

The Role of Multi-Method Linear Solvers in 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 {`mcinnes,norris`}@`mcs.anl.gov`.

Abstract. The solution of large-scale, nonlinear PDE-based simulations typically depends on the performance of sparse linear solvers, which may be invoked at each nonlinear iteration. We present a framework for using multi-method solvers in such simulations to potentially improve the execution time and reliability of linear system solution. We consider *composite solvers*, which provide reliable linear solution by using a sequence of preconditioned iterative methods on a given system until convergence is achieved. We also consider *adaptive solvers*, where the solution method is selected dynamically to match the attributes of linear systems as they change during the course of the nonlinear iterations. We demonstrate how such multi-method composite and adaptive solvers can be developed using advanced software architectures such as PETSc, and we report on their performance in a computational fluid dynamics application.

1 Introduction

Many areas of computational modeling, for example, turbulent fluid flow, astrophysics, and fusion, require the numerical solution of nonlinear partial differential equations with different spatial and temporal scales [13, 18]. The governing equations are typically discretized to produce a nonlinear system of the form $f(u) = 0$, where $f : \mathcal{R}^n \rightarrow \mathcal{R}^n$. When such systems are solved using semi-implicit or implicit methods, the computationally intensive part often involves the solution of large, sparse linear systems.

A variety of competing algorithms are available for sparse linear solution, including both direct and iterative methods. Direct methods rely on sparse matrix factorization and will successfully compute the solution if one exists. However, sparse factorization causes zeroes in the coefficient matrix to become nonzeros

[★] This work was supported in part by the Maria Goeppert Mayer Award from the U. S. Department of Energy and Argonne National Laboratory, by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational Technology Research, 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.

in the factor, thus increasing memory demands. Hence, the method could fail for lack of memory. Additionally, even when sufficient memory is available, the computational cost might be quite high, especially when the sparse coefficient matrix changes at each nonlinear iteration and must therefore be refactored. Krylov subspace iterative methods require little additional memory, but they are not robust; convergence can be slow or fail altogether. Various preconditioners can be combined with Krylov methods to produce solvers with vastly differing computational costs and convergence behaviors. It is typically neither possible nor practical to predict a priori which sparse linear solver algorithm works best for a given class of problems. Furthermore, the numerical properties of the linear systems can change during the course of the nonlinear iterations. Consequently, the most effective linear solution method need not be the same at different stages of the simulation. This situation has motivated us (and other researchers) to explore the potential benefits of multi-method solver schemes for improving the performance of PDE-based simulations [5, 8, 11].

Ern et al have investigated the performance of several preconditioned Krylov iterations and stationary methods in nonlinear elliptic PDEs [11]; their work is towards a polyalgorithmic approach. Polyalgorithms have been studied for multiprocessor implementations by Barrett et al in a formulation where different types of Krylov methods are applied in parallel to the same problem [5]. Our work on multi-method solvers for PDE-based applications focuses on formulating general-purpose metrics that can be used in different ways to increase reliability and reduce execution times. We also consider the instantiation of such multi-method schemes using advanced object-oriented software systems, such as PETSc [3, 4], with well-defined abstract interfaces and dynamic method selection.

The first type of multi-method solver we consider is a *composite*, which comprises several underlying (base) methods; each base method is applied in turn to the same linear system until convergence is achieved. A composite solver could fail in the event that all underlying methods fail on the problem, but in practice this is rare even with as few as three or four methods. The composite is thus more reliable than any of its constituent methods. However, the order in which base methods are applied can significantly affect total execution time. We have developed a combinatorial scheme for constructing an optimal composite in terms of simple observable metrics such as execution time and mean failure rates [6, 7]. The second type of multi-method scheme is an *adaptive* solver; such a solver once again comprises several base methods, but only one base method is applied to a given linear system. The numerical properties of the linear systems can change during the course of nonlinear iterations [15], and our adaptive solver attempts to select a base method for each linear system that best matches its numerical attributes. Once again, our focus is on automated method selection using metrics such as normalized time per iteration and both linear and nonlinear convergence rates.

