

Robust Algorithms and Software for Parallel PDE-Based Simulations ^{*}

S. Bhowmick¹, L. McInnes², B. Norris², and P. Raghavan¹

¹ Department of Computer Science and Engineering, The Pennsylvania State University,
220 Pond Lab, University Park, PA 16802-6106

E-mail: {bhowmick,raghavan}@cse.psu.edu

² Mathematics and Computer Sciences Division, Argonne National Laboratory,
9700 South Cass Ave., Argonne IL 60439-4844

E-mail: {curfman,norris}@mcs.anl.gov

Keywords: software architecture, iterative linear solvers, component software, multimethod solvers

Abstract

The solution of nonlinear partial differential equations (PDEs), such as those arising in computational fluid dynamics, forms an important class of applications in scientific computing. Simulation times typically depend to a large extent on the robustness and efficiency of the sparse linear solver used at each iteration of the nonlinear PDE solver. We consider algorithms and software to develop robust and efficient solvers as composites of multiple preconditioned iterative methods. Iterative solvers allow scalable parallel implementation but can often converge slowly or fail to converge. Our methods address this deficiency by using a sequence of iterative methods to provide a highly reliable solution with good parallel performance. We describe our algorithms and software for instantiating such multimethod composite solvers and report on their performance in a driven cavity flow application.

1 INTRODUCTION

The numerical solution of large-scale nonlinear PDEs using implicit and semi-implicit schemes requires the solution of linear systems at each nonlinear iteration. The

linear solution time often dominates the total time of the computation. A large variety of linear solvers, including both direct and iterative solution methods, are available [2, 12, 14]. However, it is very difficult to select a linear solution method that is robust and provides the best overall performance [13]. Direct methods are robust but their memory requirements grow nonlinearly with the matrix dimension because of fill (i.e., zeros becoming non-zeros) during factorization. In addition, the cost of factorization is incurred repeatedly during the simulation when the coefficient matrix changes at each nonlinear iteration. Consequently, Krylov iterative methods are more suitable for such applications. Their memory requirements scale with the matrix dimension and they allow effective parallelization. Iterative methods, however, are less reliable in general than direct methods. Depending on the problem, iterative methods may converge slowly or fail altogether. Preconditioners are often used to improve the convergence. Combining preconditioners with the Krylov subspace iterative methods [2] results in a large number of solvers that differ greatly in computational costs and convergence behaviors. Now, the application developer is confronted by the challenge of selecting the right solver from a large set of candidate methods. Our algorithms and software are aimed at helping the application developer address this issue. We develop techniques for defining a composite solver that can meet application demands by using a sequence of preconditioned iterative methods.

^{*} This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-ENG-38, and by the National Science Foundation through grants ACI-0102537, DMR-0205232, and ECS-0102345.

2 MULTIMETHOD SOLVERS

The motivation for using multiple solver methods arises from the difficulty of selecting the best linear solution scheme [5, 9, 13]. Our work includes *adaptive* techniques that adapt the solution method to match changing problem attributes in a long-running simulation [8, 17]. We also consider *composites* that use a sequence of methods on the same linear system to provide a highly reliable solution [6, 8]. In this paper we discuss the software environment for instantiating composite solvers in advanced uniprocessor and multiprocessor architectures.

We have earlier developed composite solvers [6] that comprise a sequence of different base algorithms. The failure of one method results in the invocation of the next method in the sequence until the given problem is solved successfully. The reliability of method M_i is defined as $r_i = 1 - f_i$, where f_i is the failure rate. If we assume that the failures of the base methods are mutually independent, then the reliability of the composites is a product of the reliabilities of the base methods and is therefore independent of the order in which the methods are invoked. However, the total running time of the composite depends on the ordering of the methods. We can thus define composites by the sequence in which the base methods are invoked. Consider the base methods M_1, M_2, \dots, M_n . Let π be a permutation of $1, \dots, n$, and let $\pi(j)$ denote its j -th element; $M_{\pi(j)}$ represents the j -th method in the corresponding composite, C_π . The worst case time required by C_π is given by

$$T_\pi = t_{\pi(1)} + f_{\pi(1)}t_{\pi(2)} + \dots + (f_{\pi(1)} \dots f_{\pi(n-1)})t_{\pi(n)}.$$

