1 hit rate accompanied by a low or mediocre ICC L2 hit rate. The traces with a high ICC L1 hit rate (*its*.4*B.cad.brems*, *its*.4*B.cg.brems*, *sppm*.2*B*) perform well. The traces with mediocre ICC L1 hit rate along with a high L2 hit rate (*xyce*.4*B.invertgen_v*3_75, *xyce*.4*B.invertgen*75, *cth.4B.amr-2d-cust4*) offered little improvement. The remaining five traces don't fall into any of the prior categories, exhibiting mediocre ICC L1 and L2 hit rates. For these traces, the performance generally is in proportion to *IR*.

6. Conclusions. Using available cache space to improve performance, especially for applications with high memory bandwidth requirements, requires intelligent data placement decisions. Adding computational logic next to the caches increases the difficulty of the problem of resource utilization, but reveals opportunities for cache amplification, decreased memory traffic, and increased parallelism. In this work, this available logic is put to use with In-Cache Computations (ICCs), revealing some of the difficulties in computation assignment and data placement. The results show that offloading computation to in-cache processors renders data placement decisions more important, as performance benefits of ICCs are highly dependent on cache hit rates. Cache conflicts between the HWP and ICCs are a key difficulty in the

TABLE 5.3
Graphlets Simulation Statistics

Application	IR	AR	L2_HR	IHR_L1	IHR_L2	Impr.
alegra.4B.WES04	0.08	0.12	1.00	0.59	0.34	-0.04
alegra.4B.hemipen	0.08	0.11	0.99	0.59	0.53	0.00
cth.4B.2gas	0.09	0.15	1.00	0.85	0.82	0.08
cth.4B.amr-2d-cust4	0.12	0.23	0.98	0.63	0.85	0.03
cth.4B.efp	0.10	0.16	1.00	0.93	0.37	0.08
cth.4B.sc	0.08	0.14	1.00	0.85	0.51	0.06
cube3.4B.crs	0.00	0.01	0.91	0.00	0.00	-0.03
cube3.4B.vbr	0.00	0.00	0.98	0.00	0.01	0.00
its.4B.cad.brems	0.06	0.17	0.99	0.97	0.75	0.06
its.4B.cg.brems	0.06	0.17	0.99	0.98	0.25	0.05
lmp.4B.flow.langevin	0.07	0.15	1.00	0.85	0.21	0.00
lmp.4B.lj.fix	0.05	0.09	1.00	0.81	0.82	0.02
lmp.4B.poly.chain.a	0.08	0.14	1.00	0.77	0.49	0.00
mpsalsa.4B.airport3d_3	0.01	0.02	0.88	0.00	0.00	-0.04
mpsalsa.4B.thermal_conv_square	0.03	0.10	0.99	0.93	0.41	0.03
sppm.2B	0.04	0.13	1.00	1.00	0.92	0.05
xyce.4B.invertgen75	0.08	0.14	1.00	0.58	0.91	0.02
xyce.4B.invertgen_v3_75	0.08	0.14	1.00	0.54	0.87	0.01

problem of data placement.

Our simulation results show that a system supporting ICCs can offer significant speedups over processors backed by a conventional cache hierarchy. There are several topics of interest for future work for improving ICC performance, particularly in the area of optimizing HWP and ICC cache utilization. This work will consider how to leverage past studies of advanced cache management in scratchpads, bypassing, insertion policies, and variable line sizes to improve data placement decisions. Another task of merit is the enhancement of multicore hierarchies with DCPs. ICCs may significantly reduce memory traffic and increase parallelism, especially in the presence of high cache coherence traffic, synchronization overhead, and cache pollution.

REFERENCES

[1] S. BYNA, Y. CHEN, AND X.-H. SUN, *A taxonomy of data prefetching mechanisms*, Parallel Architectures, Algorithms, and Networks, 2008. I-SPAN 2008. International Symposium on, (2008), pp. 19–24.

[2] J. CHANG AND G. S. SOHI, *Cooperative cache partitioning for chip multiprocessors*, in ICS '07: Proceedings of the 21st annual international conference on Supercomputing, New York, NY, USA, 2007, ACM, pp. 242–252.

[3] Y. ETSION AND D. FEITELSON, *Probabilistic prediction of temporal locality*, Computer Architecture Letters, 6 (2007), pp. 17–20.

[4] Z. FANG, L. ZHANG, J. B. CARTER, A. IBRAHIM, AND M. A. PARKER, *Active memory operations*, in ICS '07: Proceedings of the 21st annual international conference on Supercomputing, New York, NY, USA, 2007, ACM, pp. 232–241.

[5] A. GANDHI, H. AKKARY, R. RAJWAR, S. T. SRINIVASAN, AND K. LAI, *Scalable load and store processing in latency tolerant processors*, in ISCA '05: Proceedings of the 32nd annual international symposium on Computer Architecture, Washington, DC, USA, 2005, IEEE Computer Society, pp. 446–457.

[6] C. J. HUGHES AND S. V. ADVE, *Memory-side prefetching for linked data structures for processor-in-memory systems*, J. Parallel Distrib. Comput., 65 (2005), pp. 448–463.

[7] I. HUR AND C. LIN, *Memory prefetching using adaptive stream detection*, in MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture, Washington, DC, USA, 2006, IEEE Computer Society, pp. 397–408.

[8] S. IYER, J. J.E. BARTH, P. PARRIES, L. NORUM, J. RICE, L. LOGAN, AND D. HOYNIAK, *Embedded dram: Technology platform for the blue gene/l chip*, IBM Journal of Research and Development, 49 (2005), pp. 333–350.

[9] T. L. JOHNSON, D. A. CONNORS, M. C. MERTEN, AND W. MEI W. HWU, *Run-time cache bypassing*, IEEE Transactions on Computers, 48 (1999), pp. 1338–1354.

[10] N. JOUPPI, *Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers*, Computer Architecture, 1990. Proceedings., 17th Annual International Symposium on,

(1990), pp. 364–373.

[11] M. Kharbutli and Y. Solihin, *Counter-based cache replacement and bypassing algorithms*, Computers, IEEE Transactions on, 57 (2008), pp. 433–447.

[12] S. Li, A. Kashyap, S. Kuntz, J. Brockman, P. Kogge, P. Springer, and G. Block, *A heterogeneous lightweight multithreaded architecture*, Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International, (2007), pp. 1–8.

[13] R. Murphy, A. Rodrigues, P. Kogge, and K. Underwood, *The implications of working set analysis on supercomputing memory hierarchy design*, in ICS '05: Proceedings of the 19th annual international conference on Supercomputing, New York, NY, USA, 2005, ACM Press, pp. 332–340.

[14] M. Qureshi, A. Jaleel, Y. Patt, S. Steely, and J. Emer, *Set-dueling-controlled adaptive insertion for high-performance caching*, Micro, IEEE, 28 (2008), pp. 91–98.

[15] M. K. Qureshi and Y. N. Patt, *Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches*, Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on, (2006), pp. 423–432.

[16] A. Rodrigues, R. Murphy, P. Kogge, and K. Underwood, *Characterizing a new class of threads in scientific applications for high end supercomputers*, in ICS '04: Proceedings of the 18th annual international conference on Supercomputing, New York, NY, USA, 2004, ACM Press, pp. 164–174.

[17] A. F. Rodrigues, *Programming future architectures: dusty decks, memory walls, and the speed of light*, PhD thesis, Notre Dame, IN, USA, 2006. Adviser-Peter Kogge.

[18] K. Rupnow, A. Rodrigues, K. Underwood, and K. Compton, *Scientific applications vs. spec-fp: a comparison of program behavior*, in ICS '06: Proceedings of the 20th annual international conference on Supercomputing, New York, NY, USA, 2006, ACM Press, pp. 66–74.

[19] S. Thoziyoor, J. Brockman, and D. Rinzler, *Pim lite: a multithreaded processor-in-memory prototype*, in GLSVSLI '05: Proceedings of the 15th ACM Great Lakes symposium on VLSI, New York, NY, USA, 2005, ACM, pp. 64–69.

[20] J. Tuck, L. Ceze, and J. Torrellas, *Scalable cache miss handling for high memory-level parallelism*, Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on, (2006), pp. 409–422.

[21] S. P. VanderWiel and D. J. Lilja, *Data prefetch mechanisms*, ACM Computing Surveys, 32 (2000), pp. 174–199.

ARBITRARY DIMENSION REED-SOLOMON CODING AND DECODING FOR EXTENDED RAID ON GPUS

MATTHEW L. CURRY*, H. LEE WARD†, ANTHONY SKJELLUM‡, AND RON BRIGHTWELL§

Abstract. Reed-Solomon coding is a method of generating arbitrary amounts of checksum information from original data via matrix-vector multiplication in finite fields. Previous work has shown that CPUs are not well-matched to this type of computation, but recent graphical processing units (GPUs) have been shown through a case study to perform this encoding quickly for the 3 + 3 (three data + three parity) case. In order to be utilized in a true RAID-like system, it is important to understand how well this computation can scale in both the number of data disks and parity disks supported. This paper details the performance of a general Reed-Solomon encoding and decoding library that is suitable for use in RAID-like systems. Both generation and recovery are benchmarked and discussed.

1. Introduction. Our previous work [2] has given a thorough treatment of the reliability of disk drives composed into arrays. Some metrics of the individual drives, such as the mean time to failure (MTTF), are more optimistic than real-world measured results [6]. Furthermore, preventative reporting mechanisms like SMART (Self-Monitoring, Analysis, and Reporting Technology) are not a reliable means of proactively preventing data loss by identifying likely drive failures [3]. Even bit error rates are becoming more important as individual drives become larger. By showing failure rates for several configurations and situations, we concluded that more reliable storage configurations are required to avoid data corruption. While nested RAID configurations (*e.g.*, RAID 1+0 and RAID 5+0) can alleviate reliability concerns for very large arrays, this is an inefficient use of hardware resources. Nested RAID drastically increases the amount of storage hardware required, increasing expense per unit of storage.

We proposed extending the RAID philosophy by implementing storage groups with arbitrary numbers of parity disks, then described the challenges of implementing a system. Generation of data on the parity disks requires some type of error correcting code. An example of a common space-efficient coding scheme for generating arbitrary parity is Reed-Solomon coding [5]. In an example system of $n + m$ disks, a code can be created such that any n disks can be used to regenerate the contents of any other m disks in the set.

While space-efficient, Reed-Solomon coding is computationally intensive. For an $n + m$ system, generating m bytes of parity for n bytes of data requires nm multiplication operations in a finite field [4], where n is customarily larger than m. Multiplying numbers with direct methods in a finite field requires several elementary bitwise operations. A lookup table is a common optimization, but most x86/x86-64 CPUs do not implement an operator to allow parallel table lookups [1]. SSE5 may allow such vectorization in the future, but this instruction set has yet to be implemented. These conditions cause x86/x86-64 CPUs to be slow in performing large numbers of finite field multiplications.

In order to deal with the high cost of computing the parity for $n + m$ systems where $m > 2$, we proposed using an outboard compute device which is better suited to Reed-Solomon coding. In particular, we detailed the advantages of using a GPU, while addressing integration into a RAID system. We benchmarked a 3 + 3 implementation of the Reed-Solomon coding component, showing up to a tenfold improvement over a CPU running a well-known Reed-Solomon coding library.

*University of Alabama at Birmingham, curryml@cis.uab.edu
†Sandia National Laboratories, lee@sandia.gov
‡University of Alabama at Birmingham, tony@cis.uab.edu
§Sandia National Laboratories, rbbrigh@sandia.gov

In this paper, we show the implementation and performance of a generalized GPU coding library. It is capable of generating arbitrary amounts of parity, and recovering up to m erasures.

2. Motivation: GPU RAID. The intended application of this library will be a RAID system which utilizes a GPU to perform checksumming operations. This is a challenging application because of the nature of GPU computation, which is vastly different from that of traditional RAID controllers and CPUs used for software RAID. Being that RAID controllers are inline computation devices, high bandwidth and low latency are easily achievable while performing the parity computations. Similarly, CPUs operate directly on main memory, allowing partial results to be written to disk while the CPU is still working on other parts of the same task. GPUs have a different method of computing: In order to be efficient, large contiguous buffers must be transferred to the GPU, operated upon, then transferred back into main memory for writing to disk. This introduces an unavoidable amount of latency.

In order to combat latency, a system architecture (pictured in Figure 2.1) has been proposed which provides full bandwidth available from the disks used for the RAID system, and little apparent latency for writes from the user perspective. The main requirement of the system is the gathering of requests into buffers. Here, the driver which accepts the write requests copies the relevant data into the GPU Accumulate Buffer, which gathers requests while the GPU is busy doing other work. Once the GPU becomes available, the buffers are rotated such that new requests go into an empty buffer and the GPU Accumulate Buffer is made to be the GPU Operate Buffer. The GPU performs DMA transfers from this buffer, operates on its contents, and returns the parity portions to the GPU Operate buffer. When this is completed (and outstanding disk write requests are completed), the buffers are rotated again. The GPU Operate Buffer moves to the Disk Writeout Buffer, where its contents are flushed to disk.

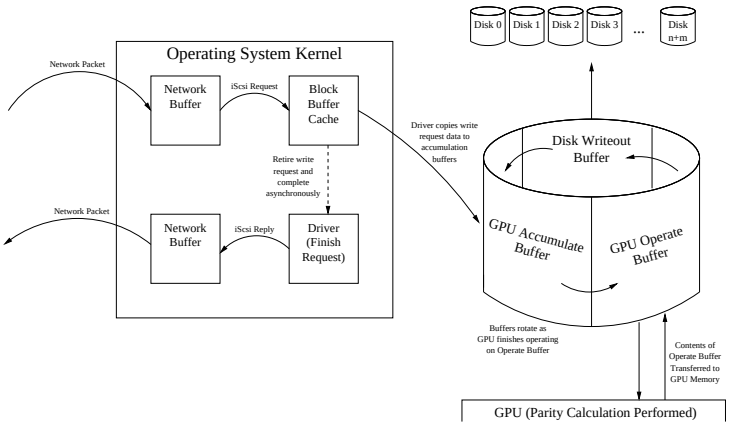


FIG. 2.1. *A GPU RAID Architecture for Write Access*

2.1. ***Caveats.*** This is a simplified diagram which assumes full stripe updates rather than partial stripe updates, with the implementation of the experiment mirroring this assumption. Future revisions of this work will include update calculations for partial stripe writes. Also, an often-discussed downside to this architecture is that there is a need for three times the buffer space compared to other more traditional RAID implementations.

3. Implementation. Due to a novel memory architecture, as well as good general availability, this implementation is for CUDA-enabled GPUs produced by NVIDIA. CUDA is a GPGPU (general-purpose computation on GPUs) technology which allows programming of

GPUs using a C-like language and syntax divorced from the graphics idiom. While other GPGPU toolkits exist, CUDA is the most general purpose and widely available. For more information on the choice of CUDA, please see the initial paper [2].

RAID implementations are expected to exhibit several dynamic behaviors, including automatic regeneration of lost content and addition of extra data disks. A point to note is that the size of the tasks involved are long running and infrequent. RAID array regeneration takes place over several hours, as does the reshaping of an array. Furthermore, it is desired that the array can operate for long periods without either of these events occurring. These assumptions allow for compile-time fixing of n and m, which confers many benefits. One of the biggest benefits is the speed at which the portions of the program can run on the GPU. As with many architectures, loop unrolling is a well-known technique for increasing the speed of a given program on a GPU. However, this becomes difficult to exploit effectively on a GPU when the loop size is determined at runtime. As exceptional situations occur, new CUDA binaries can be generated to meet the new needs. CUDA provides facilities to allow loading and unloading of routines during runtime. This philosophy was applied to the parity generation portion of this work. The previous paper also used compile-time fixing of m and n, but used hard-coded literals instead of constants. This implementation has been tested for widely varying values of m and n.

This implementation further differentiates itself from the previous implementation in that several other parameters are easily tunable at compile time for current and upcoming graphics card architectures. While performing this work, many surprises were discovered in the implementation of the compiler for CUDA which were exposed specifically during various performance tunings. For instance, CUDA supports a `sizeof` operation for determining the size of a type. In many C and C++ compilers, this operation is compiled to an integer constant when a definite type is used as an argument. However, the CUDA compiler does not compile the `sizeof` operator to a constant, but instead evaluates it at runtime. Being that fetches from global memory can be more efficiently performed using types that are larger than a single byte, the GPU programs perform fetches with different data types than those used for calculation. In order to easily change the type used for fetching data (for compile-time performance tuning), a `typedef` and the `sizeof` operator are used for defining the fetch type and looping over its contents, respectively. A significant performance improvement was found when using a constant value instead of the `sizeof` operator in loops.

4. Results. These experiments were carried out on a workstation which contains a Core 2 Quad Q6600, 4 gigabytes of memory, and an NVIDIA GeForce 8800 Ultra.

Three performance graphs (Figures 4.1, 4.3, and 4.4) have been generated. In general, the horizontal axis denotes how much data is given to a routine for $n + m$ encoding. For both cases, n blocks of data are provided as input, each of $size/n$ bytes. For example, for a data point at 1000 kilobytes in a 5 + 3 system, an input of 1000 kilobytes is provided to the routine, which splits it into 5 blocks of 200 kilobytes apiece. This allows direct comparison of the methods with the same amount of input data. The vertical axis shows the throughput of the routines, which includes all data transfer between the host memory and the GPU, for the amount of data provided. So, while a $5+3$ computation of 1000 kilobytes in total requires the transfer of 1600 kilobytes, the vertical axis denotes the time required to compute the parity and divides it by the cumulative size of the n data blocks. While this causes a less dense sampling of performance as n increases, this provides the most fair representation.

Figure 4.1 shows the performance of three $n + 3$ configurations corresponding to arrays with 8, 16, and 32 disks. This result is significantly more optimistic than the last paper's results, even while the benchmark is being run on the same platform. Figure 4.3 shows the relative improvement of the new implementation over the old implementation. One trend to

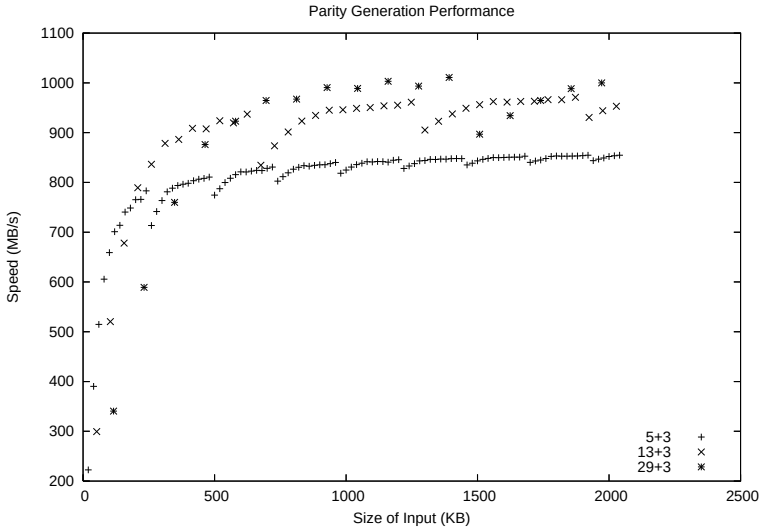


FIG. 4.1. *Speed of GPU Parity Generation*

note is that as n increases relative to m, overall performance increases. This is attributable to the amount of parity that is transferred from the GPU to the CPU. As n increases, the number of computations per byte of data remains constant. Given n, m, and a data size of s:

$$\begin{aligned} complexity(n+m) &= matvec(n \times m)(numberOfStripes) \\ &= (2nm)(s/n) \\ &= 2ms \end{aligned}$$

A true RAID system would include the notion of a block size in the above equation. However, the block size is not a concern of this layer of software. The block size has been factored out of the equation, as it does not affect the result.