In this paper, we provide overviews of our composite and adaptive solver techniques and demonstrate their effectiveness in a computational fluid dynam-

ics application. In Section 2, we describe a PDE-based simulation of driven cavity flow and our solution approach using inexact Newton methods. In Section 3 we illustrate how an optimal composite solver can be constructed using a combinatorial scheme to yield ideal reliability for linear solution and an overall faster simulation. In Section 4, we describe our method for constructing an adaptive solver and report on observed reductions in simulation times. Section 5 contains concluding remarks.

2 Problem Description and Solution Methods

We consider a driven cavity flow simulation to illustrate the benefits of multi-method solvers in the context of nonlinear PDEs. This simulation incorporates numerical features that are common in many large-scale scientific applications, yet is sufficiently simple to enable succinct description. We introduce the model problem in Section 2.1 and discuss our solution schemes using inexact Newton methods in Section 2.2.

2.1 Simulation of Driven Cavity Flow

The driven cavity flow model combines lid-driven flow and buoyancy-driven flow in a two-dimensional rectangular cavity. The lid moves with a steady and spatially uniform velocity, and thus sets a principal vortex and subsidiary corner vortices. The differentially heated lateral walls of the cavity induce a buoyant vortex flow, opposing the principal lid-driven vortex. See [9] for a detailed problem description. We use a velocity-vorticity formulation of the Navier-Stokes and energy equations, in terms of the velocity u_x, u_y in the (x, y) directions and the vorticity ω on a domain Ω defined as $\omega(x, y) = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}$.

This governing differential equations are:

$$-\Delta u_x + \frac{\partial \omega}{\partial y} = 0, \quad (1)$$

$$-\Delta u_y + \frac{\partial \omega}{\partial x} = 0, \quad (2)$$

$$-\Delta \omega + u \frac{\partial \omega}{\partial x} + v \frac{\partial \omega}{\partial y} - \text{Gr} \frac{\partial T}{\partial x} = 0, \quad (3)$$

$$-\Delta T + \text{Pr} \left(u \frac{\partial T}{\partial x} + v \frac{\partial T}{\partial y} \right) = 0, \quad (4)$$

where $T(x, y)$ is the temperature, Pr is a Prandtl number, and Gr is a Grashof number. The boundary conditions are $\omega(x, y) = -\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}$. We discretize this system using a standard finite difference scheme with a five-point stencil for each component on a uniform Cartesian grid. For a given problem size, the values of the Grashof number and lid velocity determine the degree of nonlinearity of the system and thus affect the convergence of the nonlinear solver.

2.2 Inexact Newton Methods

We use inexact Newton methods (see, e.g., [16]) to simultaneously solve the equations (1) through (4), which have the combined form

$$f(u) = 0. \quad (5)$$

We use a Krylov iterative method to (approximately) solve the Newton correction equation

$$f'(u^{\ell-1}) \delta u^\ell = -f(u^{\ell-1}), \quad (6)$$

in the sense that the linear residual norm $\|f'(u^{\ell-1}) \delta u^\ell + f(u^{\ell-1})\|$ is sufficiently small. We then update the iterate via $u^\ell = u^{\ell-1} + \alpha \cdot \delta u^\ell$, where α is a scalar determined by a line search technique such that $0 < \alpha \leq 1$. We terminate the Newton iterates when the relative reduction in the residual norm falls below a specified tolerance, i.e., when $\|f(u^\ell)\| < \epsilon \|f(u^0)\|$, where $0 < \epsilon < 1$. We precondition the Newton-Krylov methods, whereby we increase the linear convergence rate at each nonlinear iteration by transforming the linear system (6) into the equivalent form $B^{-1} f'(u^{\ell-1}) \delta u^\ell = -B^{-1} f(u^{\ell-1})$, through the action of a preconditioner, B , whose inverse action approximates that of the Jacobian, but at smaller cost.

