http://dx.doi.org/10.1090/conm/218/3020

Contemporary Mathematics
Volume **218**, 1998
B 0-8218-0988-1-03020-X

On the Interaction of Architecture and Algorithm in the Domain-based Parallelization of an Unstructured-grid Incompressible Flow Code

Dinesh K. Kaushik, David E. Keyes, and Barry F. Smith

1. Introduction

The convergence rates and, therefore, the overall parallel efficiencies of additive Schwarz methods are often notoriously dependent on subdomain granularity. Except when effective coarse-grid operators and intergrid transfer operators are known, so that optimal multilevel preconditioners can be constructed, the number of iterations to convergence and the communication overhead per iteration tend to increase with granularity for elliptically-controlled problems, for either fixed or memory-scaled problem sizes.

In practical large-scale applications, however, the convergence rate degradation of fine-grained single-level additive Schwarz is sometimes not as serious as the scalar, linear elliptic theory would suggest. Its effects are mitigated by several factors, including pseudo-transient nonlinear continuation and dominant intercomponent coupling that can be captured exactly in a point-block ILU preconditioner. We illustrate these claims with encouraging scalabilities for a legacy unstructured-grid Euler flow application code, parallelized with the pseudo-transient Newton-Krylov-Schwarz algorithm using the PETSc library. We note some impacts on performance of the horizontal (distributed) and vertical (hierarchical) aspects of the memory system and consider architecturally motivated algorithmic variations for their amelioration.

2. Newton-Krylov-Schwarz

The discrete framework for an implicit PDE solution algorithm, with pseudo-timestepping to advance towards an assumed steady state, has the form: $(\frac{1}{\Delta t^\ell})\mathbf{u}^\ell + \mathbf{f}(\mathbf{u}^\ell) = (\frac{1}{\Delta t^\ell})\mathbf{u}^{\ell-1}$, where $\Delta t^\ell \to \infty$ as $\ell \to \infty$. Each member of the sequence of nonlinear problems, $\ell = 1, 2, \ldots$, is solved with an inexact Newton method. The

1991 *Mathematics Subject Classification.* Primary 65N55; Secondary 65Y05, 65N30.

Supported in part by NASA contract NAGI-1692 and by a GAANN fellowship from the U. S. Department of Education.

Supported in part by the National Science Foundation grant ECS-9527169 and by NASA Contracts NAS1-19480 and NAS1-97046.

Supported by U.S. Department of Energy, under Contract W-31-109-Eng-38.

resulting Jacobian systems for the Newton corrections are solved with a Krylov method, relying only on matrix-vector multiplications. The Krylov method needs to be preconditioned for acceptable inner iteration convergence rates, and the preconditioning is the "make-or-break" aspect of an implicit code. The other phases parallelize well already, being made up of vector updates, inner products, and sparse matrix-vector products.

The job of the preconditioner is to approximate the action of the Jacobian inverse in a way that does not make it the dominant consumer of memory or cycles in the overall algorithm. The true inverse of the Jacobian is usually dense, reflecting the global Green's function of the continuous linearized PDE operator it approximates. A good preconditioner saves time and space by permitting fewer iterations in the Krylov loop and smaller storage for the Krylov subspace. An additive Schwarz preconditioner [**4**] accomplishes this in a localized manner, with an approximate solve in each subdomain of a partitioning of the global PDE domain. Applying any preconditioner in an additive Schwarz manner tends to increases flop rates over the same preconditioner applied globally, since the smaller subdomain blocks maintain better cache residency. Combining a Schwarz preconditioner with a Krylov iteration method inside an inexact Newton method leads to a synergistic parallelizable nonlinear boundary value problem solver with a classical name: Newton-Krylov-Schwarz (NKS) [**5, 8**].

When nested within a pseudo-transient continuation scheme to globalize the Newton method [**11**], the implicit framework (called ΨNKS) has four levels:

```
do l = 1, n_time
   SELECT TIME-STEP
   do k = 1, n_Newton
      compute nonlinear residual and Jacobian
      do j = 1, n_Krylov
         do i = 1, n_Precon
            solve subdomain problems concurrently
         enddo
         perform Jacobian-vector product
         ENFORCE KRYLOV BASIS CONDITIONS
         update optimal coefficients
         CHECK LINEAR CONVERGENCE
      enddo
      perform vector update
      CHECK NONLINEAR CONVERGENCE
   enddo
enddo
```

The operations written in uppercase customarily involve global synchronizations.