As computations remain constant, data transferred back to the host does not remain constant. As mentioned previously, with a fixed data size of s, the size of each data block *and* parity block is s/n. As n grows, the collective size of all parity (ms/n) decreases, causing a decrease in transfer time. Figure 4.2 supports this hypothesis by showing that, although the absolute amount of computation remains the same, the percentage of time spent transferring data across the PCI bus decreases as n increases.

A further detail indicates that the buffer size required is not a problem, even on a system which has other processes and tasks. For all tested configurations, only approximately one megabyte of data is required to keep the system exhibiting peak performance for the largest configuration. As previously mentioned, the system needs at least three buffers of the size processed on the GPU in order to function. This amounts to only three megabytes, an incredibly tiny amount compared to today's typical memory size.

A more interesting trend is found in Figure 4.4. While the throughput of the decoding routine is still higher than the throughput of the generation routine in our original paper, it is significantly slower than the generation in this paper. There are many reasons for this

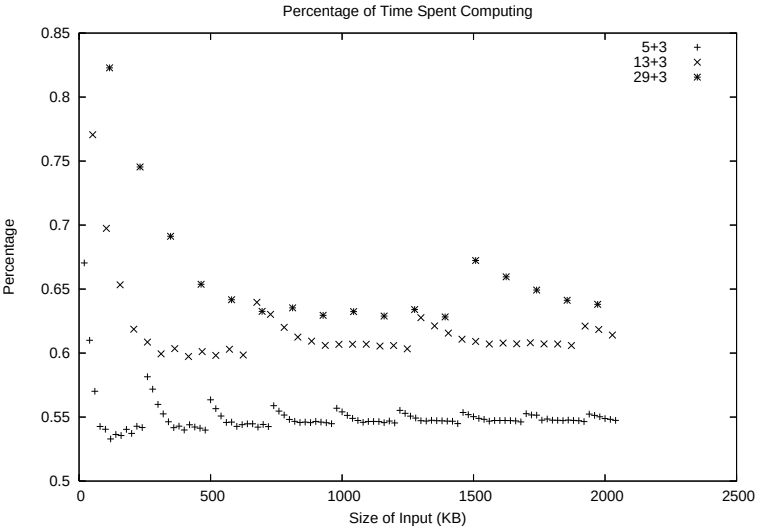


FIG. 4.2. *Time Spent Computing (Not Including PCI Traffic) as a Percentage of Total Time*

disparity:

- While generation requires multiplication of a vector of n bytes by a $m \times n$ matrix, recovery requires multiplying a vector of n bytes by a $n \times n$ matrix. This is significant, as m is often much smaller than n.
- Generation creates m bytes of output for every n bytes of data. However, recovery requires reconstructing a full set of n data bytes from a mixture of data bytes and parity bytes. The current implementation replaces buffers that are marked "failed" with parity buffers, then updates these parity buffers in place with the data buffers' previous value. The result requires either noncontiguous transfers from GPU memory or transfer of all n data buffers from the GPU. This procedure can obviously be optimized, but as it is currently implemented the transfer of the recalculated data is significant.
- One single regeneration algorithm is not sufficient to exploit the GPU's best performance. In order to obtain the best performance, one must balance how slow global memory of the GPU is versus how much computation is required to avoid such accesses. In this case, m out of n buffers are being updated with a matrix multiplication. It is simple to optimize such a computation such that no more than mn multiplications are required instead of the full n^2 described by the original decoding algorithm. Such a computation requires knowledge of which buffers have failed, which would typically be passed via a pointer to a data structure describing which buffers need regeneration. For the smaller cases ($n = 5$ and $n = 13$), the expense of loading and indexing this array slows the overall computation because of the additional memory references. The larger case obviously requires more careful optimization.

As it is, the regeneration time for large arrays is not desirable. Such an operation should be as fast as possible in order to support read requests of a degraded array, as well as to

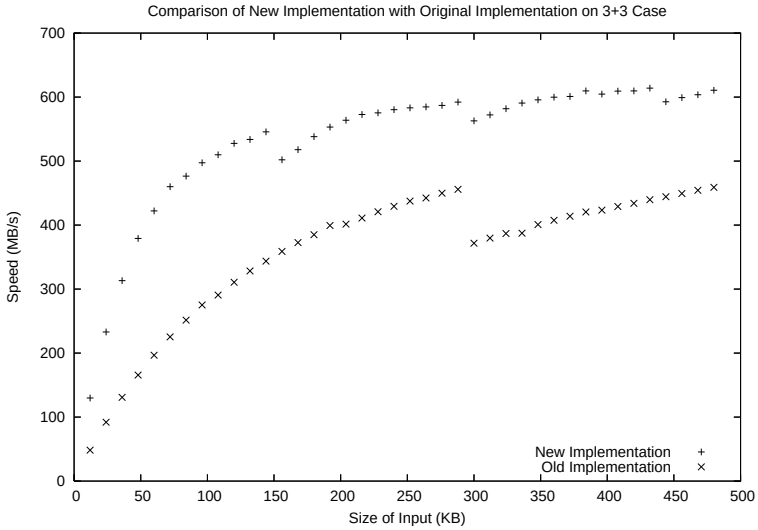


FIG. 4.3. *Performance Comparison of New and Old Implementations*

speed rebuilding time. Luckily, we believe that one lesson applied to parity generation can be applied to data recovery. Data recovery in a RAID system is required when a disk in the array fails, requiring its entire contents to be regenerated. This operation consists of decoding gigabytes of data, causing long periods of disk input from remaining disks and output to the replacement disk. This routine can gain *a priori* information on which disks are failed by being compiled specifically to recover the failed buffers and handle them appropriately in memory to allow contiguous transfers. Such an optimization will require only a few seconds for initialization, but can yield significant performance increases over the required hours of computation.

5. Conclusion and Future Work. This work presented an update to a previous work describing a Reed-Solomon coding library suitable for extended RAID arrays. It utilizes GPUs enabled with NVIDIA's CUDA technology, and allows for coding and decoding faster than modern CPUs. This work aimed to generalize, broaden, and optimize the previously developed library.

The results for arbitrary dimension Reed-Solomon coding are encouraging: The computation is very fast, with improvements already available via hardware upgrades. With the advent of asynchronous overlapping transfers to CUDA-enabled GPUs, it may become possible to hide the cost of the PCI-Express transfer altogether.

Decoding for small cases looks promising, much like the original paper showed promise for coding. GPUs certainly outperform CPUs at the same task for $n \leq 13$. However, as the size of an array scales, it becomes obvious that decoding is going to prove to be a more complex challenge. It is not a simple matter to get the most performance possible out of a GPU due to its restrictions on data layout, access, and transfer. In order to address these challenges, this paper has laid out several approaches, including further compile-time optimizations beyond simple loop unrolling. This includes compilation of code for specific data layouts and

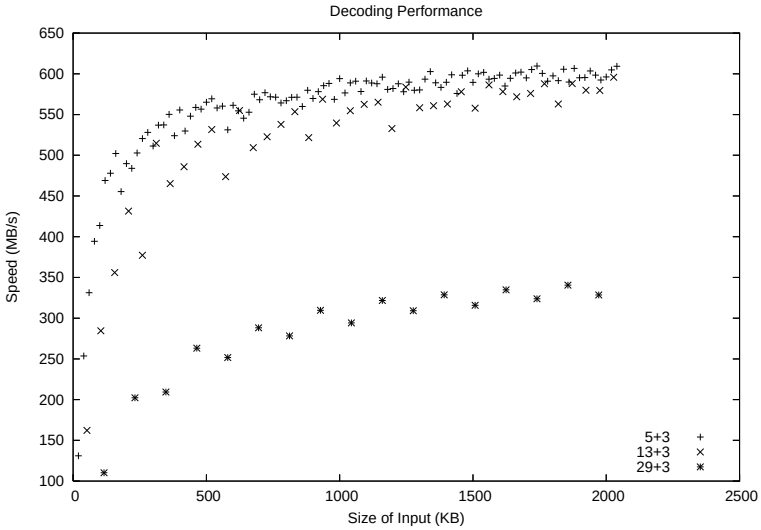


FIG. 4.4. *Speed of GPU Decoding (Three Lost Buffers)*

recovery workloads on the fly.

REFERENCES

[1] H. P. ANVIN, *The mathematics of RAID-6*. `http://kernel.org/pub/linux/kernel/people/hpa/raid6.pdf`, 2007. Accessed on April 8, 2008.

[2] M. L. CURRY, A. SKJELLUM, H. L. WARD, AND R. BRIGHTWELL, *Accelerating Reed-Solomon coding in RAID systems with GPUs*, in Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on, 2008.

[3] E. PINHEIRO, W.-D. WEBER, AND L. A. BARROSO, *Failure trends in a large disk drive population*, in Proceedings of the 5th USENIX Conference on File and Storage Technologies, Berkeley, CA, USA, 2007, USENIX Association, pp. 17–28.

[4] J. S. PLANK, *A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems*, Software – Practice & Experience, 27 (1997), pp. 995–1012.

[5] I. S. REED AND G. SOLOMON, *Polynomial codes over certain finite fields*, Journal of the Society for Industrial and Applied Mathematics, 8 (1960), pp. 300–304.

[6] B. SCHROEDER AND G. A. GIBSON, *Disk failures in the real world: what does an MTTF of 1,000,000 hours mean to you?*, in Proceedings of the 5th USENIX Conference on File and Storage Technologies, Berkeley, CA, USA, 2007, USENIX Association, pp. 1–1.

Applications

Necessity is the mother of invention and, ultimately, applications drive the advances in computational science, mathematics, and algorithms. The papers in this section span several disciplines, and utilize advanced mathematical and computational tools to address important problems and applications in their respective fields.

Seleson et al. introduce peridynamics, a formulation of continuum mechanics that is based on integral equations rather than PDEs, as an upscaling of molecular dynamics. In particular, they show that dispersion effects and the higher order gradient PDE obtained from molecular dynamics and peridynamics are consistent to leading order. Numerical results confirm that in contrast to continuum mechanics, dispersion effects are well–captured by peridynamics simulations. *Kim and Hennigan* provide an overview of the mathematical models included in Charon, a numerical simulator code used to model electrical semiconductor devices, and discuss changes required for the addition of quantum effects. In particular, they present a numerical scheme based on the coupling of Schrödinger and Poisson equations. Numerical results indicate that taking into account quantum effects, the peak electron density values in an n-channel MOSFET device are located a few nanometers away from the interface, differing from the classical density distribution that predicts a peak at the interface. *Jayaraman et al.* describe an atomistic simulation procedure used for the calculation of melting points of complex molecules. The procedure, implemented in LAMMPS, is based on recently published work involving a novel thermodynamic integration technique. The results obtained from LAMMPS simulations are compared to a reference code (APSS) and indicate that the LAMMPS implementation was carried out successfully. *Harden and Lehoucq* present a novel idea that enables convergence verification, typically applied in the context of mesh–based PDE simulations, in the inherently mesh–free setting of molecular dynamics. They develop equivalent molecular dynamics/continuum mechanics formulations for a typical test problem and, using the molecular dynamics package LAMMPS, demonstrate that it is possible to recover convergence rates associated with the PDE model. *Ames et al.* demonstrate that long–term sustainability of large scientific application codes can be achieved by building a team ethic that relies on the principle of Lean Software Development in combination with a testing environment that offers rigorous verification and validation services. Proof of concept is given on the example of ALEGRA, a complex multiphysics code that couples magnetics, hydrodynamics, thermal conduction, and radiation transport.

D. Ridzal
S.S. Collis

December 11, 2008

PERIDYNAMICS AS AN UPSCALING OF MOLECULAR DYNAMICS

PABLO SELESON[*,‡], MICHAEL L. PARKS[†,‡], AND MAX GUNZBURGER[*]

Abstract. The nonlocal continuum mechanics theory *peridynamics*, presented in [7], is based on an integral formulation, in contrast to the local classical theory of elasticity. We focus on the nonlocality of the peridynamic model and show how peridynamics preserves dispersion effects inherent to nonlocal molecular dynamics models. A one dimensional analytical comparison of dispersion relations and higher order gradient equations of motion for molecular dynamics and peridynamics is presented, as well as computational results.

1. Introduction. Substantial computational challenges are involved in materials science modeling, because of the complexity of the systems of interest. Two main descriptions are traditionally presented in the literature of materials science modeling: the continuum mechanics (CM) model, which assumes a continuity of matter, and the molecular dynamics (MD) model, which employs a discrete description of materials. For large systems, MD models are computationally too expensive, while classical CM models do not accurately capture the microscopic properties caused by spatial inhomogeneities. Our purpose is to develop peridynamics (PD) as an upscaling of MD so that it preserves characteristic properties of MD models lost by classical CM models, while being less expensive than MD.

Starting with MD models, we derive PD models, and show that the equations of motion and the dispersion relations agree between both models when preserving the nonlocality of the interaction. Following that, we expect to recover in PD similar dynamics as in MD at reduced cost. We support this analysis with numerical experiments.

In section 2 we summarize the peridynamics theory and describe the specific model implemented in this investigation. In section 3 we present analytical results for the dispersion relations obtained in MD and PD. In section 4 we present the higher order gradient PDE obtained for PD and MD, showing agreement between the equations of motion of both models up to leading order. In section 5 we present a stability analysis for the velocity Verlet algorithm implemented on our experiments. In section 6 we present the results of our numerical simulations, showing how dispersion effects introduced by the nonlocality of the MD system are preserved in PD. In section 7 we present conclusions.

2. The Peridynamics Model. The nonlocal continuum theory *peridynamics* is based on an integral formulation, in contrast to the differential formulation of classical continuum mechanics. For more on the peridynamic theory, see [7]. For more on the discretization of the PD model, see [8, 4].

The general peridynamic equation of motion [7] is

$$\rho\ddot{\mathbf{u}}(\mathbf{x},t) = \int_{\mathcal{H}_\mathbf{x}} \mathbf{f}\left(\mathbf{u}(\mathbf{x}',t) - \mathbf{u}(\mathbf{x},t), \mathbf{x}' - \mathbf{x}\right) \mathrm{d}V_{\mathbf{x}'} + \mathbf{b}(\mathbf{x},t), \tag{2.1}$$

with $\mathcal{H}_\mathbf{x}$ a neighborhood of $\mathbf{x}$ (i.e. a spherical region of radius δ around $\mathbf{x}$, where δ is called the *horizon*), $\mathbf{u}$ the displacement vector field, $\mathbf{b}$ the body force, ρ the mass density, and $\mathbf{f}$ the pairwise force function. We define the relative position $\boldsymbol{\xi} = \mathbf{x}' - \mathbf{x}$, and the relative displacement $\boldsymbol{\eta} = \mathbf{u}(\mathbf{x}',t) - \mathbf{u}(\mathbf{x},t)$. Let $\mathbf{f}$ be derived from a scalar micropotential w s.t.

$$\mathbf{f}(\boldsymbol{\eta},\boldsymbol{\xi}) = \frac{\partial w}{\partial \boldsymbol{\eta}} \qquad \forall \boldsymbol{\eta},\boldsymbol{\xi}.$$

[*]Department of Scientific Computing, Florida State University, {seleson,gunzburg}@scs.fsu.edu. Research supported by DOE/OASCR under grant number DE-FG02-05ER25698.

[†]Sandia National Laboratories, mlparks@sandia.gov. Research supported by DOE/OASCR.

[‡]Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

In general, we express the pairwise force function as

$$\mathbf{f}(\boldsymbol{\eta}, \boldsymbol{\xi}) = f(|\boldsymbol{\xi} + \boldsymbol{\eta}|, \boldsymbol{\xi}) \frac{\boldsymbol{\xi} + \boldsymbol{\eta}}{|\boldsymbol{\xi} + \boldsymbol{\eta}|},$$

with f a scalar-valued function. The macroelastic energy density is

$$W = \frac{1}{2} \int_{\mathcal{H}_{\mathbf{x}}} w(\boldsymbol{\eta}, \boldsymbol{\xi}) dV_{\boldsymbol{\xi}}.$$

In the *prototype microelastic brittle* (PMB) model [8], we choose

$$f = cs,$$

with c a constant depending on the bulk modulus of the material and the horizon, where for a 3-D material the relation

$$c = \frac{18K}{\pi \delta^4}$$

holds, with K the bulk modulus. Furthermore, s is the stretch defined by

$$s = \frac{|\boldsymbol{\xi} + \boldsymbol{\eta}| - |\boldsymbol{\xi}|}{|\boldsymbol{\xi}|},$$

and the corresponding pairwise potential function of the PMB model is

$$w = \frac{1}{2} c s^2 |\boldsymbol{\xi}|.$$

3. Relating Peridynamics and Molecular Dynamics through Dispersion Relations. In this section, we show that the dispersion relation (3.3) obtained for a 1-D nonlocal linear chain of atoms matches the corresponding relationship (3.5) obtained for the peridynamic PMB model in the continuum limit, when preserving the nonlocality of the system. In contrast, (3.3) is not consistent with a local continuum model. For the purposes of this paper, we will use a nonlocal mass-spring network as our representative molecular dynamics model.