An optimal composite solver would have the minimum worst case running time among all possible composites. Our earlier paper [7] proves that such a composite is formed by arranging the base methods in increasing order of the *utility ratio*, $u_i = \frac{t_i}{r_i}$. This ratio can be computed using estimates for t_i and $r_i = 1 - f_i$ through suitable sampling. For example, all base methods can be executed on a small set of representative problems (sample); the utility ratios are then computed using mean observed values of the times per iteration and the failure rates. The optimal composite, henceforth denoted by CU, has base methods arranged in increasing order of u_i .

Our earlier work on sequential composite solvers demonstrated that the reliability of the composite is much

higher than any of the constituent methods. Additionally, for the optimal composite, CU, the number of nonlinear iterations are reduced; thus, simulation times can be significantly reduced. We now focus on the parallelization of composite solvers for systems of the form $Ax = b$, where A is the coefficient matrix, b is the right-hand-side vector, and x is the solution vector. Consider a set of fully parallel base linear solution methods; each of these could either be a simple Krylov iteration with or without a suitable preconditioner. All of these methods are assumed to use the same data distribution across P processes of A , x , and b . Assume that a composite is constructed using some specific ordering of the base methods. Upon failure of a base method, the next method in the sequence can be easily invoked on the same system without any extra data redistribution. Additionally, as in the sequential case, the data structures used by the failed method are released and the preconditioner for the next Krylov method is constructed, etc. Although such a parallel composite is conceptually quite simple, its implementation can be remarkably complex, as discussed in the next section. The complexity stems from the need to access implementations of a variety of base methods, preconditioners, data distributions, mapping schemes, and application interfaces.

3 SOFTWARE ENVIRONMENT

A software architecture that enables the implementation and use of robust multimethod linear solvers is illustrated in Figure 1. A “proxy” linear solver method mediates the interactions between the nonlinear solution algorithm and the linear solver algorithms. Both the proxy and concrete linear solver methods implement the same abstract linear solver interface, which is used by the nonlinear solver and the application driver to create, configure, and solve the linear systems arising in the course of the nonlinear solution. This approach not only allows the easy substitution of one linear solution method for another, but also enables multimethod solvers to be incorporated without a change to the client nonlinear algorithms.

To implement our composite solver strategy, we employ a linear solver proxy that mediates the selection of a sequence of solvers to be used according to some ordering strategy produced by the ordering agent component in Figure 1. As described in Section 2, our composite ordering

strategy selects linear solution methods based on their utility ratios, which are obtained from past performance history. The linear solver proxy can be used in a black-box fashion with a Newton-Krylov solver. Alternatively, the Newton-Krylov solver can influence the ordering of linear solver methods by interacting with the ordering agent. In this manner, domain or problem specific knowledge can be used to influence the selection of linear solver algorithms. For example, the nonlinear solver may specify the relative accuracy that is required for achieving good nonlinear convergence during different phases of the computation. No implementation changes are made to the concrete linear solver algorithms in order to use them as parts of a composite.

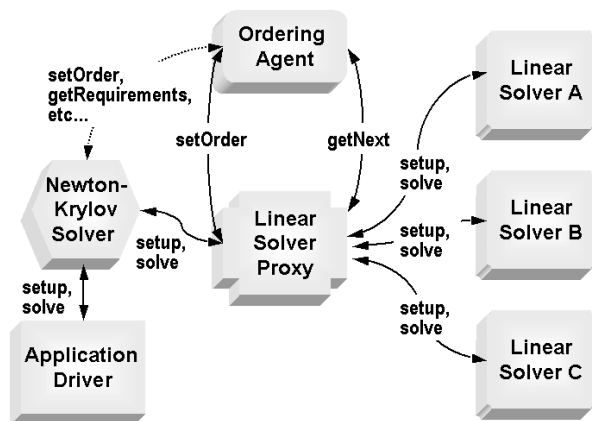


Fig. 1. Multimethod software architecture.