We have experimented with a number of Schwarz preconditioners, with varying overlap and varying degrees of subdomain fill-in, including the new, communication-efficient, Restricted Additive Schwarz (RAS) method [**6**]. For the cases studied herein, we find the degenerate block Jacobi form with block ILU(0) on the subdomains is adequate for near scalable convergence rates.

3. Parallel Implementation Using PETSc

The parallelization paradigm we employ in approaching a legacy code is a compromise between the "compiler does all" and the "hand-coded by expert" approaches. We employ the "Portable, Extensible Toolkit for Scientific Computing" (PETSc) [**2, 3**], a library that attempts to handle through a uniform interface, in a highly efficient way, the low-level details of the distributed memory hierarchy. Examples of such details include striking the right balance between buffering messages and minimizing buffer copies, overlapping communication and computation, organizing node code for strong cache locality, preallocating memory in sizable chunks rather than incrementally, and separating tasks into one-time and every-time subtasks using the inspector/executor paradigm. The benefits to be gained from these and from other numerically neutral but architecturally sensitive techniques are so significant that it is efficient in both the programmer-time and execution-time senses to express them in general purpose code.

PETSc is a large and versatile package integrating distributed vectors, distributed matrices in several sparse storage formats, Krylov subspace methods, preconditioners, and Newton-like nonlinear methods with built-in trust region or line-search strategies and continuation for robustness. It has been designed to provide the numerical infrastructure for application codes involving the implicit numerical solution of PDEs, and it sits atop MPI for portability to most parallel machines. The PETSc library is written in C, but may be accessed from user codes written in C, FORTRAN, and C++. PETSc version 2, first released in June 1995, has been downloaded thousands of times by users worldwide. PETSc has features relevant to computational fluid dynamicists, including matrix-free Krylov methods, blocked forms of parallel preconditioners, and various types of time-stepping.

A diagram of the calling tree of a typical ΨNKS application appears below. The arrows represent calls that cross the boundary between application-specific code and PETSc library code; all internal details of both are suppressed. The top-level user routine performs I/O related to initialization, restart, and post-processing and calls PETSc subroutines to create data structures for vectors and matrices and to initiate the nonlinear solver. PETSc calls user routines for function evaluations $\mathbf{f}(\mathbf{u})$ and (approximate) Jacobian evaluations $\mathbf{f}'(\mathbf{u})$ at given vectors $\mathbf{u}$ representing the discrete state of the flow. Auxiliary information required for the evaluation of $\mathbf{f}$ and $\mathbf{f}'(\mathbf{u})$ that is not carried as part of $\mathbf{u}$ is communicated through PETSc via a user-defined "context" that encapsulates application-specific data. (Such information typically includes dimensioning data, grid data, physical parameters, and quantities that could be derived from the state $\mathbf{u}$, but are most conveniently stored instead of recalculated, such as constitutive quantities.)

4. Parallel Port of an NKS-based CFD Code

We consider parallel performance results for a NASA unstructured grid CFD code that is used to study the high-lift, low-speed behavior of aircraft in take-off and landing configurations. FUN3D [**1**] is a tetrahedral vertex-centered unstructured grid code developed by W. K. Anderson of the NASA Langley Research Center for compressible and incompressible Euler and Navier-Stokes equations. FUN3D uses a control volume discretization with variable-order Roe schemes for approximating the convective fluxes and a Galerkin discretization for the viscous terms. Our parallel experience with FUN3D is with the incompressible Euler subset thus far,