3.1. Molecular Dynamics Dispersion Relation. Our one dimensional MD model consists of a nonlocal linear chain of atoms. Each particle is assumed to have a mass m, and is connected to N neighbors on each side. The relaxed distance for nearest neighbors is assumed to be a, and the spring constant between a particle and its j-th neighbor is $K/(ja)$, with K a constant. The equation of motion is

$$m\ddot{u}(x,t) = \sum_{j=-N}^{N} \frac{K}{ja} \Big[u(x + ja, t) - u(x,t) \Big], \tag{3.1}$$

with $u(x,t)$ the displacement field. Setting $u(x,t) = e^{i(kx+\omega t)}$, we get the dispersion relation

$$\omega^2 = \sum_{j=1}^{N} 2 \frac{K}{m} \frac{1}{ja} \Big[1 - \cos(jka) \Big].$$

Assuming $Nka \ll 1$ (i.e. we assume the wavelength λ is much longer than the maximum interaction distance Na) we can apply a Taylor expansion for every j and get the approximate dispersion relation

$$\omega^2 \approx 2 \frac{K}{m} \Big[\Big(\sum_{j=1}^{N} j \Big) \frac{ak^2}{2!} - \Big(\sum_{j=1}^{N} j^3 \Big) \frac{a^3 k^4}{4!} + \Big(\sum_{j=1}^{N} j^5 \Big) \frac{a^5 k^6}{6!} - \Big(\sum_{j=1}^{N} j^7 \Big) \frac{a^7 k^8}{8!} \Big].$$

Introducing the notation $\zeta^N(n) = \sum_{j=1}^{N} j^n$, and using the Taylor expansion (up to order 4) of the function $\sqrt{1+x}$ and the relation between mass and density $m = \rho a$, we obtain the expression

$$\omega \approx \sqrt{\frac{K}{a^2\rho}}\left[\sqrt{\zeta^N(1)}ak - \frac{\zeta^N(3)}{\sqrt{\zeta^N(1)}}\frac{a^3k^3}{24} + \left(\frac{\zeta^N(5)}{\sqrt{\zeta^N(1)}}\frac{1}{720} - \frac{1}{1152}\frac{\left(\zeta^N(3)\right)^2}{\left(\zeta^N(1)\right)^{3/2}}\right)a^5k^5\right].$$

Using closed form expressions for the summations we can write

$$\omega \approx \sqrt{\frac{K}{a^2\rho}}\left[\sqrt{\frac{1}{2}N^2 + \frac{1}{2}N}ak - \frac{\frac{1}{4}N^4 + \frac{1}{2}N^3 + \frac{1}{4}N^2}{\sqrt{\frac{1}{2}N^2 + \frac{1}{2}N}}\frac{a^3k^3}{24}\right. \tag{3.2}$$

$$\left. + \left(\frac{\frac{1}{6}N^6 + \frac{1}{2}N^5 + \frac{5}{12}N^4 - \frac{1}{12}N^2}{\sqrt{\frac{1}{2}N^2 + \frac{1}{2}N}}\frac{1}{720} - \frac{1}{1152}\frac{\left(\frac{1}{4}N^4 + \frac{1}{2}N^3 + \frac{1}{4}N^2\right)^2}{\left(\frac{1}{2}N^2 + \frac{1}{2}N\right)^{3/2}}\right)a^5k^5\right].$$

For $N \gg 1$ (i.e. the number of neighbor interactions is very large) we can keep only the dominant terms and get

$$\omega \approx \sqrt{\frac{K}{2a^2\rho}}\left[(Na)k - \frac{1}{48}(Na)^3k^3 + \left(\frac{1}{2160} - \frac{1}{4608}\right)(Na)^5k^5\right], \tag{3.3}$$

which agrees with the dispersion relation (3.5) obtained for PD in the next section. The assumptions $N \gg 1$ and $Nka \ll 1$ are both satisfied in the continuum limit ($a \ll 1$) case, when we preserve the nonlocality of the system, such that Na remains constant. If instead we use (3.2) for a nearest-neighbor interaction ($N = 1$), in the continuum limit we get

$$\omega \approx ck,$$

the classical wave equation (6.1) dispersion relation with $c = \sqrt{\frac{K}{\rho}}$ the wave speed.

3.2. Peridynamics Dispersion Relation. We now derive the corresponding dispersion relation for the peridynamic model. The peridynamic equation of motion for our one dimensional model is

$$\rho\ddot{u}(x,t) = \int_{-\delta}^{\delta}\frac{c}{|\epsilon|}(u(x+\epsilon,t) - u(x,t))d\epsilon. \tag{3.4}$$

We assume $u(x,t) = e^{i(kx+\omega t)}$, and get the dispersion relation [9]

$$\omega^2 = \int_0^{\delta} 2\frac{c}{\rho|\epsilon|}(1 - \cos(k\epsilon))d\epsilon.$$

Assuming $k\delta \ll 1$ (i.e. we assume long wavelengths in the sense of $\lambda \gg \delta$, similar to the previous section) and using a Taylor expansion we obtain, after integration, the relation

$$\omega^2 \approx 2\frac{c}{\rho}\left(\frac{k^2\delta^2}{2!2} - \frac{k^4\delta^4}{4!4} + \frac{k^6\delta^6}{6!6}\right).$$

Using the Taylor expansion of the function $\sqrt{1+x}$ we get

$$\omega \approx \sqrt{\frac{c}{2\rho}}\left[\delta k - \frac{1}{48}\delta^3k^3 + \left(\frac{1}{2160} - \frac{1}{4608}\right)\delta^5k^5\right], \tag{3.5}$$

which is consistent with the nonlocal molecular dynamics model (3.3) under the assumption of $N \gg 1$ and $a \ll 1$, s.t. $Na = \delta$, using the relation $c = K/a^2$. Thus, by preserving the nonlocality of the MD model, our upscaled PD model (3.4) has produced the same dispersion relationship as the MD model (3.1).

4. Relating Peridynamics and Molecular Dynamics through Higher Order Gradient Continuum Models. In this section we derive higher order gradient equations of motion for both the MD and the PD models and show that for a continuum nonlocal approximation, the equations of motion agree between the models.

4.1. Higher Order Gradient Continuum Model for Molecular Dynamics. To develop a higher order gradient PDE for the nonlocal molecular dynamics model (3.1), we perform a Taylor expansion in a similar way to the direct expansion technique [5, 1]. Starting with the 1-D nonlocal linear chain (3.1), we perform a Taylor expansion to get (up to sixth order) the expression

$$m\ddot{u}(x,t) = 2K\Big[\Big(\sum_{j=1}^{N} j\Big)\frac{1}{2!}au''(x,t) + \Big(\sum_{j=1}^{N} j^3\Big)\frac{1}{4!}a^3u^{(4)}(x,t) + \Big(\sum_{j=1}^{N} j^5\Big)\frac{1}{6!}a^5u^{(6)}(x,t)\Big].$$

Using closed form expressions for the summations we can write

$$\begin{aligned} m\ddot{u}(x,t) = 2K\Big[&\Big(\frac{N^2}{2} + \frac{N}{2}\Big)\frac{1}{2!}au''(x,t) + \Big(\frac{N^4}{4} + \frac{N^3}{2} + \frac{N^2}{4}\Big)\frac{1}{4!}a^3u^{(4)}(x,t) \\ &+ \Big(\frac{N^6}{6} + \frac{N^5}{2} + \frac{5}{12}N^4 - \frac{N^2}{12}\Big)\frac{1}{6!}a^5u^{(6)}(x,t)\Big]. \end{aligned}$$

As in section 3.1, we take $N \gg 1$ (keeping only the dominant terms), and use the mass-density relation $m = \rho a$ to get a higher order gradient continuum model

$$\ddot{u}(x,t) = \frac{2K}{a^2\rho}\Big[\frac{(Na)^2}{2!2}\frac{d^2u}{dx^2} + \frac{(Na)^4}{4!4}\frac{d^4u}{dx^4} + \frac{(Na)^6}{6!6}\frac{d^6u}{dx^6}\Big], \tag{4.1}$$

which agrees with the higher order gradient continuum model (4.2) derived for the PD model in the next section.

4.2. Higher Order Gradient Continuum Model for Peridynamics. To develop a higher order gradient PDE for peridynamics, we start from (3.4). Following [2], we use a Taylor expansion to get the expression

$$\rho\ddot{u} \approx \int_{-\delta}^{\delta} \frac{c}{|\epsilon|}(u'(x,t)\epsilon + \frac{1}{2!}u''(x,t)\epsilon^2 + \frac{1}{3!}u'''(x,t)\epsilon^3 + \frac{1}{4!}u^{(4)}(x,t)\epsilon^4)d\epsilon.$$

All the odd powers cancel due to symmetry, leaving

$$\begin{aligned} \rho\ddot{u} &\approx \int_{-\delta}^{\delta} c(\frac{1}{2!}u''(x,t)|\epsilon| + \frac{1}{4!}u^{(4)}(x,t)|\epsilon|^3 + \frac{1}{6!}u^{(6)}(x,t)|\epsilon|^5)d\epsilon \\ &= 2\int_{0}^{\delta} c(\frac{1}{2!}u''(x,t)\epsilon + \frac{1}{4!}u^{(4)}(x,t)\epsilon^3 + \frac{1}{6!}u^{(6)}(x,t)\epsilon^5)d\epsilon. \end{aligned}$$

Therefore, up to the approximation order presented, we get the expression

$$\ddot{u}(x,t) = \frac{2c}{\rho}\Big[\frac{\delta^2}{2!2}\frac{d^2u}{dx^2} + \frac{\delta^4}{4!4}\frac{d^4u}{dx^4} + \frac{\delta^6}{6!6}\frac{d^6u}{dx^6}\Big], \tag{4.2}$$

where we recover (4.1), the same higher order gradient PDE as in the nonlocal molecular dynamics case, using the relations $\delta = Na$ and $c = K/a^2$ as in section 3.2. We have matched the equations of motion (4.1) and (4.2) to the sixth order, although one can do this to arbitrary order.

5. Velocity Verlet Stability Analysis. We perform a stability analysis on the velocity Verlet algorithm we implement for our molecular dynamics and peridynamics simulations. We will use this analysis to choose a refinement path for our numerical experiments in the next section.

Following [3], instead of performing the stability analysis on the velocity Verlet algorithm, we can perform it on the equivalent equation

$$m\frac{u_i^{n+1} - 2u_i^n + u_i^{n-1}}{(\Delta t)^2} = \sum_p F_{p,j}, \tag{5.1}$$

with m the particle's mass, Δt the time resolution, $u_j^n = u(x_j, t^n)$ the displacement at time t^n of a particle that was at x_j in the reference configuration, and $F_{p,j}$ the force that particle p exerts on particle j. Following our MD model (3.1), we implement a linear spring force of the form

$$F_{p,j} = \frac{K}{|x_p - x_j|}(u_p - u_j).$$

We are interested in mesh refinement and thus prefer to work with densities, instead of masses. We therefore replace m by $\rho\Delta x$ (with ρ the mass density of the system and Δx the spatial resolution) in (5.1). We let $u_j^n = \zeta^n e^{ikj}$ as in a standard Von Neumann stability analysis, obtaining

$$\frac{\rho}{(\Delta t)^2}\left(\zeta - 2 + \zeta^{-1}\right) = \sum_{p=j-N}^{j+N} \frac{K}{(p-j)\Delta x^2}\left(e^{ik(p-j)} - 1\right),$$

with N the number of neighbor interactions. Using the notation $q = p - j$ [8] we write

$$\frac{\rho}{(\Delta t)^2}\left(\zeta - 2 + \zeta^{-1}\right) = \sum_{q=-N}^{N} \frac{K}{q\Delta x^2}\left(e^{ikq} - 1\right) = \sum_{q=1}^{N} 2\frac{K}{q\Delta x^2}\left(\cos(kq) - 1\right) \equiv -2M_k,$$

and we notice that $M_k \geq 0$. This reduces to the quadratic equation

$$\zeta^2 - 2\left(1 - M_k\frac{(\Delta t)^2}{\rho}\right)\zeta + 1 = 0. \tag{5.2}$$

Solving (5.2) and requiring that $|\zeta| \leq 1$ leads to the stability condition

$$\Delta t < \sqrt{2\rho/M_k} = \sqrt{\frac{2\rho}{\sum_{q=1}^{N} \frac{K}{q\Delta x^2}\left(1 - \cos(kq)\right)}}.$$

We pick the smallest Δt for this condition to be satisfied for all k, and replace $(1 - \cos(kq))$ by 2 to get the stability condition

$$\Delta t < \sqrt{\frac{\rho}{K\sum_{q=1}^{N} \frac{1}{q}}}\Delta x. \tag{5.3}$$

This will give us an appropriate refinement path for our numerical experiments in the next section.

6. Numerical Results. Here we present a few simulation results (produced in a 1-D Matlab code) showing peridynamics as an upscaling of molecular dynamics. We show that the numerical dispersion appearing in MD is preserved for the case of the continuum PD solution, in contrast to the CM wave equation

$$\ddot{u}(x,t) = \frac{K}{\rho} u''(x,t). \tag{6.1}$$

In addition, the dispersion effects are also preserved using a coarse grid for the peridynamic model. For the purposes of this paper, we take the high resolution solutions as numerically "exact".

Following [1], we choose our domain to be $\Omega = [0, 1000]$. The initial displacement profile is defined by $u(x, 0) = p(x)$ for all $x \in \Omega$, where $p(x)$ is a smooth 21th-order polynomial such that $p \equiv 0$ on $[0, 490] \cup [510, 1000]$, $p(500) = 1$, and $p'(x) = p''(x) = ... = p^{(10)}(x) = 0$ for $x = 490, 500, 510$. The initial profile is presented in Figure 6.1.

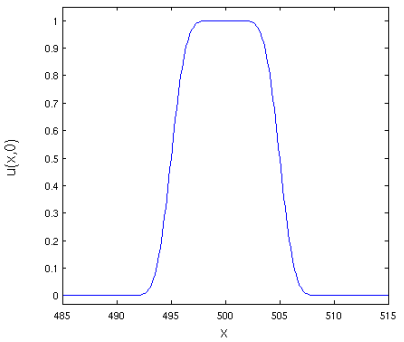


FIG. 6.1. *Initial profile over the range* [485, 515]

Because we implement a multiple neighbor interaction, we need to determine how the force constant K in (3.1) changes for different numbers of neighbor interactions N. Following [6], we implement a uniform force constant choice that depends on the number of neighbor interactions as

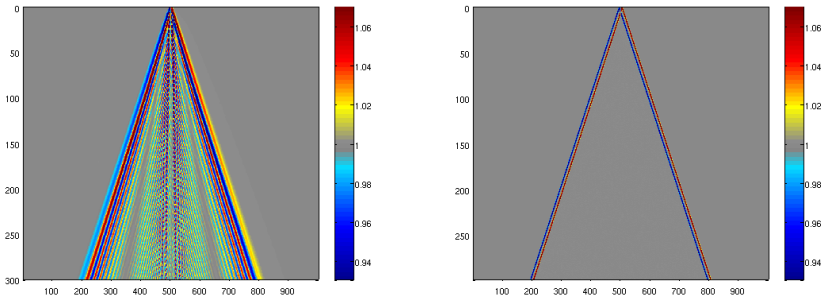
$$K = \frac{2K_a}{N(N+1)},$$

with K_a the force constant for the case of nearest neighbor interaction. In addition, we choose a specific refinement path where the time refinement is connected to the spatial refinement. The relation used is based on the stability condition (5.3), with the form

$$\Delta t = \Delta t_0 \sqrt{\frac{\rho}{K \sum_{i=1}^{N} \frac{1}{i}}} \Delta x,$$

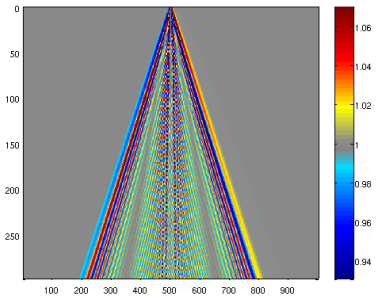
with $\Delta t_0 = 0.85$ chosen to get a stable time step.

In Figure 6.2 we compare the results of a molecular dynamics simulation on $4,001$ atoms with 20 neighbor interactions, a high resolution solution of the peridynamics model ($100,001$ particles with 500 neighbor interactions), a coarse peridynamics simulation ($2,001$ particles with 10 neighbor interactions), and a continuum mechanics high resolution solution ($200,001$ nodes with 20 neighbor interactions).

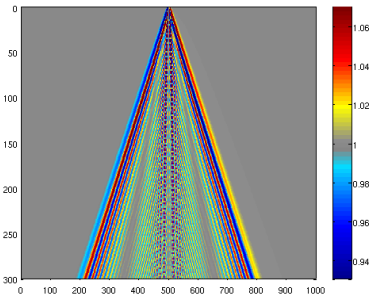


(a) MD: $N_x = 4,001$; $N = 20$

(b) CM: $N_x = 200,001$; $N = 20$



(c) Fine PD: $N_x = 100,001$; $N = 500$



(d) Coarse PD: $N_x = 2,001$; $N = 10$

FIG. 6.2. *Density evolution for the MD, CM and PD (fine and coarse) cases. The time is represented in the y-axis (from top to bottom), and the x-axis represents the reference configuration. The colors represent the value of* $(y'(x,t))^{-1}$*, corresponding to the density, with* $y(x,t)$ *the current position at time* t *of a point at* x *in the reference configuration.* N_x *is the number of atoms(a), nodes(b), or particles(c,d), and* N *the number of neighbor interactions*

In the CM simulation the horizon tends to 0 producing a local model, in contrast to the peridynamic approach which keeps a constant horizon of $\delta = 5$, producing a nonlocal interaction. As we can see, the MD simulation produces similar dispersive effects to the "exact" solution of peridynamics, in contrast to the continuum mechanics model (6.1) in which no dispersion occurs. In addition, the peridynamics approach allows us to solve our system in a coarser mesh which is less computationally expensive. In Table 6.1 we compare the computational cost of the coarse PD simulation in relation to the MD simulation, showing the PD simulation incurs only $1/5$ the cost of the MD simulation.

TABLE 6.1

PD and MD computational costs. N_x *the number of atoms/particles,* N *the number of neighbor interactions, and* N_t *the number of timesteps. The cost of a simulation is estimated by* $N_x \cdot N \cdot N_t$*. The relative cost is calculated in relation to the MD simulation cost.*

Model	N_x	N	N_t	Relative cost
MD	4,001	20	184	1.00
PD (coarse)	**2,001**	**10**	**162**	**0.22**

7. Conclusions. We have introduced the peridynamic model as an upscaling of molecular dynamics. We have shown that the dispersion relations and higher order gradient PDE obtained from MD and PD are consistent to leading order, when the horizon length of the

interaction is preserved. In particular, we have presented numerical experiments showing that the dispersion effects appearing in MD simulations are recovered in PD simulations, in contrast to classical continuum mechanics, where the dispersion effects disappear.

REFERENCES

[1] M. Arndt and M. Griebel, *Derivation of higher order gradient continuum models from atomistic models for crystalline solids*, Multiscale Model. Simul., 4 (2005), pp. 531–562.

[2] E. Emmrich and O. Weckner, *On the well-posedness of the linear peridynamic model and its convergence towards the Navier equation of linear elasticity*, Commun. Math. Sci., 5 (2007), pp. 851–864.

[3] E. Hairer, C. Lubich, and G. Wanner, *Geometric numerical integration illustrated by the Stomer/Verlet method*, Acta Numerica, (2003), pp. 1–51.

[4] M. L. Parks, R. B. Lehoucq, S. J. Plimpton, and S. A. Silling, *Implementing peridynamics within a molecular dynamics code*, Comp. Phys. Comm., (2008).

[5] P. Rosenau, *Dynamics of dense lattices*, Phys. Rev. B, 36 (1987), pp. 5868–5876.

[6] P. Seleson and M. Gunzburger, *Bridging methods and boundary treatment for AtC coupling problems*, In preparation.

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

[8] S. A. Silling and E. Askari, *A meshfree method based on the peridynamic model of solid mechanics*, Computers and Structures, 83 (2005), pp. 1526–1535.

[9] O. Weckner and R. Abeyaratne, *The effect of long-range forces on the dynamics of a bar*, Journal of the Mechanics and Physics of solids, 53 (2005), pp. 705–728.

MODELLING QUANTUM EFFECTS FOR CHARON

JIHAN KIM* AND GARY L. HENNIGAN†

Abstract. We provide an overview of the mathematical models included in Charon, a numerical simulator code used to model electrical semiconductor devices. We discuss changes that need to be implemented in order to add quantum effects to Charon. Specifically, a numerical scheme based on the coupling of the Schrödinger and the Poisson equations is presented. Finally, we provide results of the electron density distribution at the semiconductor-insulator interface in the n-channel MOSFET (metal-oxide-semiconductor field-effect transistor), which is obtained using a simple code written in MATLAB. Results show that taking into account the quantum effects, the peak density values are located a few nm away from the interface, differing from the classical density distribution that predicts a peak at the interface.