At higher values of the driven cavity model's nonlinearity parameters, Newton's method often struggles unless some form of continuation is employed. Hence, for some of our experiments we incorporate pseudo-transient continuation ([9, 14]), a globalization technique that solves a sequence of problems derived from the model $\frac{\partial u}{\partial t} = -f(u)$, namely

$$g_\ell(u) \equiv \frac{1}{\tau^\ell} (u - u^{\ell-1}) + f(u) = 0, \quad \ell = 1, 2, \dots, \quad (7)$$

where τ^ℓ is a pseudo time step. At each iteration in time, we apply Newton's method to equation (7). As discussed by Kelley and Keyes [14], during the initial phase of pseudo-transient algorithms, τ^ℓ remains relatively small, and the Jacobians associated with equation (7) are well conditioned. During the second phase, the pseudo time step τ^ℓ advances to moderate values, and in the final phase τ^ℓ transitions toward infinity, so that the iterate u^ℓ approaches the root of equation (5).

Our implementation uses the Portable, Extensible Toolkit for Scientific Computation (PETSc) [3, 4], a suite of data structures and routines for the scalable solution of scientific applications modeled by PDEs. The software integrates a hierarchy of libraries that range from low-level distributed data structures for meshes, vectors, and matrices through high-level linear, nonlinear, and timestepping solvers.

3 Composite Solvers

Sparse linear solution techniques can be broadly classified as being direct, iterative, or multilevel. Methods from the latter two classes can often fail to converge

and are hence not reliable across problem domains. A composite solver uses a sequence of such inherently unreliable base linear solution methods to provide an overall robust solution. Assume that a sample set of systems is available and that it adequately represents a larger problem domain of interest. Consider a set of base linear solution methods B_1, B_2, \dots, B_n ; these methods can be tested on sample problems to obtain two metrics of performance. For method B_i , t_i represents the normalized execution time, and f_i represents the failure rate. The latter is the number of failures as a fraction of the total number of linear system solutions; the reliability or success rate is $r_i = 1 - f_i$. A composite executes base methods in a specified sequence, and the failure of a method causes the next method in the sequence to be invoked. If failures of the base methods occur independently, then the reliability of the composite is $1 - \prod_{i=1}^n f_i$, that is, its failure rate is much smaller than the failure rate of any component method. In this model, all composites are equally reliable, but, the exact sequence in which base methods are selected can affect the total execution time.

A composite is thus specified by a permutation of methods $1, \dots, n$. Although all permutations yield the same reliability of the composite, there exists some permutation that will yield the minimal worst-case average time. Computing this specific permutation will result in the development of an optimal composite. Let π be a permutation of $1, \dots, n$ and let $\pi(j)$ denote its j -th element $0 < j \leq n$; $B_{\pi(j)}$ is the j -th method in the corresponding composite C_π . The worst case time required by C_π is given by $T_\pi = t_{\pi(1)} + f_{\pi(1)}t_{\pi(2)} + f_{\pi(1)}f_{\pi(2)}t_{\pi(3)} + \dots + f_{\pi(1)}f_{\pi(2)} \dots f_{\pi(n-1)}t_{\pi(n)}$. The optimal composite corresponds to one with minimal time over all possible permutations of $1, \dots, n$. As shown in our earlier paper [6], the optimal composite is one in which the base methods are ordered in increasing order of the *utility ratio*, $u_i = \frac{t_i}{1-f_i}$ (the proof is rather complicated). This composite can be easily constructed by simply sorting the base methods in increasing order of u_i . It is indeed interesting that composites which order base methods in increasing order of time and decreasing order of failure are not optimal.