The Portable Extensible Toolkit for Scientific Computation (PETSc) [3,4] supports flexibility in algorithm selection, including our multimethod schemes. The linear solvers are accessed via an abstract interface, and it is possible to control linear solver selection at run time, if desired. Thus, we are able to provide applications with robust composite solvers without imposing special requirements on the nonlinear solver implementation or the user’s code.

Our multimethod research contributes to ongoing community efforts in developing high-performance component software for scientific computations [1, 15, 19, 20]. Common linear solver interfaces allow the definition of linear solver components that can be used in a plug-and-play fashion, and also enable the seamless incorporation of mul-

timethod strategies. In addition, a component environment makes it possible to define a *quality of service* architecture for scientific applications [16]. Numerical components implemented in such an environment can take advantage of more intelligent algorithm choices that take into account the quality requirements (e.g., accuracy, execution time) of a given client algorithm, such as the Newton-Krylov solver in the example scenario in 1. Furthermore, this approach enables the transparent use of auxiliary components to monitor system performance, retrieve and analyze past performance information, and make algorithm selection decisions.

4 PROBLEM DESCRIPTION AND EXPERIMENTAL RESULTS

In this section, we provide some sample empirical results to demonstrate the effectiveness of our multimethod schemes. For our test application, we used a driven cavity flow simulation described in detail in [11]. The driven cavity flow model is an example of incompressible flow in a two-dimensional rectangular cavity. The governing differential equations are obtained by using the Navier-Stokes and energy equations. We solve the system of nonlinear equations using an inexact Newton method, with a line search technique to extend its radius of convergence [18].

Earlier, as reported in [8], our optimal composite performed significantly better than the base methods (and composites with arbitrary orderings) on uniprocessors (see Figure 2). Over 24 simulations, all composites had no failures and achieved reductions in total nonlinear iterations as a consequence of reliable linear solution. In addition, the optimal composite, CU, incurred the least total linear solution time, approximately 56% of the time required by the best base method. The linear solution time comprises on average 96% of the total simulation time and consequently, CU required less than 58% of the total simulation time of the best base method.

We now consider results of experiments using our parallel composite solvers, where we employed a cluster with a Myrinet 2000 network and 2.4 GHz Pentium Xeon processors with 1-2 GB of RAM. We used a 128×128 mesh for discretizing the driven cavity flow model. At each nonlinear iteration, the discretized sparse linear system had rank 65,536 with approximately 1,302,528 nonzeros. We

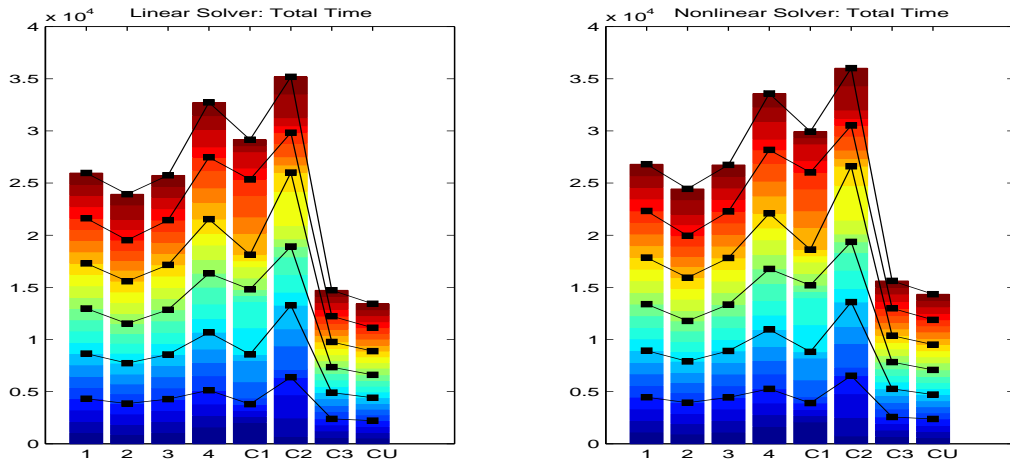


Fig. 2. Sequential performance of the driven cavity application.