1. Introduction. As the size of electronic device dimensions shrinks, quantum mechanical effects become more prominent and interfere with proper functioning of the devices[3]. With these phenomena looming ahead, it's important to obtain an accurate physical picture of the system in order to correctly capture and to predict device characteristics. For example, in minituarized MOSFET devices, significant discrepancies exist between predictions using classical and quantum physics in the charge densities located at the semiconductor-insulator interface[4]. This difference becomes more pronounced near inversion gate voltage bias when carrier concentrations typically are on the same order of magnitude or exceed the dopant concentration of the bulk semiconductor. Using the classical model, the peak value of the densities is located at the semiconductor-insulator interface whereas in the more accurate quantum mechanical model, the peak densities occur away from the interface. As a result, the effective gate oxide length in the quantum mechanical picture is larger, resulting in smaller gate capacitance. Also, since the eigenenergies are quantized along the confinement direction, the ground state eigenenergy level is located above the conduction band edge, effectively increasing the bandgap energy of the bulk material. Ultimately, these differences manifest themselves in measurable device electrical characteristics such as the threshold voltage, drain currents, and gate leakage currents. Thus, it's important to take into account quantum effects in order to obtain a more accurate characterization of minituarized devices.

The goal of our project is to add quantum mechanics to Charon, which is an in-house developed device simulator code. It solves the Poisson and the drift–diffusion electron and hole currents for various devices such as diodes, BJTs, and MOSFETs. In Section 2, we describe the mathematical models used in Charon, which is based on both the finite element method (FEM) and the finite volume method (FVM). For the paper, we focus mainly on the FEM. In Section 3, formulation of the coupled Schrödinger and the Poisson equations is presented, and the numerical methods used to solve the equations are outlined. In Section 4, we show results of the total carrier densities in the channel of a simple n-channel 1D and 2D MOSFET devices. We show that the distribution of the carrier concentration agrees well with models predicted by quantum mechanics. All of the results shown in Section 4 are obtained using a MATLAB code and not in Charon as it is still an ongoing process to add quantum mechanics to Charon.

2. Drift Diffusion Model of Charon. The PDE residuals of the Poisson and the drift-diffusion equations are given as follows[1].

$$R_\psi(n, p, \psi) = \nabla \cdot \lambda^2 \nabla\psi + (p - n + N_d - N_a) \tag{2.1}$$

*University of Illinois at Urbana-Champaign, jihankim@uiuc.edu
†Sandia National Laboratories, glhenni@sandia.gov

$$R_n(n,p,\psi) = -\nabla\cdot(\mu_n n\nabla\psi + D_n\nabla n) + \frac{\partial n}{\partial t} + r(\psi,n,p) \tag{2.2}$$

$$R_p(n,p,\psi) = -\nabla\cdot(\mu_p p\nabla\psi + D_p\nabla p) + \frac{\partial p}{\partial t} + r(\psi,n,p) \tag{2.3}$$

λ is the minimal Debye length of the device, D_n and D_p are electron and hole diffusivities, μ_n and μ_p are electron and hole mobilities, and r is the recombination term. ψ, n, p, N_d, and N_a represent the electric potential, electron and hole concentration, and donor and acceptor concentration. Both the Dirichlet and the Neumann boundary conditions are used in Charon with the boundary $\Gamma = \partial\Omega$, which is composed of two disjoint parts Γ_D and Γ_N. Ω is the total domain boundary.

$$n = n_D, p = p_D, \quad \text{on } \Gamma_D \tag{2.4}$$

$$(\mu_n n\nabla\psi + D_n\nabla n)\cdot\mathbf{n} = 0, \quad \text{on } \Gamma_N \tag{2.5}$$

$$(\mu_p p\nabla\psi + D_p\nabla p)\cdot\mathbf{n} = 0, \quad \text{on } \Gamma_N \tag{2.6}$$

The finite element method (FEM) is used to discretize the equations. The corresponding weak form of the coupled Poisson and the drift-diffusion equations can be represented as follows

$$F_\psi(n,p,\psi,\hat{\psi}) = \int R_\psi\hat{\psi}d\Omega, \quad \text{in } \Omega \tag{2.7}$$

$$F_n(n,p,\psi,\hat{n}) = \int R_n\hat{n}d\Omega + \langle R_n, W(\hat{n})\rangle, \quad \text{in } \Omega \tag{2.8}$$

$$F_p(n,p,\psi,\hat{p}) = \int R_p\hat{p}d\Omega + \langle R_p, W(\hat{p})\rangle, \quad \text{in } \Omega \tag{2.9}$$

where $\hat{\psi}$, $\hat{n}$, and $\hat{p}$ are test functions. Because the drift-diffusion equations contain both the advective and diffusive fluxes, they require a special treatment of upwind stabilization term, indicated by the inner product terms in eqs. (2.8) and (2.9). $W(\cdot)$ is a suitable weighting function that provides additional viscosity. Applying the divergence theorem, we can transform the weak form equations into following equations.

$$\begin{aligned} F_\psi = \int R_\psi\hat{\psi}d\Omega = &-\int \lambda^2\epsilon_r\nabla\psi\cdot\nabla\hat{\psi}d\Omega \\ &+\int_\Omega \lambda^2\hat{\psi}\epsilon_r\nabla\psi\cdot\hat{\eta}d\Omega \\ &+\int (p-n+N_D-N_A)\hat{\psi}d\Omega, \quad \text{in } \Omega \end{aligned} \tag{2.10}$$

$$\begin{aligned} F_n &= \int R_n\hat{n}d\Omega + \langle R_n, W(\hat{n})\rangle \\ &= \int \frac{\partial n}{\partial t}\hat{n}d\Omega - \int \nabla\cdot(-\mu_n n\nabla\psi + D_n\nabla n)\hat{n}d\Omega + \int R\hat{n}d\Omega + \langle R_n, W(\hat{n}\rangle \\ &= \int \frac{\partial n}{\partial t}\hat{n}d\Omega - \int \mu_n n\nabla\psi\cdot\nabla\hat{n}d\Omega + \int D_n\nabla n\cdot\nabla\hat{n}d\Omega \\ &+ \int_\Omega (\mu_n n\nabla\psi - D_n\nabla n)\hat{n}\cdot\hat{\eta}d\Omega + \int R\hat{n}d\Omega + \langle R_n, W(\hat{n})\rangle, \quad \text{in } \Omega \end{aligned} \tag{2.11}$$

$$F_p = \int \frac{\partial p}{\partial t}\hat{p}d\Omega + \int \mu_p p\nabla\psi\cdot\nabla\hat{p}d\Omega + \int D_p\nabla p\cdot\nabla\hat{p}d\Omega - \int_\Omega(\mu_p p\nabla\psi - D_p\nabla p)\hat{p}\cdot\hat{\eta}d\Omega + \int R\hat{p}d\Omega + \langle R_p, W(\hat{p})\rangle, \quad \text{in } \Omega \tag{2.12}$$

Finally, the Newton-Kantorovich formulation is used to linearize the equations.

$$F(x^*) = 0 \tag{2.13}$$

$$F_x^k x'^{k+1} = -F^k \tag{2.14}$$

$$x^{k+1} = x^k + x'^{k+1} \tag{2.15}$$

The resulting Jacobian in the Newton-Kantorovich formulation looks as follows.

$$\begin{pmatrix} \mathbf{D}_\psi & \mathbf{-1} & \mathbf{1} \\ \mathbf{F}_{n\psi} & (\frac{\mathbf{M}_n}{\Delta t} + \mathbf{C}_n + \mathbf{R}_{nn}) & \mathbf{R}_{np} \\ \mathbf{F}_{p\psi} & \mathbf{R}_{pn} & (\frac{\mathbf{M}_p}{\Delta t} + \mathbf{C}_p + \mathbf{R}_{pp}) \end{pmatrix} \tag{2.16}$$

where

$$\mathbf{D}_\psi = -\int \lambda^2\bar{\epsilon}_r\nabla\psi'\cdot\nabla\hat{\psi}d\Omega, \quad \text{in } \Omega \tag{2.17}$$

$$\mathbf{F}_{n\psi} = -\int \bar{\mu}_n\bar{n}\nabla\psi'\cdot\nabla\hat{n}d\Omega, \quad \text{in } \Omega \tag{2.18}$$

$$\mathbf{M}_n = \int \alpha n'\hat{n}d\Omega, \quad \text{in } \Omega \tag{2.19}$$

$$\mathbf{C}_n = -\int \bar{\mu}_n n'\nabla\hat{\psi}\cdot\nabla\hat{n}d\Omega + \int \bar{D}_n\nabla n'\cdot\nabla\hat{n}d\Omega, \quad \text{in } \Omega \tag{2.20}$$

$$\mathbf{R}_{nn} = \int n'\frac{\partial R}{\partial n}d\Omega, \quad \text{in } \Omega \tag{2.21}$$

$$\mathbf{R}_{np} = \int p'\frac{\partial R}{\partial p}d\Omega, \quad \text{in } \Omega \tag{2.22}$$

$$\mathbf{F}_{p\psi} = \int \bar{\mu}_p\bar{p}\nabla\psi'\cdot\nabla\hat{p}d\Omega, \quad \text{in } \Omega \tag{2.23}$$

$$\mathbf{M}_p = \int \alpha p'\hat{p}d\Omega, \quad \text{in } \Omega \tag{2.24}$$

$$\mathbf{C}_p = -\int \bar{\mu}_p p'\nabla\hat{\psi}\cdot\nabla\hat{p}d\Omega + \int \bar{D}_p\nabla p'\cdot\nabla\hat{p}d\Omega, \quad \text{in } \Omega \tag{2.25}$$

$$\mathbf{R}_{pp} = \int p'\frac{\partial R}{\partial p}d\Omega, \quad \text{in } \Omega \tag{2.26}$$

$$\mathbf{R}_{pn} = \int n'\frac{\partial R}{\partial n}d\Omega, \quad \text{in } \Omega \tag{2.27}$$

The preconditioned GMRES solver is used to solve the systems of linear equations where the preconditioner typically used is an ILUT, an incomplete LU preconditioner.

3. Quantum mechanical numerical Model. In order to add quantum mechanics to Charon, we first simplify the problem by looking at only the time independent domain. As a result, the electron and the hold drift-diffusion equations can be taken out from the system of equations. The general Hamiltonian for N electrons in the system is written in eqn. (3.1),

$$H = \sum_{j=0}^{n} \frac{(-i\hbar\vec{\nabla}_j - \frac{e}{c}\vec{A}_j)^2}{2m^*} + E_c(\vec{r}_j) \tag{3.1}$$

where the effective-mass approximation is used to describe the many-body effects along the electrons in the conduction band in the two-dimensional (2D) xy plane, and the extension of the wavefunction along the z-direction is neglected[2]. $E_c = -q\phi + \Delta E_c$, where ϕ is the potential energy of the system for a given material and ΔE_c is the bandgap offset at different material interfaces. For this analysis, the material is assumed to be homogeneous (Silicon), and thus we can ignore the offset term. Two values of effective mass are used: $m^* = 0.916m^0$ and $0.19m^0$ where m^0 is the free electron mass since Silicon has longitudinal and transverse valleys corresponding to different effective masses. A is the vector potential term associated with the magnetic fields and for this paper, we assume zero magnetic fields, which gets rid of this term.

From the Hamiltonian, we can construct the time-independent Schrödinger Equation

$$H\psi = E\psi \tag{3.2}$$

where ψ is the electron wavefunction and E is the eigenvalue associated with the wavefunction. We convert the equation into a weak formulation and using the Petrov-Galerkin method, we discretize this weak formulation via the finite element method. We can designate a "quantum region" where electron wavefunctions predominate and solve the Schrödinger equation in just this sub-domain. For this work, however, the "quantum region" is designated to be the entire numerical domain for simplicity. Eventually, we would want to create a separate quantum domain in order to provide more accurate boundary conditions and to improve computational speed. For the boundary conditions, we assume that outside the Schrödinger region, the wavefunctions disappear and thus use the Dirichlet boundary condition with $\psi = 0$ at the boundaries for all eigenvector modes. The electron wavefunctions and the eigenvalues are calculated using the ARPACK package, which is based on the Implicitly Restarted Arnoldi Method. In practice, only the few smallest (in magnitude) eigenvalues and eigenvectors are calculated since the higher-state contributions to the total density are small. After obtaining the eigenenergies and the corresponding wavefunctions, we can solve the electron densities using the following equation

$$n(r) = \int_{E_c}^{\infty} g(E)f(E)dE \tag{3.3}$$

where $g(E)$ is the density of states and $f(E)$ is the Fermi-Dirac distribution of the electrons. Because of quantization effects, the density of states transform into a summation of Dirac delta functions.

$$g(E) = \sum_i \frac{2g_i m_i^*}{\pi\hbar^2} \sum_j \delta(E - E_{ij})|\psi_j(r)|^2 \tag{3.4}$$

A factor of 2 is needed in front of the summation in order to correctly take into account spin degeneracies for each state. g_i is the degeneracy factor pertaining i^{th} valley of the Silicon

energy band. Substituting this expression for the density of states into eqn.(3.3), we obtain an analytical expression for the electron density.

$$n(r) = \int_{E_c}^{\infty} g(E)f(E)dE = \sum_i \frac{2g_i m_i^*}{\pi\hbar^2} \sum_j |\psi_j(r)|^2 f(E_{ij}) \tag{3.5}$$

Now, we can solve the Poisson equation using the electron densities derived from eqn.(3.5).

$$-\nabla^2\phi = \frac{q}{\epsilon_{Si}} C \tag{3.6}$$

where $C = N_d - N_a + n - p$.

The Silicon dielectric constant is set to $\epsilon_{Si} = 11.8\epsilon_o$. Finally, we obtain the potential and plug it back into the Schrödinger equation and iterate until a sufficient convergence criterion is met.

4. Results. First, we take a look at a simple 1D MOSFET structure along the direction perpendicular to the plane of the oxide-semiconductor interface. We can ignore the presence of drain and source along this particular direction. We set the gate biases to strong inversion mode where the carrier concentration along the channel becomes predominantly electrons in this particular case (n-channel). Temperature is set to 300 K and we use three different acceptor concentrations ($N_a = 4\text{x}10^{16}, N_a = 6\text{x}10^{16}$, and $N_a = 8\text{x}10^{16} cm^{-3}$) in the bulk semiconductor. The device is simulated from the oxide-semiconductor interface to the edge of the depletion layer, and the electron concentration is assumed to be zero at the boundaries. This is a good first-order assumption since the potential profile along the numerical domain looks like a triangular quantum well.

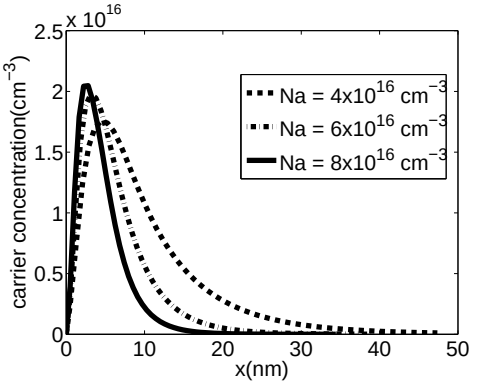


FIG. 4.1. *Charge distribution in the inversion layer of the MOS device. The simulations were conducted using 400 elements. The oxide-semiconductor interface is located at x = 0 nm. From the figures, we can see that the location of the peak concentration moves further away from the interface for lower dopant concentrations*

The electron charge distribution obtained from solving the 1D coupled Poisson Schrodinger equations are plotted in Figure 4.1. We see that for all dopant concentrations, the peak density level is not located at the interface, which disagrees with the predictions from the classical model. Also, we observe that smaller the dopant concentrations, the further the peak is from the interface. This result is elaborated in Figure 4.2.

For $N_a = 5\text{x}10^{15} cm^{-3}$, the peak carrier concentration is located 3.87 nm away from the interface. Upon increasing the dopant concentration, we can see a monotonic decrease in the peak concentration location. At $N_a = 3\text{x}10^{17} cm^{-3}$, the carrier concentration is located 1.6 nm

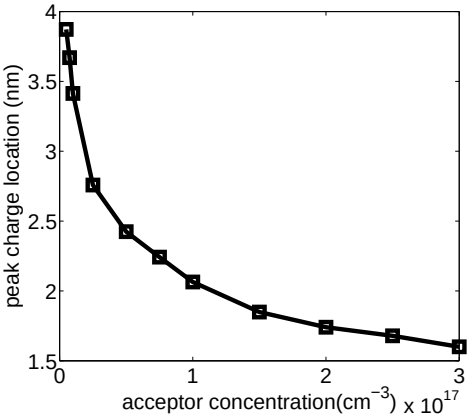


FIG. 4.2. *Peak location of the electron densities. Overall, the peak values decrease monotonically with respect to dopant concentration. The slope, indicating the rate of the change, changes significantly for within these values of dopant concentrations.*

from the interface. Overall, the change in the peak location is more drastic for lower dopant concentration as indicated by a steeper slope whereas the change becomes smaller for higher dopant concentration.

Next, we take a look at a more realistic case of a 2D MOSFET structure along a plane perpendicular to the oxide-semiconductor interface plane. The length of the channel separating the drain and source were set at 20 nm. The potential difference between the drain and the source (V_{DS}) was set at 0.3 V. We set the acceptor concentration of the Silicon bulk at $N_a = 10^{16} cm^{-3}$. And again, we assumed Dirichlet boundary condition ($\psi = 0$) along the boundaries. Figure 4.3 shows the result of the 2D electron density. Similar to the 1D case, the peak of the density is a few nm away from the potential boundaries.

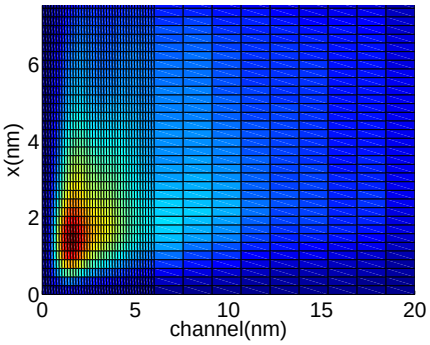


FIG. 4.3. *Electron densities in a two-dimensional MOSFET at strong inversion with $V_{DS} = 0.3$ V. The channel length separating the drain and the source region is set to 20 nm. The density peak is located at around $x = 1.75$ nm and channel $= 1.75$ nm.*

In conclusion, we have shown that using a simple MATLAB code and solving for the Schrödinger and the Poisson equations, the charge density distribution in a simplified MOSFET device looks different from the distribution predicted from the classical model. Specifically, the density peak is away from the oxide-semiconductor interface. The ultimate goal,

however, is to incorporate the quantum effects into Charon and there remains ongoing work to fulfill task.

REFERENCES

[1] P. Bochev, G. Hennigan, and J. Shadid, *Discretization and Solution of the Semiconductor Equations for CHARON*.

[2] D. Melnikov, J. Leburton, A. Taha, and N. Sobh, *Coulomb localization and exchange modulation in two-electron coupled quantum dots*, Physical Review B, (2006).