Our experiments were performed on a dedicated dual-CPU Pentium III 500 MHz workstation with 512 MB of RAM. At each iteration of the inexact Newton method we solved a sparse linear system of rank 260, 100 (with approximately 5.2 million nonzeros), which resulted from using a 128×128 mesh for the driven cavity flow application. We used the following base solution methods with a maximum iteration limit of 250: (1) GMRES(30) and an ILU preconditioner with fill level 1 with a quotient minimum degree (qmd) ordering, (2) TFQMR with an ILU preconditioner with drop threshold 10^{-2} and a reverse Cuthill Mckee (rcm) ordering, (3) GMRES(30) and an ILU preconditioner with fill level 0 and an rcm ordering, and (4) TFQMR with an ILU preconditioner with fill level 0 and an rcm ordering [2, 10, 12]. As our sample set, we used a Grashof number of 660 with a lid velocity of 10 and a Grashof number of 620 with lid velocities 13, 16, and 20. We observed the failure rates (f_i) and the mean time per iteration (t_i) of the linear solver and used these metrics to compute the utility ratio of each method; these metrics are shown in Figure 1. The optimal composite is

denoted as CU and corresponds to base methods in the sequence: 2, 3, 1, 4. We also formed three other composites using arbitrarily selected sequences: C1: 3, 1, 2, 4; C2: 4, 3, 2, 1; and C3: 2, 1, 3, 4.

We ran a total of 24 simulations with six Grashof numbers (580, 620, 660, 700, 740, and 780) and four lid velocities (10, 13, 16, and 20). We report on the performance of the four base and four composite methods using several stacked bar graphs. Each stacked bar in Figures 2 through 5 corresponds to one of the eight methods, and each segment of the stack represents a simulation point corresponding to a Grashof:lid velocity pair; the segments are arranged in increasing order of Grashof number and lid velocities (per Grashof number) with the bottom representing 580:10 and the top 780:20. Thus, starting at the top or bottom, each patch of four contiguous segments represents results for a specific Grashof value.

Figure 2 shows the number of failures and the reliability of the linear solver; each base methods suffers some failures, while all composites are robust. Figure 3 shows the time per iteration, which is nearly invariant across simulation points for a given method. As expected, the composites typically require greater time per iteration than the least expensive base method (3). Figures 4 and 5 show the total iteration counts and time for linear and nonlinear solution; the latter is the total application time. In these figures, the plots that interlace the stacked bars show cumulative values at the completion of simulations corresponding to each Grashof number. All composites show a reduction in total nonlinear iterations as a consequence of improved linear solution; CU requires only 75% (63%) of the nonlinear (linear) iterations required by the fastest base method (2). The composite based on the utility ratio, CU, incurs the least total linear solution time, which is approximately 56% of the time required by the best base method (2). The linear solution time comprises on average 96% of the nonlinear solution time (total simulation time) and consequently, CU requires approximately 58% of the total simulation time of the best base method (2).

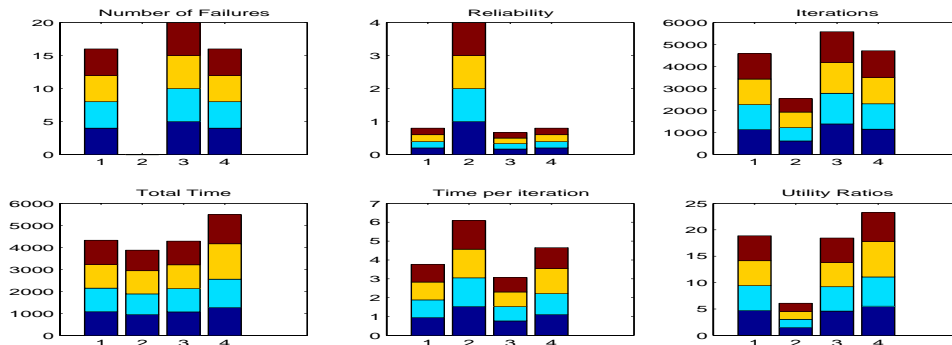


Fig. 1. Observed performance and the computed utility ratios of the four base linear solution methods at the sample simulation points. Execution times are in seconds.