set the maximum number of linear iterations to 200. As the preconditioner we used the restricted additive Schwarz method (RASM) [10] with varying subdomain solvers and varying degrees of overlap (e.g., overlap of 1 is denoted by RASM(1)).

We combined various Krylov methods [2, 14], subdomain solvers, and degrees of overlap to get the following set of base solution methods: (1) GMRES(30), RASM(1), Jacobi subdomain solver; (2) GMRES(30), RASM(1), SOR subdomain solver; (3) TFQMR, RASM(3), no-fill ILU subdomain solver; and (4) TFQMR, RASM(4), no-fill ILU subdomain solver. We composed a sample set of the following problem instances of Grashof numbers and lid velocities, which are parameters in the driven cavity model: (700:85), (800:83), (900:80), (950:73). We distributed the matrix across multiple processors and used each of the four base methods to solve the sparse linear systems that were generated by these problem instances. We obtained the resulting utility ratios by computing the ratio of the linear time per iteration to the reliability of each method, and then taking the average over the four problems in the sample set. Based on metrics obtained from the sample set, we formed the optimal composite, CU, which contained the following sequence of base methods: 3,4,2,1. We also created 3 arbitrary composites using random sequences, denoted by C1 (3,2,4,1), C2 (2,1,4,3) and C3 (4,1,2,3). We performed these experiments on a fixed

size problem, while varying the number of processors from 2, 4, 8 to 16. We ran 24 simulations, using six different Grashof numbers, [700, 750, 800, 850, 900, 950], and four lid velocities, [73, 80, 83, 85].

Our results show that the composites achieve improved reliability and good parallel performance. Base method 1 never converges, while other base methods have varying degrees of failure. The composites show near ideal reliability. Furthermore, the simulation time with the optimal composite, CU, is approximately 40% – 48% of the worst base method. We also consider the speedup and efficiency of the solvers on different processors. We use T_1 as the best estimate of the time for the full simulation on one processor using the fastest base method. Thus, we set T_1 to two times the time for method 3 on two processors (it is not meaningful to use the additive Schwarz method for a single processor with a single subdomain). We calculate the speedup $S = \frac{T_1}{T_p}$, where T_p is the observed time on p processors; the corresponding efficiency is calculated as $E = \frac{S}{p}$. The results shown in Figure 3 indicate that the speedups of the composites CU, C1, and C3 are almost as good as that of the best base method, with near ideal efficiencies.