[3] M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, 2000.

[4] F. Stern and W. E. Howard, *Properties of semiconductor surface inversion layers in the electric quantum limit*, Physical Review, (1967).

CALCULATION OF MELTING POINTS USING ATOMISTIC SIMULATIONS

SAIVENKATARAMAN JAYARAMAN*, EDWARD J. MAGINN†, STEVEN J. PLIMPTON‡, ANATOLE VON LILIENFELD§, AND AIDAN P. THOMPSON¶

Abstract. A new method for the calculation of melting point of a system was implemented in LAMMPS. This implementation was tested on a model Lennard-Jones system, and the results were compared against those obtained from the original code used in the Maginn research group. The data obtained from the LAMMPS simulations, and the melting point computed using them were in close agreement with the results obtained from the original code.

1. Introduction. Melting point prediction is an inherently difficult problem for which not many solutions exist in literature. The melting point is an important property to consider for heat transfer fluids and ionic liquids, and hence knowledge of the melting point will aid in the selection of candidate liquids for such applications. Design of new compounds for liquid applications requires knowledge of the melting point, and a method to calculate it will greatly aid efforts in such a direction.

Computational efforts for calculating melting points are few and there has not been a unique method which has been widely accepted for such a purpose. Alawi and Thompson used a void-induced melting method to compute melting points [2, 1]. In this approach, voids are introduced in the crystal, and a molecular dynamics simulation is conducted on this modified crystal while ramping up the temperature, and the density is noted. When a first order transition is observed, it is reported as the melting point. Another direct method is the simulation of a solid-liquid interface [8, 7, 14, 13, 9], where the temperature and pressure at which a stable interface between the solid and liquid phases is observed is the coexistence point.

An alternative approach to calculation of melting point is the phase equilibrium approach, or the equality of free energy, temperature, and pressure. Some of us have developed and extended a thermodynamic integration procedure, and have applied it to Lennard-Jonesium, sodium chloride, benzene, and triazole [3, 4]. Recently, we have successfully applied this method to compute the melting point of a complex ionic liquid, 1-n-butyl-3-methylimidazolium chloride [5]. After these successes in predicting melting points of complex molecules accurately, we believe the method should be readily available for public use, and hence the present effort to incorporate the method into LAMMPS [11].

2. Methodology. Our procedure to calculate the melting point involves two parts. First, relative Gibbs free energy curves are constructed for the solid and liquid phases by running a series of isothermal-isobaric (NPT) MD simulations, at various temperatures. The enthalpy of each phase is recorded at each of these temperatures, and the Gibbs-Helmholtz equation is integrated to obtain Gibbs free energies relative to an arbitrary reference temperature T_{ref}

$$\frac{G}{RT} - \left(\frac{G}{RT}\right)_{ref} = \int_{T_{ref}}^{T} -\frac{H}{RT^2} dT, \tag{2.1}$$

where G is the Gibbs free energy, H is the enthalpy, and T is the temperature, with R, the gas constant, being chosen according to the units of G and T.

*University of Notre Dame, sjayaram@nd.edu
†University of Notre Dame, ed@nd.edu
‡Sandia National Laboratories, sjplimp@sandia.gov
§Sandia National Laboratories, oavonli@sandia.gov
¶Sandia National Laboratories, athomps@sandia.gov

The second part involves the use of thermodynamic integration[6] to compute the free energy difference between the solid and the liquid phases at the reference temperature. This yields the quantity $(G/RT)_{ref}$ in Eq. 2.1. In this thermodynamic integration method, a five-state path is constructed between the liquid and the solid phases. The five states are:(1) liquid at density corresponding to the pressure of interest; (2) a weakly interacting fluid at the liquid density; (3) a weakly interacting fluid at the crystalline phase density; (4) an ordered weakly interacting state at the crystal density, and (5) the crystal at the pressure of interest. Each pair of consecutive states are connected by a path chosen to provide a smooth variation in free energy along the path. Numerically, this is accomplished by a coupling parameter, λ, such that $\lambda = 0$ corresponds to the initial state, and $\lambda = 1$ corresponds to the final state. A schematic of the thermodynamic path used in this integration procedure is shown in Fig. 2.1, and shows these five states.

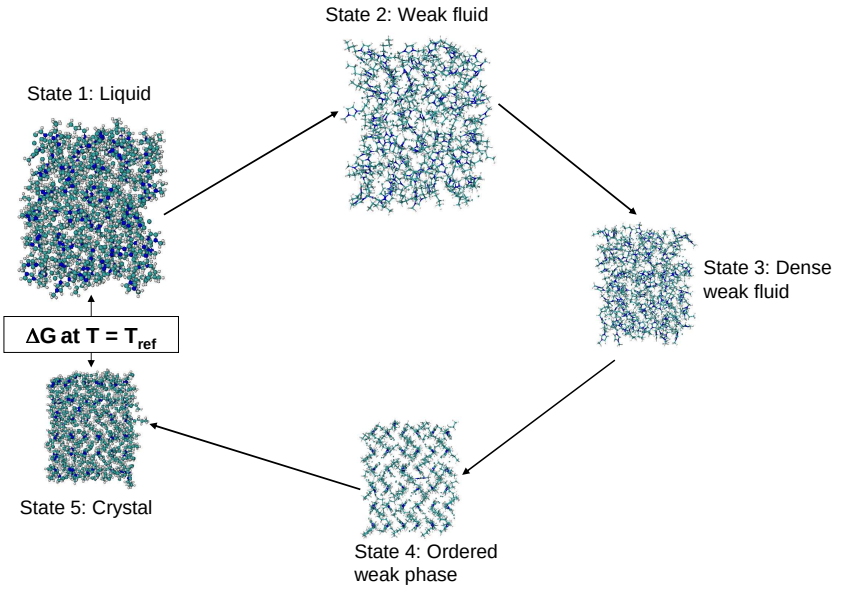


FIG. 2.1. *A schematic of the thermodynamic cycle used to compute free energy difference between the solid and liquid phases.*

The free energy difference is calculated at the pressure at which the melting point is required. In principle, any temperature at which the relative free energy in Eq. 2.1 is available, can be chosen, but in practice the temperature is usually chosen to be T_{ref}. The difference in the quantity $(G/RT)_{ref}$ between the solid and liquid phases is given by

$$\frac{\Delta G}{RT} = \frac{\sum \Delta A}{RT} + \frac{P\Delta V_{s-\ell}}{RT}, \tag{2.2}$$

where $\sum \Delta A$ is the sum of free energy changes for all four transformations in the five-state path (Fig. 2.1), and the $P\Delta V$ term converts this effective Helmholtz free energy difference to

Gibbs free energy. Here, P is the pressure chosen to compute the free energy difference, and ΔV is the difference in volumes between the liquid (state (1)) and the solid (state (5)) phases.

The changes between all adjacent states i and j (except between states 2 and 3) are accomplished by changing the coupling parameter λ, and the Helmholtz free energy is computed using thermodynamic integration as:

$$\Delta A_{i\to j} = \int_0^1 \left\langle \frac{\partial U}{\partial \lambda} \right\rangle_\lambda d\lambda, \tag{2.3}$$

where U is the potential energy of the system and pointed brackets indicate an ensemble average. λ defines a connecting path such that at $\lambda = 0$, state i is recovered, and at $\lambda = 1$, state j is recovered. By definition, the change in free energy depends only on the end points and not the specific path connecting the end points.

The transformation from state (1) to (2) reduces the intermolecular interactions for the liquid phase, at constant density, which is the liquid density at the temperature and pressure of interest. The potential function for this transformation is:

$$U_{1\to 2} = [1 + \lambda(\eta - 1)]^m U^{vdW} + [1 + \lambda(\eta - 1)]^n U^{elec} + U^{NS} \tag{2.4}$$

where η is a scaling parameter between zero and one, m and n are positive integer exponents, U^{vdW} is the potential energy due to van der Waals type interactions, U^{elec} is the potential energy due to electrostatics, and U^{NS} indicates all the other interactions which are not scaled (like bonds, angles and dihedrals), due to their intramolecular nature.

Transformation from state (2) to (3) involves compressing the box, and transforming its shape such that the final box has the density and shape of the crystal at the temperature and pressure of interest. The free energy change for this transformation is calculated as:

$$\Delta A_{2\to 3} = \int_{V^\ell}^{V^S} -\langle P \rangle \, dV, \tag{2.5}$$

where V^ℓ and V^S are the molar volumes of the liquid and solid phases, respectively. The change in volume is accomplished by changing the cell basis matrix as a function of λ as follows:

$$\mathbf{H}(\lambda) = \lambda \mathbf{H}_s + (1 - \lambda)\mathbf{H}_\ell, \tag{2.6}$$

where $\mathbf{H}_s$ and $\mathbf{H}_\ell$ are the solid and liquid cell basis matrices, respectively. The cell basis matrix, $\mathbf{H}$, is the matrix whose columns are the cell basis vectors $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$. λ is s coupling parameter equivalent to that used in Eq. 2.3.

Transformation to state (4) involves ordering the system, which is achieved by imposing an external potential field that tethers the ion centers of mass and selected atoms to lattice sites corresponding to the solid structure (state (5)). This external potential field should mimic the actual crystal as closely as possible. This can be achieved by analyzing how the centers of mass fluctuate around their average lattice positions in an NVT simulation of the crystal. These fluctuations are assumed to follow a three-dimensional harmonic oscillator, the probability distribution for which is given by:

$$P(r) = \left(\frac{\beta\kappa}{\pi}\right)^{3/2} 4\pi r^2 \exp(-\beta\kappa r^2) dr. \tag{2.7}$$

We fit this equation to the centers of mass fluctuations, and hence obtain κ. More details about this three-dimensional harmonic potential function can be found in Refs. [4] and [5].

The potential energy function for transformation between states (3) and (4) is given by:

$$U_{3\to4}(\lambda) = \eta^m U^{vdW} + \eta^n U^{elec} + U^{NS} - \lambda \sum_i \sum_j a_{ij} \exp(-b_{ij} r_{ij}^2), \tag{2.8}$$

where the various U have the same definitions as stated in Eq. 2.4. The well depth is a_{ij}, and the well width is $1/\sqrt{b_{ij}}$, where $a_{ij} = \kappa/b_{ij}$ [4].

For the transformation between states (4) and (5), the tethering potential is turned off and full intermolecular interactions are restored as λ goes from 0 to 1. The potential energy is given by:

$$U_{4\to5}(\lambda) = [\eta + \lambda(1-\eta)]^m U^{vdW} + [\eta + \lambda(1-\eta)]^n U^{elec}$$

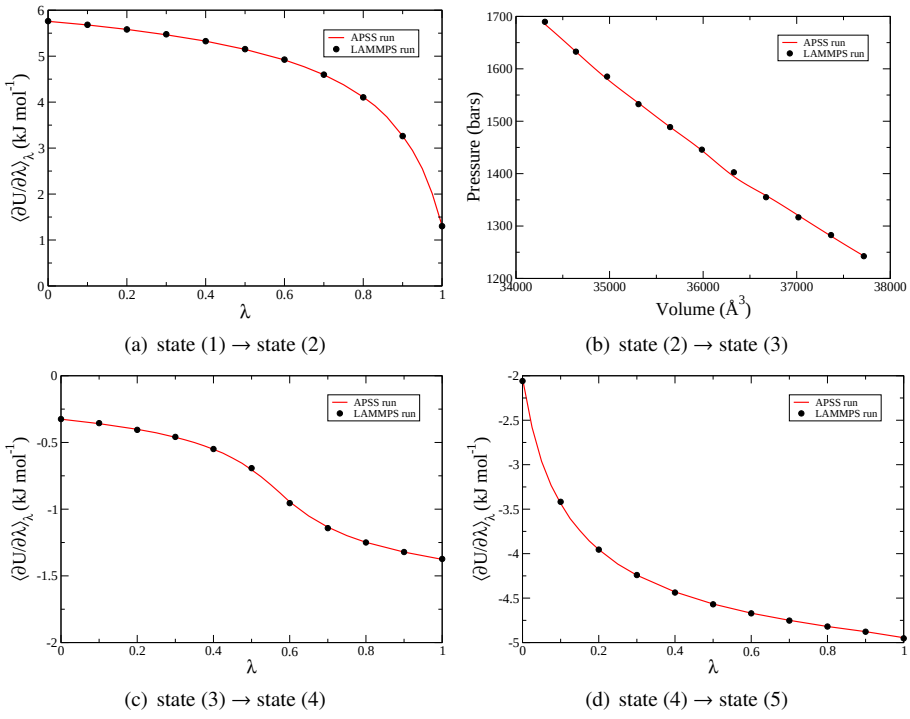
$$+U^{NS} - (1-\lambda) \sum_i \sum_j a_{ij} \exp(-b_{ij} r_{ij}^2). \tag{2.9}$$

3. Simulation details. LAMMPS[11] was chosen to implement this method to take advantage of its massively parallel infrastructure and also of its efficient handling of molecular dynamics simulations. To implement thermodynamic integration into LAMMPS, three new pair potentials had to be created. A `pair/lj/cut/alchemy` scales the Lennard-Jones interactions (force and energy) by the required scaling factors. A `pair/coul/long/alchemy` scales the coulombic interactions (force and energy) by the required scaling factors. The scaling factors for these two interactions have been listed in Eqs. 2.4, 2.8, and 2.9. The third pair potential which was created was `pair/gauss`, which imposes a gaussian-type potential field according to Eqs. 2.8 and 2.9. A LAMMPS fix style `fix/alchemy` was created to control λ, and also to change the simulation cell shape and volume for the transformation between states (2) and (3). A `compute/tideriv` was created to obtain $\langle \partial U/\partial\lambda \rangle_\lambda$ for each transformation. `pppm/alchemy` had to be created to introduce the coulombic scaling to the reciprocal Ewald terms.

The test system chosen was the same Lennard-Jonesium test system used in Ref. [5]. The simulations in that study were conducted using APSS (software developed by the Maginn group which has been verified against DL_POLY [12] and NAMD [10]). Since the goal of running these test simulations is to verify the implementation of the thermodynamic integration method, the results for the NPT simulations, and also the initial setup for the thermodynamic integration were taken from these earlier "APSS runs". The system was an 864 atom Lennard-Jonesium. A $6 \times 6 \times 6$ fcc supercell was simulated for the solid phase, which was a cube of side 32.49401 Å. All simulations were conducted at $P^* = 1.0$. The thermodynamic cycle calculations were performed using LAMMPS at $T^* = 0.7$, in the canonical ensemble using the Nosé-Hoover thermostat. These numbers correspond to 417 bar and 83.9 K for an argon system. Note that P^* and T^* are the dimensionless Lennard-Jones pressure and temperature given by $P^* = P\sigma^3/\epsilon$ and $T^* = Tk_b/\epsilon$ with k_b being Boltzmann's constant, and σ and ϵ being the Lennard-Jones parameters. The simulations were conducted using a 1 fs timestep, for 100 ps. This was found to be sufficient for equilibration and also to collect statistically meaningful data. For the Lennard-Jones interactions, $\epsilon = 0.238$ kcal/mol, and $\sigma = 3.40$Å were used with a cutoff of 2.8σ, and standard long-range corrections were applied. For the tethering potential used in transformations between states (3) and (4) and between (4) and (5), $\kappa = 0.473$ kcal/mol, and $b = 0.5$Å^{-2} were used.

4. Results. Comparison between simulations run using LAMMPS and those run using APSS are shown in Fig. 4.1. The quantities which contribute to the free energy difference between the solid and liquid phases are compared here. For transformations between (1) &

(2) (Fig. 4.1(a)); (3) & (4) (Fig. 4.1(c)) and (4) & (5) (Fig. 4.1(d)), the plots show $\langle \partial U/\partial \lambda \rangle_\lambda$ against λ, while for the transformation between states (2) and (3), the plot shows $\langle P \rangle$ against V (Fig. 4.1(b)). Rough timing data indicate that LAMMPS is an order of magnitude faster than APSS.



(a) state (1) → state (2)

(b) state (2) → state (3)

(c) state (3) → state (4)

(d) state (4) → state (5)

FIG. 4.1. *Comparison of LAMMPS and APSS across the four different transformations. The red line connects points obtained from simulations run using APSS, while the black data points were obtained from LAMMPS simulations.*

The relative difference in ΔA at each λ point between simulations run using APSS and those run using LAMMPS for the four transformations is shown in Fig. 4.2. Most of the points are within 1%, which indicates a good match between the results obtained from the two codes.

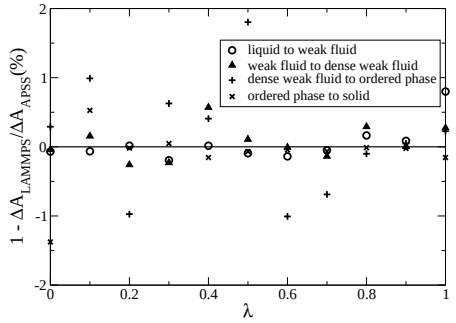


FIG. 4.2. *Relative difference in ΔA between APSS and LAMMPS obtained for the four different transformations as a function of λ.*

Fig. 4.3 shows the variation of $\Delta G/RT$ as a function of T^*. The point at which this curve intersects $\Delta G/RT = 0$ is taken to be the melting point. Assuming ΔG is normally distributed, the dashed lines are an estimate of the standard deviation on $\Delta G/RT$. Further details about calculation of this standard deviation can be found in Ref. [5]. The melting point was then computed to be $T^* = 0.74 \pm 0.02$, which is consistent with the value obtained in the previous study [5]. This translates to $T = 89 \pm 2K$ for an argon system.

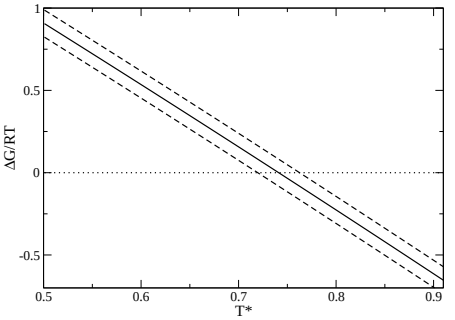


FIG. 4.3. *Plot of $\Delta G/RT$ as a function of T^*. The dotted line indicates $\Delta G/RT = 0$, and the T^* at its intersection with the solid line is the melting point. Uncertainty in $\Delta G/RT$ is indicated by the dashed lines.*

5. Conclusions. A method to compute melting point has been implemented in LAMMPS. In this method, thermodynamic integration is performed along a thermodynamic cycle constructed between the solid and the liquid phases via three intermediate "pseudo-supercritical" states. The method ensures the equality of pressure, temperature and Gibbs free energy between the phases.

The method was tested using a model Lennard-Jones system. The results obtained from the LAMMPS simulations were verified against those from a software package developed by the Maginn group. The difference between the two calculations was about 1%, which is consistent with the inherent statistical uncertainty of the method. This indicates that the method has been correctly implemented in LAMMPS. The melting point for the system was calculated to be $T^* = 0.74 \pm 0.02$ which is consistent with Ref. [5].

The melting points of a variety of materials, notably alkali nitrate salts, can now be calculated. These molten salts find application as heat-transfer fluids in thermal solar power plants. This implementation will also be released as part of the LAMMPS distribution in future.