For the range of observed Grashof numbers and lid velocities, many simulations showed convergence of the nonlinear solver even upon repeated failure of the linear solver. We conjecture that performance improvements from using composites could be even more dramatic in applications where nonlinear convergence depends more critically on accurate linear solution.

4 Adaptive Solvers

As discussed in Section 2.2, when using pseudo-transient Newton techniques, the conditioning of the linear systems, and the corresponding difficulty of solving them, vary throughout the overall simulation. Consequently, pseudo-transient Newton methods are natural candidates for the use of an adaptive linear solver. Recall that an adaptive linear solver attempts to select a linear solution method (from a set of base methods) that best matches the characteristics of the linear system to be solved.

We consider the development of adaptive linear solvers for the driven cavity model using a 96×96 mesh with a Grashof number of 10^5 and lid velocity of 100. We performed these experiments on a dedicated dual-CPU Intel P4 Xeon 2.2GHz workstation with 4 GB of RAM. The left-hand graph of Figure 6 plots the growth of the pseudo time step and the convergence of the nonlinear residual norm. The right-hand graph depicts the time spent in solving linear systems during each time step, using GMRES(30) and a fixed preconditioner of point-block ILU(1) with block size four. At each time step, we solve the resulting nonlinear system loosely, using a relative reduction in residual norm of 10^{-3} , which for this case requires one or two Newton iterations. We observe that as the pseudo time step size τ increases, the linear systems become more difficult to solve, as evidenced by increasing time spent in this facet of the simulation; see [11] for a discussion of similar behavior.

We develop an adaptive solver that employs different preconditioners based on the changing numeric properties of the Newton systems that correspond to

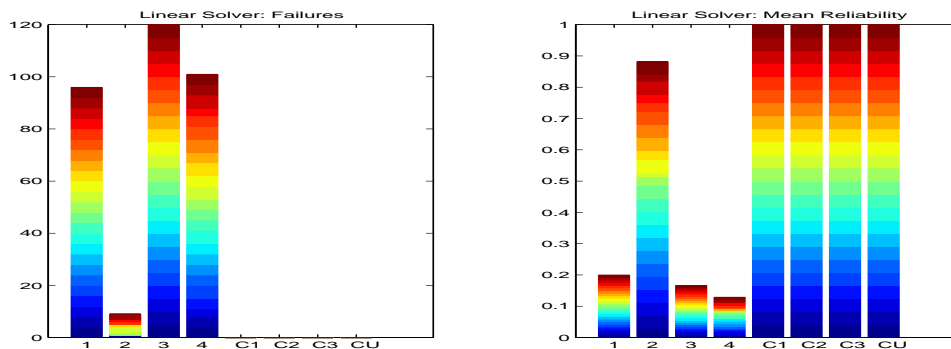


Fig. 2. The number of failures and the mean reliability of the linear solver.

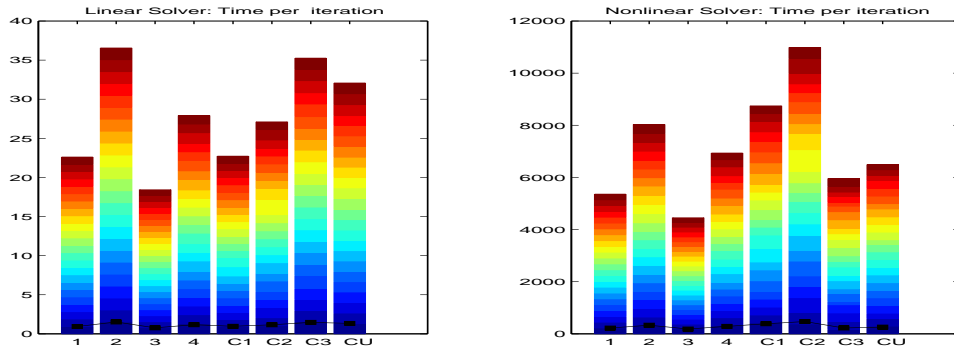


Fig. 3. Time per iteration (in seconds) for linear and nonlinear solution; the plot shows the mean time per iteration over all simulations.