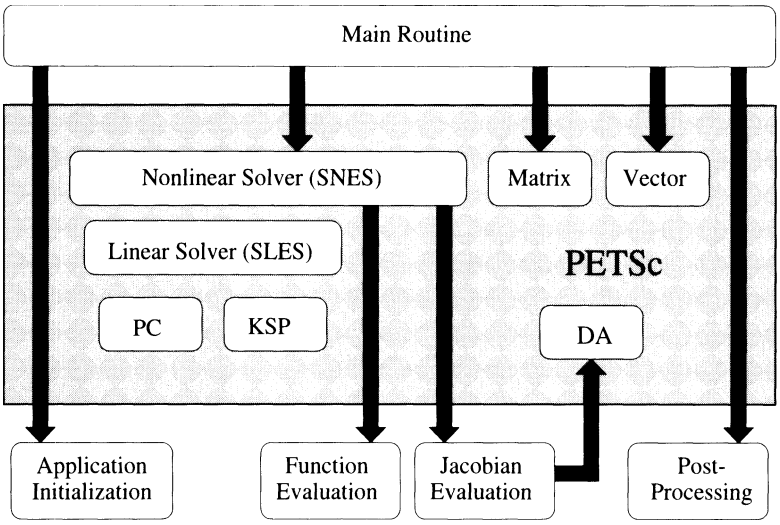

FIGURE 1. Coarsened calling tree of the FUN3D-PETSc code, showing the user-supplied main program and callback routines for providing the initial nonlinear iterate, computing the nonlinear residual vector at a PETSc-requested state, and evaluating the Jacobian (preconditioner) matrix.

but nothing in the solution algorithms or software changes for the other cases. Of course, convergence rate will vary with conditioning, as determined by Mach and Reynolds numbers and the correspondingly induced grid adaptivity. Furthermore, robustness becomes more of an issue in problems admitting shocks or making use of turbulence models. The lack of nonlinear robustness is a fact of life that is largely outside of the domain of parallel scalability. In fact, when nonlinear robustness is restored in the usual manner, through pseudo-transient continuation, the conditioning of the linear inner iterations is enhanced, and parallel scalability may be improved. In some sense, the Euler code, with its smaller number of flops per point per iteration and its aggressive trajectory towards the steady state limit may be a *more*, not less, severe test of scalability.

We employ ΨNKS with point-block ILU(0) on the subdomains. The original code possesses a pseudo-transient Newton-Krylov solver already. Our reformulation of the global point-block ILU(0) of the original FUN3D into the Schwarz framework of the PETSc version is the primary source of additional concurrency. The timestep grows from an initial CFL of 10 towards infinity according to the switched evolution/relaxation (SER) heuristic of Van Leer & Mulder [**12**]. In the present tests, the maximum CFL is 10^5. The solver operates in a matrix-free, split-discretization mode, whereby the Jacobian-vector operations required by the GMRES [**13**] Krylov method are approximated by finite-differenced Fréchet derivatives of the nonlinear residual vector. The action of the Jacobian is therefore always "fresh." However, the submatrices used to construct the point-block ILU(0) factors on the subdomains as part of the Schwarz preconditioning are based on a lower-order discretization than the one used in the residual vector, itself. This is a common approach in practical codes, and the requisite distinctions within the residual and Jacobian subroutine calling sequences are available in the legacy FUN3D version.

TABLE 1. Cray T3E parallel performance (357,900 vertices)

procs	its	time	speedup	Efficiency		
				η_{alg}	η_{impl}	$\eta_{overall}$
16	77	2587.95s	1.00	1.00	1.00	1.00
32	75	1262.01s	2.05	1.03	1.00	1.03
64	75	662.06s	3.91	1.03	0.95	0.98
128	82	382.30s	6.77	0.94	0.90	0.85

4.1. Parallel Scaling Results. We excerpt from a fuller report to appear elsewhere [**10**] tables for a 1.4-million degree-of-freedom (DOF) problem, converged with a relative steady-state residual reduction of 10^{-10} in approximately 6.5 minutes using approximately 1600 global fine-grid flux balance operations (or "work units" in the multigrid sense), on 128 processors of a T3E; and for an 11.0-million DOF problem, converged in approximately 30 minutes on 512 processors. Relative efficiencies in excess of 80% are obtained over relevant ranges of processor number in both cases. Similar results are presented in [**10**] for the IBM SP. The minimum relevant number of processors is (for our purposes) the smallest power of 2 that can house a problem completely in distributed DRAM. In practice, using fewer than this holds high performance resources captive to paging off of slow disks (and dramatically inflates subsequent parallel speedups!). The maximum relevant number is the maximum number available or the largest power of 2 that allows enough volumetric work per processor to cover the surfacial overhead. In practice, tying up more processors than this for long runs can be construed as wasting DRAM.

The physical configuration is a three-dimensional ONERA M6 wing up against a symmetry plane (see Fig. 2) an extensively studied standard case. Our tetrahedral Euler grids were generated by D. Mavriplis of ICASE. We use a maximum Krylov dimension of 20 vectors per pseudo-timestep. The pseudo-timestepping is a nontrivial feature of the algorithm, since the norm of the steady state residual does not decrease monotonically in the larger grid cases. (In production, we would employ mesh sequencing so that the largest grid case is initialized from the converged solution on a coarser grid. In the limit, such sequencing permits the finer grid simulation to be initialized within the domain of convergence of Newton's method.)