REFERENCES

[1] S. Alavi and D. L. Thompson, *Molecular dynamics studies of melting and some liquid-state properties of 1-ethyl-3-methylimidazolium hexafluorophosphate [emim][pf$_6$]*, J. Chem. Phys., 122 (2005), p. 154704.

[2] S. Alavi and D. L. Thompson, *Simulations of the solid, liquid, and melting of 1-n-butyl-4-amino-1,2,4-triazolium bromide*, J. Phys. Chem. B, 109 (2005), pp. 18127–18134.

[3] D. M. Eike, J. F. Brennecke, and E. J. Maginn, *Toward a robust and general molecular simulation method for computing solid-liquid coexistence*, J. Chem. Phys., 122 (2005), p. 014115.

[4] D. M. Eike and E. J. Maginn, *Atomistic simulation of solid-liquid coexistence for molecular systems: Application to triazole and benzene*, J. Chem. Phys., 124 (2006), p. 164503.

[5] S. Jayaraman and E. J. Maginn, *Computing the melting point and thermodynamic stability of the orthorhombic and monoclinic crystalline polymorphs of the ionic liquid 1-n-butyl-3-methylimidazolium chloride*, J. Chem. Phys., 127 (2007), p. 214504.

[6] J. G. Kirkwood, *Statistical mechanics of fluid mixtures*, J. Chem. Phys., 3 (1935).

[7] J. R. Morris and X. Y. Song, *The melting lines of model systems calculated from coexistence simulations*, J. Chem. Phys., 116 (2002), pp. 9352–9358.

[8] J. R. Morris, C. Z. Wang, K. M. Ho, and C. T. Chan, *Melting line of aluminum from simulations of coexisting phases*, Phys. Rev. B, 49 (1994), pp. 3109–3115.

[9] E. G. Noya, C. Vega, and E. de Miguel, *Determination of the melting point of hard spheres from direct coexistence simulation methods*, J. Chem. Phys., 128 (2008), p. 154507.

[10] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kale, and K. Schulten, *Scalable molecular dynamics with NAMD*, J. Comput. Chem., 26 (2005), pp. 1781–1802.

[11] S. J. Plimpton, *Fast parallel algorithms for short-range molecular dynamics*, J. Comput. Phys., 117 (1995), pp. 1–19. The LAMMPS WWW site is at http://lammps.sandia.gov.

[12] W. Smith and T. R. Forester, *DL_POLY_2.0: A general-purpose parallel molecular dynamics simulation package*, J. Mol. Graphics, 14 (1996), pp. 136–141.

[13] Z. G. Xia, D. Y. Sun, M. Asta, and J. J. Hoyt, *Molecular dynamics calculations of the crystal-melt interfacial mobility for hexagonal close-packed mg*, Phys. Rev. B, 75 (2007).

[14] S. Yoo, X. Zeng, and J. Morris, *The melting lines of model silicon calculated from coexisting solid-liquid phases*, J. Chem. Phys., 120 (2004), pp. 1654–1656.

CONVERGENCE VERIFICATION OF STATIC SOLVERS VIA ENERGY MINIMIZATION IN LAMMPS

CHRIS HARDEN* AND RICH LEHOUCQ†

Abstract. Convergence verification is a well-known and well-understood mathematical exercise used to build confidence and to perform code verification most typically in the context of solving ordinary and partial differential equations on a mesh. The challenge presented here is an attempt to recover spatial convergence rates for simulations performed in a molecular dynamics setting with no underlying mesh. The approach is to use equivalent formulations on both the atomistic and the continuum scales to develop appropriate problems to be used for convergence testing. The goal is to be able to generate continuum solutions of the field properties of a system by solving an equivalent particle based formulation on a lattice. The simulations will be performed in the open source molecular dynamics package, LAMMPS, developed at Sandia National Laboratories.

1. Introduction. Convergence verification is a mathematical exercise to test that a code or simulation is converging properly in accordance with its analytical properties [4], [2]. The convergence rate is determined by the local truncation error. As a simple illustration consider a second centered difference in space approximation to the second derivative.

$$\frac{d^2u}{dx^2} = \frac{U_{i-1} - 2U_i + U_{i+1}}{h^2} + \frac{h^2}{12}u'''' + H.O.T. \tag{1.1}$$

where h is a uniform mesh spacing and $U_i \approx u(ih)$. The local truncation error τ_u is the error obtained by approximating the second derivative by the discrete centered difference scheme.

$$\|\tau_u\| = \|\frac{d^2u}{dx^2} - \frac{U_{i-1} - 2U_i + U_{i+1}}{h^2}\| \leq \frac{h^2}{12}\|u''''\| \tag{1.2}$$

where $\|\cdot\|$ is a norm appropriate for the space in which the solutions to the problem exist.

If the fourth derivative exists, then the second centered difference method is said to be of second order, and thus we expect to recover a second order rate of convergence.

1.1. Measuring the Rate of Convergence. In practice, we measure the rate of convergence by cooking up an exact solution to our problem by either solving the equations directly, if the problem is simple enough, or by the well known method of manufactured solutions. In this case, we assume there exists a constant C independent of the mesh parameter h such that,

$$\|E\| = \|U_i - u_{exact}(ih)\| \leq Ch^\alpha \tag{1.3}$$

for an α ordered method. Then, we will compute U_k for a given mesh spacing h_k and compute the error corresponding to this level of refinement E_k. Next, we further refine the mesh by choosing a smaller mesh spacing say h_{k+1} for which we compute U_{k+1} and then the corresponding error E_{k+1} is computed. Combining two levels of refinement and (1.3) we obtain a simple formula for the observed rate of convergence. We have that

$$\alpha = \frac{\log(E_{k+1}/E_k)}{\log(h_{k+1}/h_k)}. \tag{1.4}$$

The goal of this work is to verify that we can recover rates of convergence in this way for static problems within the LAMMPS, Large-scale Atomic/Molecular Massively Parallel Simulator, code developed as an open source project at Sandia National Laboratories [3].

*Florida State University Department of Scientific Computing, charden@scs.fsu.edu
†Sandia National Laboratories, rblouh@sandia.gov

2. Minimization in LAMMPS. The LAMMPS code solves static problems in terms of energy minimization. Thus, to solve a static problem in the LAMMPS code we use the minimization routine to find the minimum energy configuration for the system. LAMMPS solves the following problem

$$\min\Phi(X_1, X_2, \ldots, X_N), \tag{2.1}$$

where N is the number of particles used in the system and Φ is a functional representing the potential energy of the system.

The LAMMPS code has two method options for solving the minimization problems. The user has the option of either using a conjugate gradient method, which is the default method, or one can specify a steepest descent method. While the conjugate gradient method tends to converge faster the steepest descent method will converge to the same minimum.

3. Choosing The Model. To begin we start with a simple linear spring system.

Consider the Cauchy equation of static equilibrium for elastostatics [1],

$$\mathrm{div}\sigma + b = 0 \tag{3.1}$$

where for our problem we have no external body force and thus take $b = 0$ and σ is the Cauchy stress tensor. In one dimension we have that,

$$\sigma = -k\frac{du}{dx} \tag{3.2}$$

where k is the elastic modulus and u is the displacement field for the material. Thus, plugging (3.2) into (3.1) we have,

$$\mathrm{div}\sigma = \mathrm{div}(-k\frac{du}{dx}) = -k\frac{d^2u}{dx^2} = 0. \tag{3.3}$$

We now have a continuum equation for the displacement of the particles in our linear spring system. We are interested in the static steady state solution, so to solve this equation we impose Dirichlet boundary conditions. In the simulation we hold the left boundary at $x = 0$ to be zero and at the other end, $x = L$, we fix the particle there to have a constant displacement of 1. Thus, we solve the following two point boundary value problem,

$$\begin{aligned} -k\frac{d^2u}{dx^2} &= 0 \\ u(0) = 0, \quad u(L) &= 1.0. \end{aligned} \tag{3.4}$$

We easily solve this equation, and apply the boundary conditions, for our continuous displacement field u, which gives us,

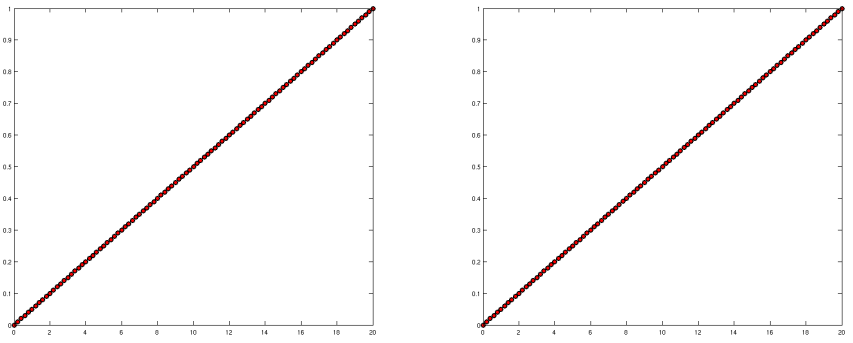
$$u(x) = \frac{1}{L}x, \tag{3.5}$$

which is a linear expression.

Unfortunately this problem is too simple for a meaningful convergence study. Recall from (1.2) that the truncation error of a second centered difference approximation is bounded by the norm of the fourth derivative of the solution. For this problem the solution has a zero fourth derivative. The issue here is that regardless of how much the mesh is refined the problem will be solved exactly to within machine precision.

What is observed in practice is that for small systems the error is very near to machine precision. As the mesh is refined, the system gets larger involving more floating point operations. Since the discretization error is already zero to within machine precision then when the system becomes larger the effects of the round off associated with the growing number of floating point operations begins to effect the total error which will begin to grow slightly as the mesh is further refined. This will make any convergence study meaningless.

In fig 3.1(a) and fig 3.1(b) one can observe that the LAMMPS and the Matlab results were well approximated for even a small number of particles.



(a) LAMMPS Result (b) Matlab Result

FIG. 3.1. *Comparative Result for LAMMPS and Matlab w/ 100 Particles*

In order to be able to recover convergence rates we need a model with a solution that has a nonzero fourth derivative. To achieve this we go back to equation (3.1) and choose a nonzero b that corresponds to adding a loading force to our original system. Here we choose $b = \sin(\frac{\pi x}{L})$. Using (3.2), our choice for b, and homogeneous Dirichlet boundary conditions then (3.4) becomes,

$$\begin{aligned} -k\frac{d^2u}{dx^2} &= \alpha \sin(\frac{\pi x}{L}) \\ u(0) = 0, \quad u(L) &= 0. \end{aligned} \tag{3.6}$$

The solution is easily obtained by an application of a variation of parameters technique that results in,

$$u(x) = \alpha(\frac{\pi}{L})^2 \frac{1}{k} \sin(\frac{\pi x}{L}) \tag{3.7}$$

where L is the length of our simulation domain and α is just a scaling parameter to keep the displacements from being too large relative to our simulation domain. In our simulations we take $L = 20$ and $\alpha = \frac{1}{40}$.

We now have a single equation by which we can compare our simulations regardless of the number of particles we are using in a particular simulation. The goal now is to test how well this continuum equation is represented as we add more and more particles into our simulation domain.

3.1. The Problem LAMMPS is Solving. The way that the LAMMPS solves a static problem is to perform a minimization of the potential energy of the system so that the system realizes its minimum energy configuration. For the atom and bond style being used the energy functional for our system is,

$$\Phi = \frac{1}{2}\sum_i \bar{k}(x_i - x_{i-1} - r_0)^2 + \sum_i x_i b_i, \tag{3.8}$$

where $\bar{k} = \frac{k}{\Delta x^2}$ is the spring constant, k is a parameter measured in force times length, $i = 2 \ldots N$, where N is the number of particles in the simulation, x_i, x_{i-1} are positions in the current configuration and r_0 is the lattice spacing. Looking ahead a bit what we would like is to show that this minimization problem is equivalent to solving the discretized continuum equation. In order to see this more easily we would like to be able to write our potential, and thus the minimization problem, in terms of the displacements u_i which is the quantity of interest in our continuum formulation. This can be done easily by noting that,

$$r_0 = y_i - y_{i-1}, \tag{3.9}$$

where y_i, y_{i-1} are positions in the relaxed reference configuration. This gives,

$$x_i - x_{i-1} - r_0 = (x_i - y_i) - (x_{i-1} - y_{i-1}) = u_i - u_{i-1}. \tag{3.10}$$

Thus the first term of the potential can be written in terms of the displacements of our particles,

$$\Phi = \frac{1}{2}\sum_i \bar{k}(u_i - u_{i-1})^2, \tag{3.11}$$

where $\bar{k} = \frac{k}{\Delta x^2}$ is the spring constant, $i = 2 \ldots N$, where N = the number of particles in the simulation, and u is the displacement subject to the constraint defined in (3.4).

The second term that has been added due to the external loading force also has an equivalent displacement formulation. We know that

$$u_i = x_i - y_i \tag{3.12}$$

where y_i is the reference position. Thus the added term becomes

$$\sum_i u_i b_i = \sum_i x_i b_i - \sum_i y_i b_i = \sum_i x_i b_i + C \tag{3.13}$$

where C is a constant. This is an equivalent formulation since energy is defined only up to a constant. Further, in minimization one is solving the gradient of the energy functional equal

to zero and the gradient of a constant is zero then one obtains exactly the same optimality system to solve during the minimization.

The potential energy functional can thus be written equivalently in terms of the positions or the displacements of the particles in the system.

$$\Phi = \frac{1}{2}\sum_i \bar{k}(u_i - u_{i-1})^2 - \sum_i u_i b_i, \tag{3.14}$$

where $\bar{k} = \frac{k}{\Delta x^2}$ is the spring constant, k is a parameter measured in force times length, $i = 2 \ldots N$, where N is the number of atoms in the simulation, u is the displacement subject to the constraint defined in (3.6), and $b_i = \alpha \sin(\pi i \Delta x / L)$.

We can write Φ in terms of a quadratic form if we introduce the $N \times N$ matrix $\mathbf{K}$ where,

$$\mathbf{K} = \bar{k}\begin{pmatrix} 1 & -1 & 0 & 0 & \ldots & 0 \\ -1 & 2 & -1 & 0 & \ldots & \vdots \\ 0 & -1 & 2 & -1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & -1 & 2 & -1 \\ 0 & \ldots & 0 & 0 & -1 & 1 \end{pmatrix}$$

and we define the vector of displacements,

$$\mathbf{u} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{pmatrix}$$

We may express the potential energy in terms of a quadratic form as follows,

$$\Phi = \frac{1}{2}\mathbf{u}^T\mathbf{K}\mathbf{u} - \mathbf{u}^T\mathbf{b}, \tag{3.15}$$

where $\mathbf{b}$ is given by,

$$\mathbf{b} = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_N \end{pmatrix}$$

where, again, $b_i = \alpha \sin(\pi i \Delta x / L)$.

LAMMPS then is essentially solving the problem of finding u such that $\nabla_u \Phi = 0$. Thus our optimality system is,

$$\nabla_u \Phi = \mathbf{K}\mathbf{u} - \mathbf{b} = 0. \tag{3.16}$$

If we apply Dirichlet boundary conditions to our 1D system, then we reduce the number of unknowns by two and obtain the following reduced dimension, $(N-2)\times(N-2)$ system to solve,

$$\hat{\mathbf{K}}\bar{\mathbf{u}} = \bar{\mathbf{b}} \tag{3.17}$$

where the matrix $\hat{\mathbf{K}}$ is given by,

$$\hat{\mathbf{K}} = \bar{k}\begin{pmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & -1 & 2 & -1 \\ 0 & \dots & 0 & -1 & 2 \end{pmatrix}$$

the vector $\bar{\mathbf{u}}$ is given by,

$$\bar{\mathbf{u}} = \begin{pmatrix} u_2 \\ u_3 \\ \vdots \\ u_{N-1} \end{pmatrix}$$

If we apply the same Dirichlet conditions as in (3.6), then our right-hand side vector $\bar{\mathbf{b}}$ becomes,

$$\bar{\mathbf{b}} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{N-1} \end{pmatrix}$$

It is easy to see that the system of equations given in (3.17) is exactly what one obtains from discretizing (3.6) with a second centered difference in space. Thus, solving the problem of minimizing (3.14) is equivalent to solving the discretized differential equation in (3.6).

Also, it now becomes clearer why we needed to choose $\bar{k} = \frac{k}{\Delta x^2}$ for our spring constant. Using this choice for the spring constant allows us to directly equate the minimization problem in LAMMPS with that of solving the discretized differential equation of interest.

4. Numerical Experiments. Now that we have an analytic solution to be able to compare our computational results with, we can use our norm calculations that we have implemented to try and recover the convergence rates of the minimization routine implemented within the LAMMPS code. For the present work we just look at the behavior of the computational results in the L_∞ norm that is defined,

$$\|E\|_{L_\infty} = \max_i |u(x_i) - U_i| \tag{4.1}$$

where, U_i is the approximate simulated solution at the point x_i.

To compute the rate of convergence we mimic what is done in the case of mesh based simulations where we take the lattice spacing to be the mesh constant. If we let h_i be the mesh

constant, assuming a uniform mesh here to start, associated with the i^{th} level of refinement and E_i be the residual associated with the same level of refinement then the rate of convergence, call it R_α, is computed according to the formula

$$R_\alpha = \frac{\log(E_{i+1}/E_i)}{\log(h_{i+1}/h_i)}. \tag{4.2}$$

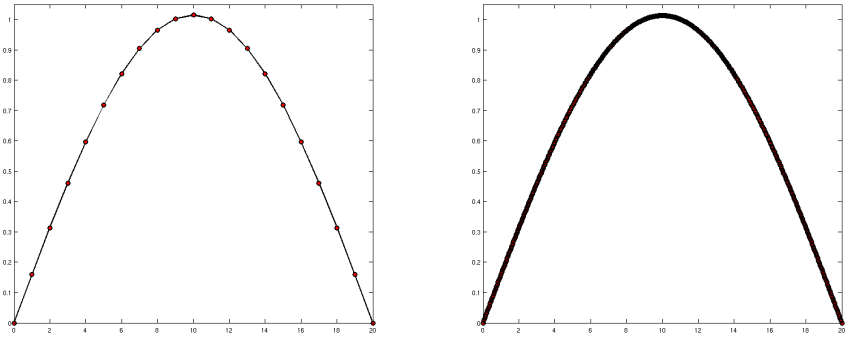
This is the standard approach for mesh based simulations on a uniform mesh.

4.1. Matlab. We first look at the convergence of our system (3.6) in Matlab. The tabulated convergence results for the L_∞ norm are as follows:

TABLE 4.1
Convergence Rates for the Matlab Simulation

Refine Lvl	Natoms	L_∞	R_α
1	21	0.0020859	-
2	51	0.000333074	2.002
3	101	$8.033336e^{-5}$	1.99887
4	201	$2.08336e^{-5}$	1.999986
5	401	$5.21101e^{-6}$	1.999277

As can be seen here we do indeed recover the second order rate of convergence as expected from our analytic results concerning the second centered difference formulation.