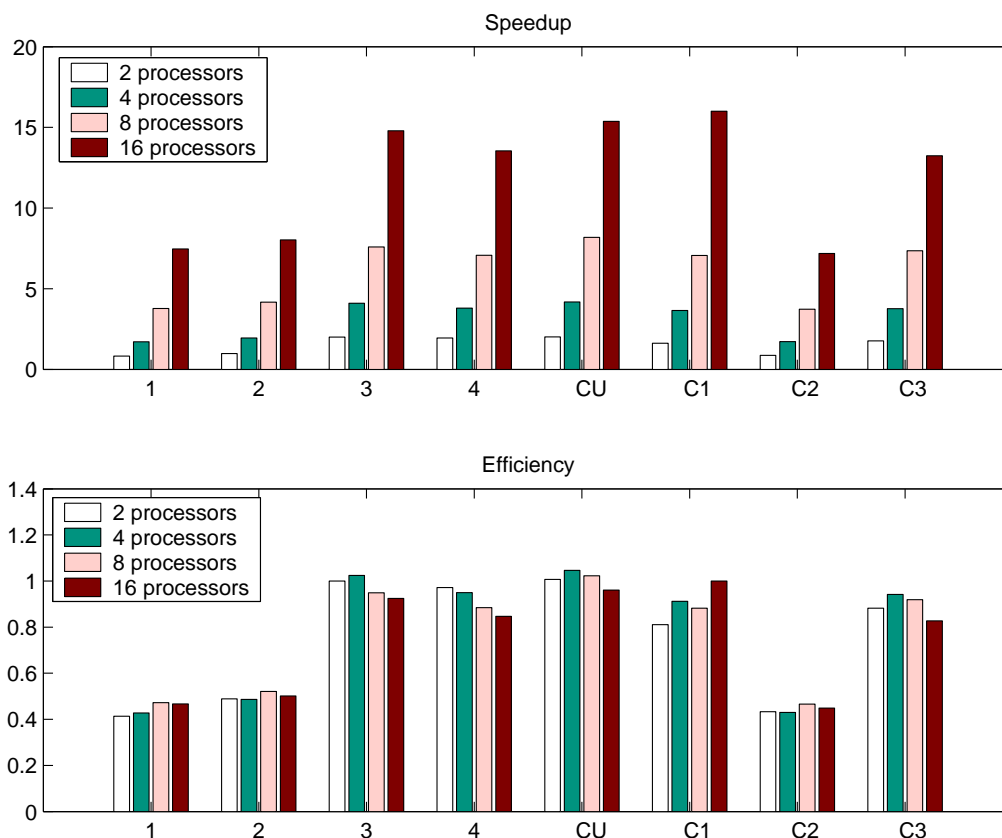


Fig. 3. Parallel performance of the driven cavity application on 2 – 16 processors.

5 CONCLUSIONS

Our work demonstrates that serial and parallel multimethod composite solvers can be developed to improve the performance of PDE-based simulations. We show that effective instantiation of such schemes through advanced software frameworks can lead to significant code reuse and ease of implementation.

We continue to develop and evaluate multimethod solvers for large-scale applications from several problem domains. We plan to study the role of multimethod schemes for other phases of PDE-based simulations, such as discretization and Jacobian evaluation. We are also working on incorporating our multimethod schemes in component-based software environments and integrating

our implementation in a quality-of-service infrastructure for scientific component applications [1, 16, 19].

References

1. R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. C. McInnes, S. Parker, and B. Smolinski, "Toward a Common Component Architecture for High-Performance Scientific Computing", In *Proceedings of High Performance Distributed Computing* (1999) pp. 115-124, (see www.cca-forum.org).
2. O. Axelsson, *A Survey of Preconditioned Iterative Methods for Linear Systems of Equations*, BIT, 25 (1987) pp. 166–187.
3. S. Balay, K. Buschelman, W. Gropp, D. Kaushik, M. Knepley, L.C. McInnes, B. Smith, and H. Zhang,