Table 1 shows a relative efficiency of 85% over the relevant range for a problem of $4 \times 357,900$ DOFs. Each iteration represents one pseudo-timestep, including one Newton correction, and up to 20 Schwarz-preconditioned GMRES steps. Convergence is defined as a relative reduction in the norm of the steady-state nonlinear residual of the conservation laws by a factor of 10^{-10}. The convergence rate typically degrades slightly as number of processors is increased, due to introduction of increased concurrency in the preconditioner, which is partition-dependent, in general.

The overall efficiency, $\eta_{overall}$, is the speedup divided by the processor ratio. The algorithmic efficiency, η_{alg}, is the ratio of iterations to convergence, as processor number varies. The implementation efficiency, η_{impl}, the quotient of $\eta_{overall}$ and η_{alg}, therefore represents the efficiency on a *per iteration* basis, isolated from the slight but still significant algorithmic degradation. η_{impl} is useful in the quantitative understanding of parallel overhead that arises from communication, redundant computation, and synchronization.

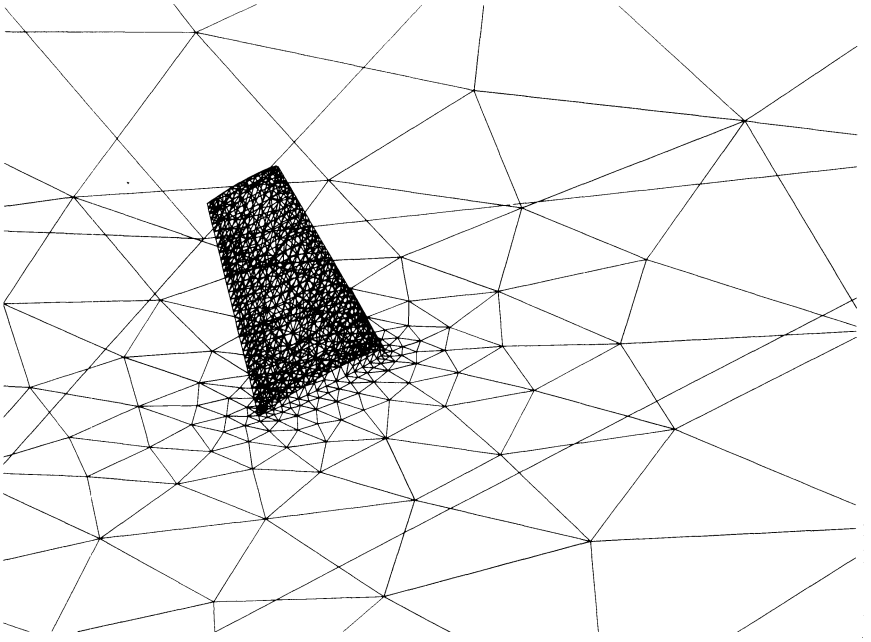

FIGURE 2. *The surface triangulation of the M6 wing, the symmetry root plane, and the farfield bounding surface are shown for a relatively coarse grid.*

TABLE 2. Cray T3E parallel performance (2,761,774 vertices)

				Efficiency			
procs	its	time	speedup	η_{alg}	η_{impl}	$\eta_{overall}$	Gflop/s
128	164	6,048.37s	1.00	1.00	1.00	1.00	8.5
256	166	3,242.10s	1.87	0.99	0.94	0.93	16.6
512	171	1,811.13s	3.34	0.96	0.87	0.83	32.1

Table 2 shows a relative efficiency of 83% over the relevant range for a problem of $4 \times 2,761,774$ DOFs. Each iteration represents up to 45 preconditioned GMRES iterations (with restarting every 16 iterations). This grid is the largest yet generated by our colleagues at NASA Langley for an implicit wing computation. Coordinate and index data (including 18 million edges) alone occupy an 857 MByte file.

The 32.1 Gflop/s achieved on 512 nodes for this sparse unstructured computation is 12% of the best possible rate of 265 Gflop/s for the dense LINPACK benchmark on a matrix of order 79,744 (with 6.36 billion nonzeros) on the identical configuration [7]. The principal "slow" routines (at present) are orthogonalization in the Krylov method and subdomain Jacobian preconditioner formation (soon to be addressed), together accounting for about 20–23% of execution time and running at only 20% of the overall sustained Gflop/s rate.