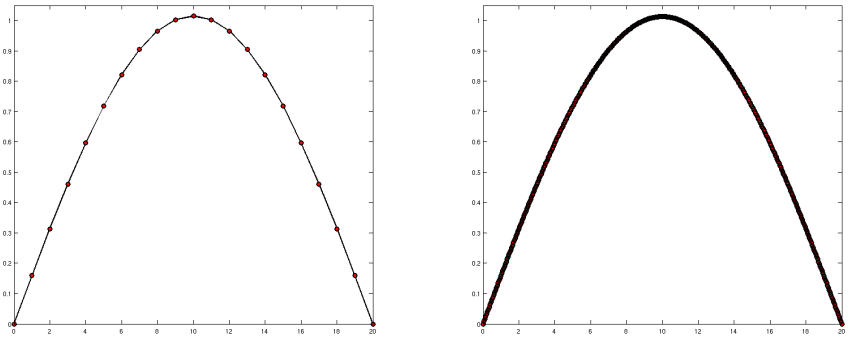


(a) 21 Particles (b) 400 Particles

FIG. 4.1. *Matlab Results*

4.2. LAMMPS. Next, we now take a look at the results from our simulations in LAMMPS. The tabulated convergence results for the L_∞ norm are as follows:

Here we see that we were indeed able to recover the appropriate rate of convergence for this problem in the LAMMPS code. One may observe that the convergence rate does begin to drift from its correct value when the number of particles becomes larger. In LAMMPS the position of the particles only converges to within single precision. It is believed then that as the error in the solution gets close to single precision then the roundoff in the simulation begins to pollute the error calculation thus affecting the measured convergence rate.



(a) 21 Particles (b) 400 Particles

FIG. 4.2. *LAMMPS Results*

5. Conclusions. We have shown that it is possible to use the minimization routines in the LAMMPS code to solve static problems. Further, we have shown that it is possible to recover the solution of continuum problems within this molecular dynamics particle based code. By seeking an equivalent formulation between both the static continuum problem and a particle based problem, posed as a minimization problem, we are able to recover the convergence rate predicted by the theory behind the discretization of the continuum problem.

We have successfully verified the spatial convergence of our mesh-free particle simulations by finding an equivalent mesh based formulation of the problem. In practice this may not always be possible so that techniques for direct convergence verification in particle based simulations are still desired. However, this approach can be useful for obtaining verification results of at least some of the most basic options within particle, lattice based simulation codes.

In LAMMPS the position of the particles only converge to within single precision. This means that when the error in the computed solutions begins to reach the limit of single precision then one can no longer recover the proper convergence rate. For these problems once the lattice was refined to contain more than 400 particles then the error in the solution was proportionately on the order of 10^{-7}, and thus we were not able to recover the correct rate of convergence for simulations larger than this. This does not mean, however, that the quality of larger simulations was degraded. As can be seen in Figure 5.1, for a simulation done with 2000 particles, the LAMMPS solution is nearly indistinguishable from the Matlab solution that can be verified for this large of a number of particles. This is not a verification argument but rather an example of code to code comparison between Matlab and LAMMPS. This

TABLE 4.2
Convergence Rates for the LAMMPS Simulation

Refine Lvl	Natoms	L_∞	R_α
1	21	0.00208563	-
2	51	0.000333074	2.002
3	101	$8.05099e^{-5}$	2.0486
4	201	$2.13959e^{-5}$	1.91183
5	401	$5.70439e^{-6}$	1.907

merely illustrates that the results are comparable for this larger problem set.

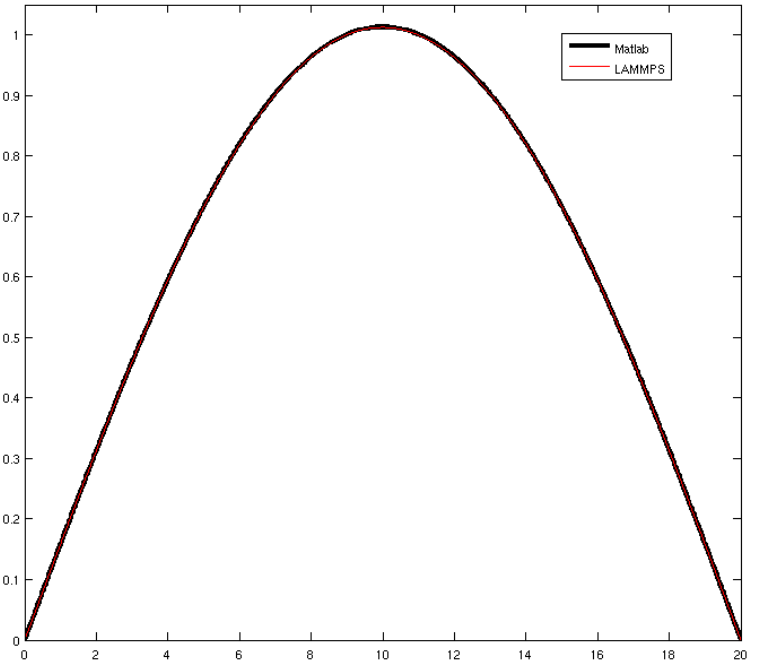


FIG. 5.1. *The LAMMPS and Matlab Solutions for 2000 particles*

6. Notes on Implementation. For this set of simulations we are using the *atom_style bond* and the *bond_style harmonic* to set up linear harmonic springs. For these styles the initial atomic positions and bond relations must be specified in a data file that is read in by LAMMPS at runtime. When testing that the routine was working correctly we first wrote up a small sample data file for testing. In order to do reasonably sized simulations we have written a C++ code, *genData.cc* to automatically generate the data file containing the atom and bond structures which allows us to specify an arbitrarily large number of atoms for a given simulation.

Also, we have written a LAMMPS fix, $fix\_add\_load.cpp$, to add the external loading force to the system and to appropriately adjust the potential energy of the system. The contribution from the loading force needs to be added to the potential due to the fact that this is the objective function to be minimized by the energy minimization routine.

A key point is the realization of needing to adjust the spring constant for each value of the lattice constant used. This realization is what allows us to directly link the particle and the continuum formulations of our problem. Without the link between these formulations, we would not be able to do the convergence verification presented in this work. Also, the LAMMPS code does not put the $\frac{1}{2}$ factor in front of $\bar{k}$ as shown in (3.8), so it is up to the user to factor this into the value of $\bar{k}$ that they input to the code. This is particularly important to keep in mind if one is using a fix to add values to the forces and particularly the potential. If the factor is not added, then there will be a disparity in the scaling between the terms in the

objective function to be minimized.

REFERENCES

[1] G. A. Holzapfel, *Nonlinear solid mechanics, a continuum approach for engineering*, (2000).
[2] P. M. Knupp and K. Salari, *Verification of computer codes in computational science and engineering*, (2002).
[3] S. J. Plimpton, *Fast parallel algorithms for short range molecular dynamics*, J. Comp. Phys., 117 (1995), pp. 1–19. Available at http://lammps.sandia.gov.
[4] P. J. Roache, *Verification and validation in computational science and engineering*, (1998).

BUILDING A SUSTAINABLE SIMULATION TESTING ENVIRONMENT

THOMAS L. AMES*, ALLEN C. ROBINSON†, RICHARD R. DRAKE‡, JOHN H. NIEDERHAUS§, V. GREGORY WEIRS ¶, AND D. A. LABRECHE ‖

Abstract. As the world of computational science and engineering matures and applications become more complicated and multidisciplinary, scientific and engineering code development practices have come under increasingly heavy criticism as predictive results are needed in the midst of a dynamic algorithmic and computing environment. We believe the only solution to obtain the required results in this environment is to elevate testing ethics and practices to a very high level. Testing is an essential part of code development that is used to ensure the correctness, robustness, and speed of a particular code. Unfortunately, code testing procedures and processes are often inadequate and can leave leave developers wasting hours of what could be productive time hunting for elusive errors. We give evidence that the modern software engineering principles of Lean Software Development, when used in conjunction with methods for code verification and validation, can be used to help mitigate this problem by building a team ethic along with a corresponding sustainable simulation testing environment.

1. Introduction. The engineering and process issues associated with computational predictive simulation have been amply discussed in the literature [7, 5]. Essentially, as simulations have become more complicated and more multidisciplinary with the potential to become truly predictive, the organizational issues associated with ensuring correctness of the modeling for a given purpose have become more and more important. The solution is not so much in the science itself but in the human organizational issues associated with dealing with complexity. One of the requests from the community writing about these difficult issues is that data be provided from individual code projects on what they consider to be the key issues to be solved. This paper is one attempt to provide this feedback.

We wish to make a contribution from the point of view that predictive simulation codes are, in a sense, always living entities and are never "done." In computational science, it is essential for codes to be correct, robust, and fast relative to the best available understanding at any given time. The only way we know how to do this is to ensure that developers pass their codes through a series of tests to confirm that their codes perform as desired. Not only should this be done before releasing a code to the customers, but it should also be carried out as a tool to aid developers in writing better codes more efficiently. That is, the development environment should facilitate completely replacing one technology with a better one.

Unfortunately, code testing procedures and processes as well as interactions between suppliers and users of current scientific code development projects can lead to very incomplete and counterproductive environments. This problem can be especially acute with complex codes that have been developed over many years with multiple authors. Codes must attempt to stay up with a changing algorithmic world lest they suffer the potential risk of stagnation and irrelevance, but the environment may not be conducive to achieving this.

Much work has gone into theories of verification and validation of codes to ensure code correctness and appropriateness for a given application [5, 8, 4]. Testing of any sort is labor intensive. Lamentably, our historical observation is that verification and/or validation efforts quickly become out-of-date after the initial studies are performed. Any specific effort to verify or validate a code which occurs outside of the permanent development process is of only limited lasting utility in a dynamic research and development environment. In general,

*Brigham Young University, tames@byu.net
†Sandia National Laboratories, acrobin@sandia.gov
‡Sandia National Laboratories, rrdrake@sandia.gov
§Sandia National Laboratories, jhniede@sandia.gov
¶Sandia National Laboratories, vgweirs@sandia.gov
‖Sandia National Laboratories, dalabre@sandia.gov

what is most lacking is an environment that allows and encourages a state-of-the-art product to be continuously maintained in ever changing circumstances.

We believe that the solution to these issues is found in the realm of team processes. It is actually in the human interaction and team management areas that success or failure is found.

Thus, it is important to have a system to manage the environment. In order for a development team to be able to commit to excellence, it needs the necessary tools to actually carry this commitment forward. We have found that the modern Lean Software Development ideas developed in the larger software development community map very well to scientific research and development environments. We believe this is true because these principles are not strongly dependent on the product being developed. Poppendieck has listed the following as key Lean principles: eliminate waste, build quality in, create knowledge, defer commitment, deliver fast, respect people and optimize the whole [6]. While we resonate with all of these ideas, we believe that the practices associated with building quality into software are key.

One of the key principles of Lean is to build quality into a product by having a "stop-the-line" mentality of development. In other words, when something is wrong (e.g. the code fails to pass the test criteria) the highest development priority should be to fix the testing problem. Additionally, tests are continuously improved to ensure that whatever problems have occurred can never occur again. This takes great dedication on the part of the developers but can greatly increase productivity because time is spent ensuring that the code is always correct as built rather than assuming that it can be fixed later. "Research" and "development" should not have rigid separations if cutting edge developments are to have immediate impact. Instead, we find that an integrated research, build, test, and delivery environment that integrates suppliers with the ultimate users is the best method to assure product utility.

In order for "stop-the-line" to be a beneficial tool for software development there must be a clear concept of what the "line" is and some automated way to recognize failures. A continually evolving test suite in a robust and convenient environment is required. In order to be sustainable, the testing environment must be easy to use as to be executed frequently to make sure that the code is still sufficient. In addition, no program can guarantee success without a complete commitment from all team members to the testing activities over the long term. Accordingly, we will discuss the processes, testing requirements and infrastructure which we believe necessary to maintain a highly complex simulation code.

2. Process and Testing Requirements. The first key to an effective software development environment is a team leader who is willing to set the standard for excellence. This team leader in the scientific software development environment is the equivalent of the "shusa" in Japanese automobile product development and production. This team leader must be knowledgeable and be willing to set and support high standards permitting the highest quality work at every stage of development. This team lead must be relentless in ensuring that the whole team buys into the pursuit of excellence. We think a somewhat independent testing team is useful, but, if their work is separated from the core build and test environment of the software product, this independent work is of only temporary help and will eventually be lost and wasted. All testing efforts must occur hand in hand with the software developers and integrated into a permanent testing environment.

We find that a key idea that is useful in building a quality product is "stop-the-line." This term is from quality manufacturing theory but is of critical importance in scientific software development. In software terms this means that all tests must have an automated pass-fail criteria, and all the tests in the suites must be continuously tested to ensure that they stay at the passing level. Any action which results in a failure to pass is understood to be of the highest priority to fix.

In order to understand the importance of a "stop-the-line" mentality of code development, it is important to understand the basic principle behind this methodology. More technically called autonomation, "stop-the-line" is part of the Toyota Production system developed by Taiichi Ohno that made Toyota Motor Corporation so successful. Ohno based his ideas on his knowledge of Toyoda automated textile looms that could operate without weavers present. They could do this because looms would detect even the slightest abnormality and shut down immediately until the cause of the problem was remedied.

Ohno related this system to the human nervous system that governs reflexes such as breathing, heartbeat, etc. If someone touches a hot stove, for instance, his autonomic nerves cause him to recoil without waiting for a message from the brain. It does not matter what he is doing. When his body detects that something is wrong, it immediately stops all tasks at hand and corrects the problem before work can be resumed [6].

Now, the obvious question is how this helps increase efficiency in computer programming. "Stop-the-line" is used to satisfy the Lean principle of building quality into a product. Poppendieck asserts that the job of tests is to prevent defects, not to find them. "According to Shigeo Shingo, there are two kinds of inspection: inspection after defects occur and inspection to prevent defects. If you really want quality, you don't inspect after the fact, you control conditions so as not to allow defects in the first place." Therefore, tests must be implemented during the development process not just when the product is ready to be released.

When tests start failing, it is vital stop additional work and fix errors before continuing. By so doing, errors will not be able to exist and hide in a code until a user complains that the code is not working properly. Of course, bugs can just be put in a prioritized list of errors to be fixed before the next release of the code, but trackers such as these are really just lists of unfinished work that has to be redone. Why would a developer start on a new task if the first one was never completed? The error has to be found regardless. Often, it is more difficult to find an error if not done immediately because it becomes harder to remember exactly what was done in a particular piece of code as time passes.

This being said, some may claim that this practices requires an unreasonable amount of team dedication or that it is too costly in terms of human time and computing resources. In response to the first argument we observe that in many fields such as athletics, music, government, or even warfare, greatness in a group setting is defined by devotion to the team. While the transition may be difficult at first, we quickly adapt to new environments if we immerse ourselves in them. With respect to the second argument, many forget that optimization of the whole is not necessarily the sum of the optimizations of the parts. In some cases it may be necessary to be very methodical at an early stage of development to save more time later. In other words, we eliminate can waste later by building quality and testing methodologies in now. The recent criticism of scientific computing practices in general have arisen largely due to the lack of quality in many large scale software simulation products. To respond to this it is clear that spending the time required to build and maintain a pervasive testing environment must be seen as an essential aspect of current best practice.

Test-driven development practices minimize "stop-the-line" events. With these practices, tests are developed early to assure that new coding is up to par before it is even included in the whole of a product. This does not eliminate indispensability of a "stop-the-line" environment though. Cutting-edge research is not static, and thus it is important to keep testing the code even though it passed tests as it was developed because it will inevitably change multiple times during the lifetime of a particular code base. In essence, we believe that test-driven development is intrinsically part of the "stop-the-line" methodology. Poppendieck reminds us that it has been found that development teams that develop unit tests early in the development process AND "stop-the-line" when errors occur have incredibly low defect rates and very

short time frames to find the cause of defects that are discovered [6].

Moreover, it is pointless to have a "stop-the-line" system implemented if the tests are not evaluating useful criteria. Roache is adamant that it is important to pay attention to the semantics when talking about verification and validation in order to ensure that all elements of the code are properly and completely tested [8]. For instance, by definition, a regression test is used only to show that the code has not changed significantly by testing it against a baseline test. It ensures that the code maintains status quo and helps identify which modification may have caused discrepancies when they arise, but it goes no further to show that the code is actually behaving as it should. Verification, on the other hand, aims toward testing for correctness of the mathematics of the code instead of just showing that it has not changed.

No single type of test problem can cover all of the aspects of a numerical simulation code because there are various types of problems that can occur. In order to have a truly quality code, it is important to test for problems in the computer science, the accuracy of the mathematics, the accuracy of the science and/or engineering approach, the speed and scaling performance of the code, and the code's ability to solve a "real" problem. When all of these criteria are met, the user and developer have a better evidence base to claim that the code is adequate for a given purpose. For us there are five useful classifications of test problems. They are primarily distinguished by the type of "metrics" that are utilized as part of the test. We define them as follows:

- Regression – In our terminology, regression tests demonstrate the adequacy of the computer science. These tests answer the question of whether or not any new version of the code gives the same answer as the previous version or alternative versions running on different numbers of processors or on different machines. These tests support refactoring, restart testing, parallel-serial consistency checks, and cross platform compatibility. Most of these tests can be made very fast. Useful software tools support checking the results of one computation against another to see if they are getting the same results to roundoff. If they are the same, the test passes, and otherwise, it fails. This is the regression metric.

  The principal purpose of regression tests is to provide a baseline against which the code can be changed when the developer only wants to change the code and not the results of the code. This is a critical point. Regression tests by themselves eventually produce a type of sluggishness or "anchor" in the long term scientific code development process because they fail by definition when next generation algorithms are implemented. Progress then slows down because tests must be reviewed, and a new baseline must be established for each one. To get around this fact, it is important to emphasize other types of tests.
- Verification – Verification tests ensure consistency of the implementation with the related mathematical problem. In many cases, the mathematical problem is posed in a continuous system but approximately solved in a discrete system. To provide a verification test metric, some independent mathematics and/or code is required. This can be an analytic solution, a symmetry condition, a null test, or a manufactured solution. Order convergence tests are used to assure users and developers that the code is converging to the right answer at the expected rate. In the end, one is always comparing one computation against another, but one makes the assumption that these two different computations utilizing entirely different methodologies will likely only agree if both are correct.

  Verification tests are the principal weapon the code team has to combat the code development sluggishness introduced by pure computer science based regression tests. A good verification suite will provide mechanisms for reducing option bloat.

As new and better algorithms are developed it is important to remove older, less effective algorithms. This is part of the "eliminate waste" aspect of Lean Software Development. A way to easily eliminate older algorithms with high confidence is to be able to run an existing verification suite which measures the distance of the current solution from the expected solution. If the code passes the criteria with a metric which is at least as good if not better as the old method, then the code can be upgraded to the new and improved algorithm. Confidence can then be built to allow complete removal of the older algorithm from the code base in order to reduce the complexity of the code base that must be maintained. Unit tests can be verification tests if they check small pieces of the code with known mathematical results.