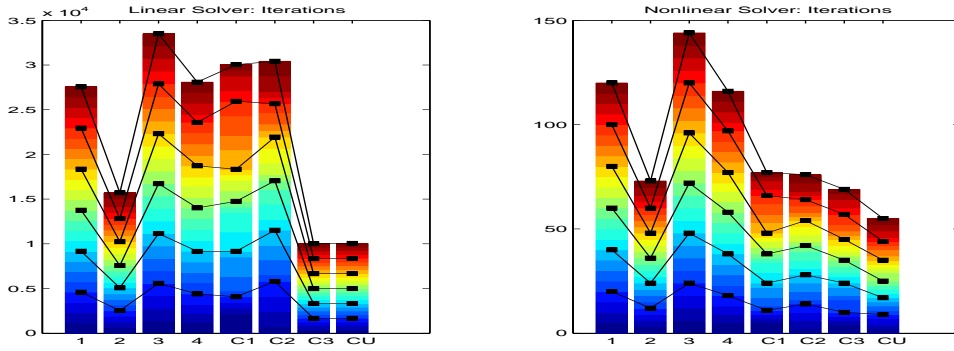


Fig. 4. Iteration counts for linear and nonlinear solution; the plots highlight cumulative values after simulations for each Grashof number.

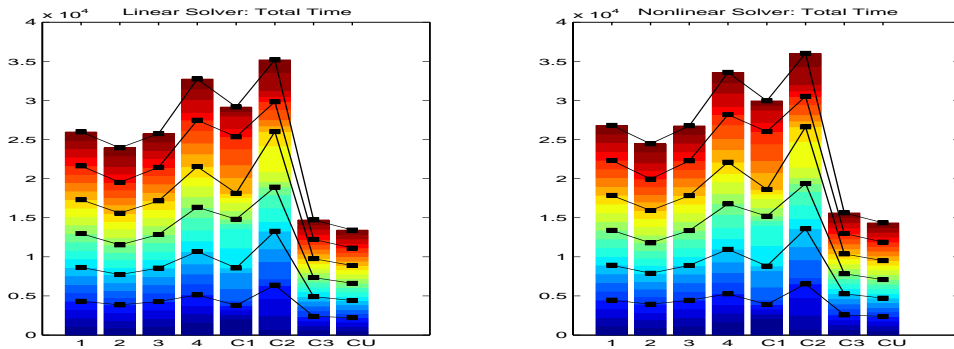


Fig. 5. Total time for linear and nonlinear solution (in seconds); the plots highlight cumulative values after simulations for each Grashof number.