It is interesting to note the source of the degradation of η_{impl} in going from 128 to 512 processors, since much finer granularities will be required in ASCI-scale computations. The maximum over all processors of the time spent at global

TABLE 3. Cray T3E parallel performance — Gustafson scaling

vert	procs	vert/proc	its	time	time/it
357,900	80	4474	78	559.93s	7.18s
53,961	12	4497	36	265.72s	7.38s
9,428	2	4714	19	131.07s	6.89s

synchronization points (reductions — mostly inner products and norms) is 12% of the maximum over all processors of the wall-clock execution time. This is almost entirely idle time arising from load imbalance, not actual communication time, as demonstrated by inserting barriers before the global reductions and noting that the resulting fraction of wall-clock time for global reductions drops below 1%. Closer examination of partitioning and profiling data shows that although the distribution of "owned" vertices is nearly perfectly balanced, and with it the "useful" work, the distribution of ghosted nodes can be very imbalanced, and with it, the overhead work and the local communication requirements. In other words, the partitioning objective of minimizing total edges cut while equidistributing vertices does *not*, in general, equidistribute the execution time between synchronization points, mainly due to the skew among the processors in ghost vertex responsibilities. This example of the necessity of supporting multiple objectives (or multiple constraints) in mesh partitioning has been communicated to the authors of major partitioning packages, who have been hearing it from other sources, as well. For PDE codes amenable to per-iteration communication and computation work estimates that are not data-dependent, it is plausible to approximately balance multiple distinct phases in an *a priori* partitioning. More generally, partitionings may need to be rebalanced dynamically, on the basis of real-time measurements rather than models. This will require integration of load balancing routines with the solution routines in parallel. We expect that a similar computation after such higher level needs are accommodated in the partitioner will achieve close to 95% overall efficiency on 512 nodes.

Since we possess a sequence of unstructured Euler grids for the same wing, we can perform a Gustafson-style scalability study by varying the number of processors and the discrete problem dimension in proportion. We note that the concept of Gustafson-style scalability does not extend perfectly cleanly to nonlinear PDEs, since added resolution brings out added physics and (generally) poorer conditioning, which may cause a shift in the "market basket" of kernel operations as the work in the nonlinear and linear phases varies. However, our shockless Euler simulation is a reasonably clean setting for this study, if corrected for iteration count. Table 3 shows three computations on the T3E over a range of 40 in problem and processor size, while maintaining approximately 4,500 vertices per processor.

The good news in this experiment is contained in the final column, which shows the average time per parallelized pseudo-transient NKS outer iteration for problems with similarly sized local workingsets. Less than a 7% variation in performance occurs over a factor of nearly 40 in scale.

4.2. Serial Cache Optimization Results. From a processor perspective we have so far looked outward rather than inward. Since the aggregate computational rate is a product of the concurrency and the rate at which computation occurs in

TABLE 4. IBM P2SC cache performance in serial (22,677 vertices)

	Enhancements			Results			
	Field Interlacing	Structural Blocking	Edge Reordering	its	time	time/it	impr. ratio
1				28	2905s	103.8s	—
2	×			25	1147s	45.9s	2.26
3	×	×		25	801s	32.0s	3.24
4	×		×	25	673s	26.9s	3.86
5	×	×	×	25	373s	14.9s	6.97

a single active thread, we briefly discuss the per-node performance of the legacy and the PETSc ported codes. Table 4 shows the effect, individually or in various combinations, of three cache-related performance enhancements, relative to the original vector-oriented code, whose performance is given in row 1. Since the number of iterations differs slightly in the independent implementations, we normalize the execution time by the number of iterations for the comparisons in the final column. These optimizations are described in more detail in [**9**]. We observe certain synergisms in cache locality; for instance, adding structural blocking to the interlaced code without edge-reordering provides a factor of 1.43, while adding structural blocking to the edge-reordered code provides a factor of 1.81. Similarly, adding edge-reordering to a code without structural blocking provides a factor of 1.71, while adding structural edge-reordering to the blocked code provides a factor of 2.15. Including the iteration count benefit, the cache-oriented serial code executes 7.79 times faster than the original, before parallelization.