- Validation – Validation tests compare the simulations code's results to experimental data in order to show how well it represents the science and engineering of interest. In essence, validation problems show that we are using the right equations for what we are trying to model. In this case, one needs experimental results with error bars. Changing code algorithms for the better should not move the computational results outside the experimental errors bars unless the physical model itself was in error. If they do, there is something wrong either in the equations that are posed or in the coefficients of those equations. With the proper equations and a verified code, the essence of a given science and engineering problem reduces to ensuring that the equations and their associated closure models are adequate for the actual physical system of interest.
- Performance – Performance tests monitor the speed or the parallel scaling of the code. It is important that these tests are run relative to well understood baselines on specific machines in order to to achieve meaningful results. The broader the set of performance metrics implemented in the test suite, the more likely that the team can immediately catch any performance loss as the code is developed.
- Prototype – Prototype tests demonstrate that the code continues to run sample "real" problems. These are key calculations that we always want to run to completion. The only metric we have is that these computations should successfully complete with reasonable results. This category is mainly an attempt to measure the robustness state of the code and to avoid regression in large scale robustness. We should always be able to complete these potentially long running problems. In many cases, these computations should migrate to validation problems whenever possible.

The five types of problems mentioned above are the categories we have found so far to be useful as a testing taxonomy to ensure that the code base is consistently improving. In order to make sure that all pertinent tests are run, it is important to make one testing environment for all developers and testers. Some of the basic requirements to make a sustainable testing environment are:

- Simplicity – A user should only have to type one command line prompt to run all necessary tests, and results should be easily interpreted with a pass-fail criteria. In addition, it should be easy for a developer or tester to use the infrastructure for making a test. The environment will be more sustainable if it is simple because all involved will be more likely to use it to make their lives easier.
- Generality – The infrastructure should allow for different tools to be used for the different types of problems and for new tools to be easily incorporated if needed.
- Capability – Different functionalities are necessary to setup and analyze different types of test problems.
- Adaptability – It may not be feasible to always run all of the test problems. A method for selecting a subset of problems applicable to the developer's needs should

be available.
- Automaticity – Running of the tests on several machines should be automatic and easy to accomplish for all developers as well as the members of the testing and delivery team.

We describe at this time the set of tools that are used to incorporate these principles into an infrastructure that is used use to test ALEGRA, a highly-complex multi-physics code. This infrastructure is described in the following section.

3. Testing Infrastructure. With the variety of test problems needing to be executed, building a sustainable testing environment requires multiple distinct tools. We are working with ALEGRA , a highly-complex multi-physics finite element code that couples magnetics, hydrodynamics, thermal conduction, and radiation transport. The code is very large and requires a correspondingly large amount of testing that needs to be maintained. We have combined tools that have been developed for the purposes of analyzing and testing ALEGRA into a more capable environment that combines required features. These tools include: `exodiff`, `compare_final`, `tampa`, and `testAlegra`; all of which have descriptions that follow.

3.1. `exodiff`. `exodiff` helps fulfill the infrastructure requirements of simplicity and capability. It takes the output from two ALEGRA simulations, called exodus files, and compares them. If they are different, `exodiff` alerts the user. In a sustainable environment, the necessary input files for an ALEGRA run are saved with along with a baseline exodus file produced by an older version of ALEGRA. When testing is required, the inputs are given to the current version of ALEGRA and `exodiff` is invoked to compare the results produced to the baseline file. A test passes if the results are the same within roundoff.

3.2. `compare_final`. `compare_final` is just as simple and adds capability to our testing environment. `compare_final` extracts history data from an exodus or a history file (a history file is another form of ALEGRA output), then grabs the values of various variables at the last time step and does a tolerance check on them. A lower tolerance is also available so that if a much "better" number is ever obtained, the test will fail. In this case the fix is simply to tighten the tolerance! It is used with single variables such as the time it took to run the simulation, and is useful as a metric tool for verification, validation, performance, and prototype problems.

One may argue that such a tool is too simple or that order convergence tests using the Method of Manufactured Solutions (MMS) is better (for more information on MMS type tests see [3]). We agree that order convergence tests are important however we also argue that it is better to have any good verification test implemented and running in the standardized suite than none at all. Thus having simple tools that every developer can easily use with almost no effort is important. Developing a verification mindset for the team is extremely important and simple tools can help facilitate this. We have found it useful to program simple exact or approximate results into the user input file. The input file is run through a preprocessor and the resulting output field which is fed to ALEGRA contains the numerical numbers that can be used in the `compare_final` test.

3.3. `tampa`. `tampa` is very useful in setting up test suites where multiple runs are needed with very similar input. For example, it may be necessary to run the exact same problem on multiple grid sizes to compare convergence rates (see sec. 4.1). Using a series of xml files, `tampa` builds a directory tree to organize the various runs and places the necessary inputs in each sub-directory. These inputs are generated by `tampa` which takes a generic input and fills in user-defined variables with values saved in a separate parameters section using a preprocessor. `tampa` then allows a user to run all of the tests that it set up and analyze them with user defined scripts.

`tampa` is invaluable for setting up and analyzing verification type problems even outside of a testing environment. Managing and modifying the myriad of inputs that might be necessary for a particular study is tedious to say the least, but `tampa` minimizes the user involvement and thus frustration. It is also very convenient that `tampa` has built in functionality to compare runs one to another because this allows the test developer to focus on the analysis of the test and not on directory searching. Thus, `tampa`'s principal value is the simplicity it adds the the test creation process and the automaticity that it adds to running a large number of tests. In addition, `tampa` has options for refining the set of problems to be run and analyzed that makes it adaptable.

3.4. `testAlegra`**.** In addition to these analysis tools, we have developed the driver `testAlegra` to run the test problems and utilize the tools that we have already discussed. It walks a directory tree looking for xml files with a certain tag. Stored in these xml files is a series of instructions that helps `testAlegra` setup, run, and analyze a problem. Then, from the analysis, it reports back as simple pass, fail, or diff ("difference") for each problem that it runs.

`testAlegra` is general enough that new tools can easily be incorporated into the test building or analyzing process as new tools are developed. Admittedly, we used tools that we had already developed with slight modifications as a starting place in our new testing methodology instead of building tools around our requirements, but as our processes evolve to more completely fulfill those requirements, `testAlegra` will easily accept the changes.

Furthermore, `testAlegra` is able to automatically run every problem in our testing environment with a single command line prompt. At the command line, keywords can be selected to adapt the tests run to the needs of the developer. Additionally, its simple output makes it easy for a user to interpret the results of the tests that they ran. One idea that is not currently present in `testAlegra` however is the need for a "footnoted" pass status. In our experience, it is common for a new verification problem to expose a weakness in the code. For example the convergence order may not be as high as one expects. This weakness may take some time to fix yet one still wants to run the verification test on a routine basis as part of the normal testing process to ensure that the results do not get worse. To this end we have proposed a "footnoted pass" status in which the outstanding issues can be documented until they are fixed.

3.5. Sustainability. The testing tool set means nothing if not backed up by a team ethic that considers any failure of the above system of binary tests as completely offensive. All testing failures are taken very seriously. If footed passes exist these should also be constantly considered for priority attention. In addition, those in charge of testing and distribution should have the authority and willingness to shut down any new feature commits to the software repository until testing issues with current committed code are resolved.

In a project that is as complex and with as extensive a history as ALEGRA, a significant testing infrastructure and coverage debt has accumulated relative to modern best practices. We believe this to be true for most scientific projects today that did not originate with a full blown modern testing methodology in mind. We see no recourse here but to start from where we are and to methodically push forward with the development of tests and testing infrastructure which are designed to benefit current customers. We believe all customers supporting current development are by default implicitly supporting whatever testing infrastructure and verification and validation work that is required to deliver a quality product. In addition, testing debt can be paid down and methodological improvements made by utilizing money set aside specifically for verification and validation activities. These activities must occur within the permanent code development testing environment if this money is to be well spent and have enduring value. The only way to sustain the environment is to consistently take time for

developing and maintaining test problems. As stated before, this takes great discipline on the part of the development team because this process is time consuming.

Some flexibility in testing methodology and policy will always be required. In the ALEGRA team, it is thought that it is not necessary to run every test every time that new code is committed. Currently, the policy that has been implemented is that all tests that run either very quickly or at least moderately fast are run every time a developer wants to commit new code, and all tests are run on a regular basis by a testing specialist (we used the keywords fast, medium, and long with fast and medium tests being run before committing). Another viewpoint that has been expressed is that it may be more efficient to sort the tests into categories such as commit, nightly, weekly, and release which categorizes the tests on a basis of how often a test needs to be run. When a policy is agreed upon, it is easy to facilitate necessary changes because of `testAlegra`'s and `tampa`'s keyword related operation.

Additionally, we find that certain computer science related practices are useful to help keep a code base that is portable and clean. In particular we utilize

- Compile warnings – We set our compilers to flag for coding which is considered to be bad programming style or potentially in error. We make policy that warnings messages from common compilers are not allowed. Keeping a clean warning status allows for easy flagging of potentially erroneous code directly from compiler. A clean warning status policy is another "stop-the-line" idea.
- Coverage testing – Various coverage tools are possible to measure the extent to which lines of code are covered by the test suite. This is used by the testing team to provide all developers with an up-to-date status of the code coverage.

4. Examples. In order so show examples of the different types of test problems and how our infrastructure handles them, a few examples are included.

4.1. Ideal MHD Shock Tube. A typical verification test is an order of convergence test over runs with progressively more refined grids. We implemented this procedure on an quasi-1D ideal magnetohydrodynamics (MHD) Riemann problem from a paper written by Dongsu Ryu and T. W. Jones [9]. It has an exact solution, which is preferable for verification problems, and we use the same exact Riemann solver that was used by Ryu and Jones. For more information on the equations of ideal MHD and the physical interpretation of their eigenmodes, see the works by Dai and Woodward, and Jeffrey [1, 2].

This problem is highly shock dominated, and we are interested in the location of the shock wave and the pre- and post-shock states. When the simulation is finished, the we study the L^1 error norms of the density, pressure, y-component of the velocity, and y-component of the magnetic field. Density is the standard variable that is studied for this problem so we start by showing that we can replicate what others in the field are doing. We study pressure because discontinuities in the solution typically have much larger magnitudes in the pressure than in the density (because the fluid compresses less and less easily as its entropy increases), and this allows us to test the thermodynamic coupling of different variables in the simulation. Further, we study the y-component of the velocity and magnetic field because some of the strongest errors that we noticed were in the transverse direction to the shock wave.

We use the L^1 error norm because shocks introduce strong discontinuities into the solution in this problem, and the L^1 norm, being the least dependent on solution smoothness, is the most likely to provide a reliable and meaningful measure of error. In problems with smooth solutions, the L^2 norm is more common.

We start by using `tampa` to set up multiple runs with different grid sizes. Since increasing grid resolution increases run time dramatically, it is important to take account that highly refined grids will take a long time to compute. Thus, we used `tampa`'s subset option to allow a user to select whether they want a fast, medium, or long test. In this particular problem,

the quasi-1D grids are $2 \times 2 \times N$. We define the fast subset as including grid resolutions with $N = 16, 32, 64, 128$. The medium subset includes the fast grids in addition to a $N = 256$ grid. The long subset includes all of the above and grids with $N = 512, 1024, 2048$.

In the analysis script the variables previously discussed are extracted, and their norms are taken. Using those norms, the effective rate of convergence is taken between the two grid resolutions. Minimum and maximum tolerances are set for the orders of convergence so that "diff" is reported if the results are outside of that range. It is important to have both a maximum and minimum tolerance in the tool so that a developer will know when the code is improving. The developer can then "raise the bar" on the code as it improves.

Earlier, we stated that the output needs to be easy to read, and to comply with that requirement, screen output at the end of the run simply reports whether each individual run passed (meaning that it ran until completion) or failed and whether the convergence study passed, failed, or diffed. Fail means that the analysis script did not run to completion, diff means that at least one of the norms of the variables did not fall within the tolerance range, and pass means that all criteria were met. In order to not clutter the screen with each variable's results, they are not individually displayed with a status, but if a user wants to know more, he or she can reference a log where all of the variables tested are listed with the two grids being compared along with the associated value of the convergence rate.

4.2. Solid Kinematics. The Method of Manufactured Solutions (MMS) is another important tool in verification testing because it allows the test developer to formulate tests that can cover features and capabilities of the code in explicit generality. Traditionally, exact solutions begin with the statement of a mathematical problem that are solved using the classical methods such as separation of variables, series solutions, integral methods, or Green's functions. MMS, on the other hand, starts by choosing a solution and applying PDE operators to it to find a source term that balances the equations. This methodology helps ensure that no testing gaps exist due to exact solutions that are too simple [3].

Using such an MMS approach, we were able to avoid the implementation of a flawed algorithm. We developed our test suite for the purpose of comparing different approaches for tracking the kinematics of high deformation solid motion. In particular, we were concerned with the order of convergence and performance. In so doing, we decided to implement a new time-stepping algorithm that has improved properties relative to the previous method.

Upon making the change, we ran the convergence suite and were quickly alerted that something was wrong. Inspection of a convergence rate test showed dramatically lower convergence rates than we had been obtaining with the previous time stepping algorithm. In many cases these convergence rates were effectively zero or even negative.

	Results Before Change		Results After Change	
1/h	L^1	Order	L^1	Order
16.	9.557E-02	–	1.243E-01	–
32.	2.038E-02	2.229	6.081E-02	1.032
64.	3.829E-03	2.412	4.810E-02	0.338
96.	1.467E-03	2.366	4.645E-02	0.086
128.	7.532E-04	2.318	4.597E-02	0.036

TABLE 4.1

Comparison of rotation tensor matrix convergence rates for this algorithm also indicated an error in the code. The left columns show convergence rates of the L^1 error norm before implimenting the new time-stepping algorithm. The right columns show the rates after the change.

The error was a subtle mistake in the implementation of the time integration algorithm.

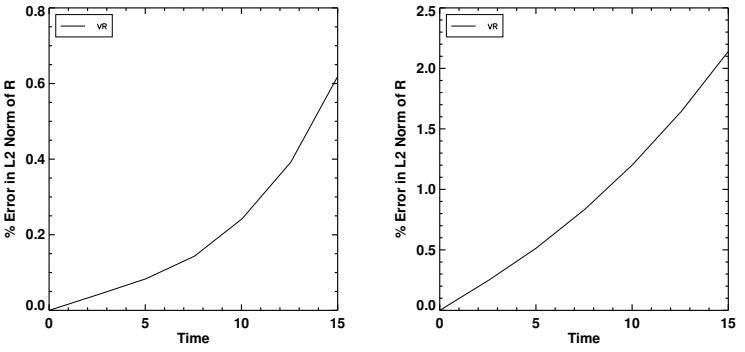


FIG. 4.1. *The left figure shows the original percent error versus time for a 2D test problem, while the figure on the right shows the percent error after the new time stepping algorithm was implemented.*

The algorithm in question was presented by Simo and Hughes [10]. They show that

$$\Lambda_{n+1} = exp[\Delta t\widehat{\omega}_{n+\alpha}]\Lambda_n; \qquad \alpha = \begin{cases} 0 & \Rightarrow \text{explicit (forward) Euler} \\ \frac{1}{2} & \Rightarrow \text{midpoint rule} \\ 1 & \Rightarrow \text{implicit (backward) Euler} \end{cases} \tag{4.1}$$

where

$$exp[\widehat{\omega}] = 1 + \frac{\sin(||\omega||)}{||\omega||}\widehat{\omega} + \tfrac{1}{2}\left[\frac{\sin(||\omega||/2)}{||\omega||/2}\right]^2 \widehat{\omega}^2, \tag{4.2}$$

and ω represents the mapping of the skew-symmetric matrix $\widehat{\omega}$ to a vector defined by the relationship

$$\widehat{\omega}h = \omega \times h; \qquad \text{for all } h \in \mathbb{R}^3. \tag{4.3}$$

The implementation mistake was in poorly distributing Δt so that

$$exp[\Delta t\widehat{\omega}] = 1 + \frac{\sin(||\omega||)}{||\omega||}\Delta t\widehat{\omega} + \tfrac{1}{2}\left[\frac{\sin(||\omega||/2)}{||\omega||/2}\right]^2 (\Delta t\widehat{\omega})^2 \tag{4.4}$$

instead of

$$exp[\Delta t\widehat{\omega}] = 1 + \frac{\sin(\Delta t||\omega||)}{||\omega||}\widehat{\omega} + \tfrac{1}{2}\left[\frac{\sin(\Delta t||\omega||/2)}{||\omega||/2}\right]^2 \widehat{\omega}^2. \tag{4.5}$$

Because sin is not a linear function, this yields a drastically different answer even though it is not visually obvious by code inspection. Without the immediate implementation of a verification test for this new feature, this bug could have gone unnoticed until much further down the road. After the fixing the error, the expected results are as shown in Figure 4.2. We are now in the process of ensuring that these tests are included in the permanent automated test suite. This experience demonstrates the importance of test-driven development.

5. Conclusion. Mistakes are unavoidable but their long reaching effects can be mitigated through the proper implementation of a sustainable testing process and environment.

	Results After Correction	
1/h	L^1	Order
16.	9.521E-02	–
32.	2.026E-02	2.232
64.	3.797E-03	2.416
96.	1.452E-03	2.370
128.	7.447E-04	2.322

TABLE 4.2
Rotation tensor matrix convergence rates after correction.

Most importantly, it is essential that the development team buys into and abides by an ethic that promotes a rigorous testing policy. This attitude is the key to the long term sustainability and health of the code project. The team leader must instill a "stop-the-line" attitude that insists that regression, verification, validation, performance, or prototype tests always indicate that the code is behaving as anticipated. Verification tests must be widely implemented to be able to effectively move algorithmic advances into the mainstream and leave older algorithm behind. It is essential to have an infrastructure that makes such policies natural, easy-to-abide and supportive of future development. The key requirements for this infrastructure are simplicity, generality, capability, adaptability, and automaticity.

Building and maintaining a team and software environment that makes these ideas feasible should be a high priority for all team members. Adopting these key attitudes and principles can be a difficult process, but the results brings with it a simulation tool that can be trusted by users yet still rapidly improved in a dynamic research and development environment.

REFERENCES

[1] W. DAI AND P. R. WOODWARD, *An approximate Riemann solver for ideal magnetohydrodynamics*, J. Comp. Phys., 111 (1994), pp. 354–372.
[2] A. JEFFREY, *Magnetohydrodynamics*, Oliver and Boyd, Edinburgh, 1966.
[3] P. KNUPP AND C. C. OBER, *A code-verification evidence-generation process model and checklist*, Sandia Report, (2008).
[4] P. KNUPP AND K. SALARI, *Verification of Computer Codes in Computational Science and Engineering*, 2003.
[5] W. L. OBERKAMPF AND T. G. TRUCANO, *Verification and validation benchmarks*, Nuclear Engineering Science, 238 (2008), pp. 716–743.
[6] M. POPPENDIECK AND T. POPPENDIECK, *Implementing Lean Software Development*, 2007.
[7] D. E. POST AND L. G. VOTTA, *Computational science demands a new paradigm*, Physics Today, (2005), pp. 42–142.
[8] P. J. ROACHE, *Verification and Validation in Computational Science and Engineering*, Hermosa Publishers, Albuquerque, 1998.
[9] D. RYU AND T. W. JONES, *Numerical magnetohydrodynamics in astrophysics: algorithm and tests for one-dimensional flow*, Ap. J., 442 (1995), pp. 228–258.
[10] J. C. SIMO AND T. J. R. HUGHES, *Computational Inelasticity*, Springer, New York, 1998.