- “PETSc Users Manual”, Tech. Rep. ANL-95/11 - Revision 2.1.6, Argonne National Laboratory, 2003 (see www.mcs.anl.gov/petsc).
4. S. Balay, W. Gropp, L.C. McInnes, and B. Smith, “Efficient Management of Parallelism in Object Oriented Numerical Software Libraries”, In *Modern Software Tools in Scientific Computing* (1997), E. Arge, A. M. Bruaset, and H. P. Langtangen, Eds., Birkhauser Press, pp. 163–202.
 5. R. Barrett, M. Berry, J. Dongarra, V. Eijkhout, and C. Romine, “Algorithmic Bombardment for the Iterative Solution of Linear Systems: A PolyIterative Approach”, *J. Comp. Appl. Math.*, 74 (1996), pp. 91-109.
 6. S. Bhowmick, P. Raghavan, L. McInnes, and B. Norris, “Faster PDE-Based Simulations Using Robust Composite Linear Solvers”, Argonne National Laboratory preprint ANL/MCS-P993-0902, 2002. To appear in *Future Generation Computer Systems*.
 7. S. Bhowmick, P. Raghavan, and K. Teranishi, “A Combinatorial Scheme for Developing Efficient Composite Solvers”, *Lecture Notes in Computer Science*, Eds. P. M. A. Sloot, C.J. K. Tan, J. J. Dongarra, A. G. Hoekstra, 2330, Springer Verlag, Computational Science- ICCS 2002, (2002) pp. 325-334.
 8. S. Bhowmick, L. McInnes, B. Norris, and P. Raghavan, “The Role of Multi-Method Linear Solvers in PDE-Based Simulation”, In *Proceedings of the 2003 International Conference on Computational Science and its Applications, ICCSA 2003*, Montreal, Canada May 18 - May 21, 2003. Lecture notes in Computer Science 2677, Editors V. Kumar, M. L. Gavrilova C. J. K. Tan, and P. L’Ecuyer, (2003) pp. 828–839.
 9. R. Bramley, D. Gannon, T. Stuckey, J. Villacis, J. Balasubramanian, E. Akman, F. Berg, S. Diwan, and M. Govindaraju, “The Linear System Analyzer”, *Enabling Technologies for Computational Science*, Kluwer, (2000).
 10. X.-C. Cai and M. Sarkis, “A restricted additive Schwarz preconditioner for general sparse linear systems”, *SIAM J. Sci. Comp.*, 21 (1999) pp. 792–797.
 11. T. S. Coffey, C. T. Kelley, and D. E. Keyes, “Pseudo-Transient Continuation and Differential-Algebraic Equations”, *SIAM J. Sci. Comp.*, to appear.
 12. I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*, Clarendon Press, Oxford, 1986.
 13. A. Ern, V. Giovangigli, D. E. Keyes, and M. D. Smooke, “Towards Polyalgorithmic Linear System Solvers For Nonlinear Elliptic Problems”, *SIAM J. Sci. Comput.*, Vol. 15, No. 3, pp. 681-703.
 14. R. Freund, G. H. Golub, and N. Nachtigal, *Iterative Solution of Linear Systems*, Acta Numerica, Cambridge University Press, (1992) pp. 57–100.
 15. T. Goodale, G. Allen, G. Lanfermann, J. Masso, T. Radke, E. Seidel, and J. Shalf, “The Cactus Framework and Toolkit: Design and Applications”, *CECPAR2002, Lecture Notes in Computer Science*, 2002.
 16. P. Hovland, K. Keahey, L. C. McInnes, B. Norris and L. F. Diachin, and P. Raghavan. “A Quality of Service Approach for High-Performance Numerical Components”, In *Proceedings of Workshop on QoS in Component-Based Software Engineering*, Software Technologies Conference, Toulouse, France, June 20, 2003.
 17. L. McInnes, B. Norris, S. Bhowmick, and P. Raghavan, “Adaptive Sparse Linear Solvers for Implicit CFD Using Newton-Krylov Algorithms”, appeared in the *Proceedings of the Second MIT Conference on Computational Fluid and Solid Mechanics*, Massachusetts Institute of Technology, Boston, USA, June 17-20, 2003.
 18. J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer-Verlag, New York, 1999.
 19. B. Norris, S. Balay, S. Benson, L. Freitag, P. Hovland, L. McInnes, and B. F. Smith, “Parallel Components for PDEs and Optimization: Some Issues and Experiences”, *Parallel Computing*, 28 (12) (2002), pp. 1811-1831.
 20. M. Parashar and J. C. Browne, “System Engineering for High Performance Computing Software: The HDDA/DAGH Infrastructure for Implementation of Parallel Structured Adaptive Mesh Refinement”, IMA Volume 117: *Structured Adaptive Mesh Refinement (SAMR) Grid Methods*, Editors: S. B. Baden, N. P. Chrisochoides, D. B. Gannon, and M. L. Norman, Springer-Verlag, pp. 1 – 18, January 2000.