5. Conclusions

We have demonstrated very respectable scaling behavior for a ΨNKS version of a 3D unstructured CFD code. We began with a legacy vector-oriented code known to be algorithmically competitive with multigrid in 2D, improved its performance as far as we could for a sequential cache orientation, and then parallelized it with minimal impact on the sequential convergence rate. The parallel version can be scaled to accommodate very rich grids.

Profiling the highest granularity runs reveals certain tasks that need additional performance tuning — load balancing being the least expected. With respect to the interaction of algorithms with applications we believe that the ripest remaining advances are interdisciplinary: ordering, partitioning, and coarsening must adapt to coefficients (and thus grid spacing, flow magnitude, and flow direction) for convergence rate improvement. Trade-offs between grid sequencing, pseudo-time iteration, nonlinear iteration, linear iteration, and preconditioner iteration must be further understood and exploited.

Acknowledgements

The authors thank W. Kyle Anderson of the NASA Langley Research Center for providing FUN3D. Satish Balay, Bill Gropp, and Lois McInnes of Argonne National Laboratory co-developed (with Smith) the PETSc software employed in this paper. Computer time was supplied by NASA (under the Computational Aero Sciences

section of the High Performance Computing and Communication Program), and DOE (through Argonne and NERSC).

References

1. W. K. Anderson, *FUN2D/3D*, `http://fmad-www.larc.nasa.gov/~wanderso/Fun/fun.html`, 1997.
2. S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, *Efficient management of parallelism in object-oriented numerical software libraries*, Modern Software Tools in Scientific Computing, Birkhauser, 1997, pp. 163–201.
3. ———, *The Portable, Extensible Toolkit for Scientific Computing, version 2.0.21*, `http://www.mcs.anl.gov/petsc`, 1997.
4. X.-C. Cai, *Some domain decomposition algorithms for nonselfadjoint elliptic and parabolic partial differential equations*, Tech. report, Courant Institute, NYU, 1989.
5. X.-C. Cai, D. E. Keyes, and V. Venkatakrishnan, *Newton-Krylov-Schwarz: An implicit solver for CFD*, Proceedings of the Eighth International Conference on Domain Decomposition Methods, Wiley, 1997, pp. 387–400.
6. X.-C. Cai and M. Sarkis, *A restricted additive Schwarz preconditioner for nonsymmetric linear systems*, Tech. Report CU-CS-843-97, Computer Science Dept., Univ. of Colorado at Boulder, August 1997, `http://www.cs.colorado/edu/cai/public_html/papers/ras_v0.ps`.
7. J. J. Dongarra, *Performance of various computers using standard linear equations software*, Tech. report, Computer Science Dept., Univ. of Tennessee, Knoxville, 1997.
8. W. D. Gropp, L. C. McInnes, M. D. Tidriri, and D. E. Keyes, *Parallel implicit PDE computations: Algorithms and software*, Proceedings of Parallel CFD'97, Elsevier, 1998.
9. D. K. Kaushik, D. E. Keyes, and B. F. Smith, *Cache optimization in multicomponent unstructured-grid implicit CFD codes*, (in preparation), 1998.
10. ———, *Newton-Krylov-Schwarz parallelization of unstructured-grid legacy CFD codes using PETSc*, (in preparation), 1998.
11. C. T. Kelley and D. E. Keyes, *Convergence analysis of pseudo-transient continuation*, SIAM J. Num. Anal. (to appear), 1998.
12. W. Mulder and B. Van Leer, *Experiments with implicit upwind methods for the Euler equations*, J. Comp. Phys. **59** (1995), 232–246.
13. Y. Saad and M. H. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput. **7** (1986), 856–869.

COMPUTER SCIENCE DEPARTMENT, OLD DOMINION UNIVERSITY, NORFOLK, VA 23529-0162 AND ICASE, NASA LANGLEY RES. CTR., HAMPTON, VA 23681-2199
E-mail address: `kaushik@cs.odu.edu`

COMPUTER SCIENCE DEPARTMENT, OLD DOMINION UNIVERSITY, NORFOLK, VA 23529-0162 AND ICASE, NASA LANGLEY RES. CTR., HAMPTON, VA 23681-2199
E-mail address: `keyes@icase.edu`

MATHEMATICS AND COMPUTER SCIENCE DIVISION, ARGONNE NATIONAL LABORATORY ARGONNE, IL 60439-4844
E-mail address: `bsmith@mcs.anl.gov`