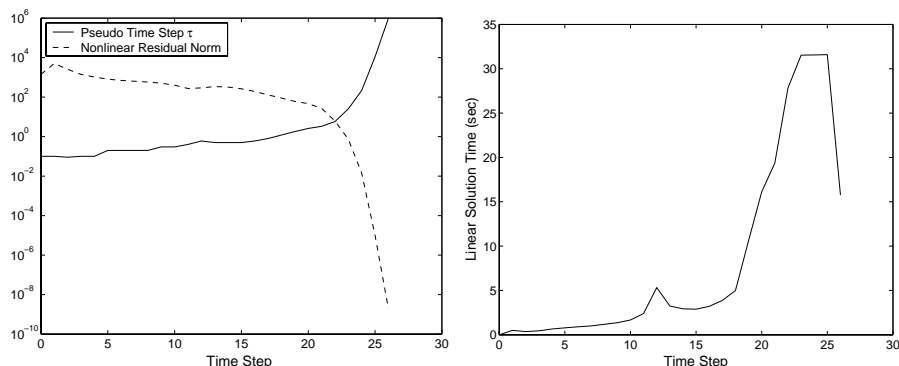


Fig. 6. Left-hand graph: Growth of pseudo time step size and convergence of residual norm for the pseudo-transient continuation version of the driven cavity problem. Right-hand graph: Plot of time spent in solving linear systems during each time step, which illustrates that different amounts of work are needed throughout the simulation to solve the linear systems.

equation (7) and the corresponding progress of the solvers during the course of the pseudo-transient iterations. We assume that for a given class of problems, base methods can be ranked clearly in terms of quality and cost. More specifically, we assume that if a method X is ranked lower than method Y , then X will require less time for its construction and application per Krylov iteration than Y , and, that it will result in slower convergence than Y . For example, we assume that using an ILU(0) preconditioner can lead to a lower-accuracy result in a shorter amount of time than an ILU(1) preconditioner. Such a ranking could also be developed based on the observed performance of the base methods for a suitable sample set of problems.

The switching criterion used in the adaptive linear solution is based on a combination of metrics:

1. The convergence rate R_L of previous linear solutions over a window of non-linear iterations,

$$R_L = \frac{1}{w} \sum_{k-w < i \leq k} \frac{r_{i,n_i} - r_{i,0}}{n_i},$$

where w is the number of time steps in the current convergence evaluation window; k is the current time step number; $r_{i,j}$ is the relative residual of the linear system solution at the i -th time step and j -th linear iteration; and n_i is the total number of linear iterations performed at the i -th time step.

2. The convergence rate R_N of the nonlinear solution over the last w time steps,

$$R_N = \frac{\|f_k\| - \|f_{k-w}\|}{\|f_{k-w}\|},$$

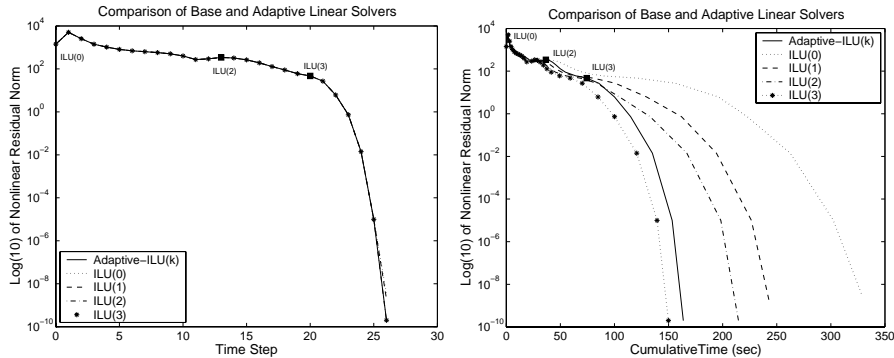


Fig. 7. Comparison of single-method linear solvers and an adaptive scheme. We plot the nonlinear convergence rate (in terms of residual norm) versus both time step (left-hand graph) and time (right-hand graph).

where $\|f_k\|$ is the function norm at the k -th time step.

3. The rate of increase in the number of iterations for linear system solutions in the previous time step,

$$R_I = \frac{n_k - n_{k-1}}{n_{k-1}},$$

where n_i is the number of linear iterations performed at the i -th time step. This metric can also be generalized over a number of time steps greater than one.

At each time step, we evaluate these three metrics to determine whether to switch to a different linear system solution method at the next time step. Although these experiments consider only adaptive ILU preconditioners, in principle any preconditioners and Krylov methods could be incorporated into an adaptive scheme. We use the following heuristic to select the level of ILU preconditioning used by the linear solver in the next time step.

- If $|R_N + R_L| \leq \lambda$, use the same solver in the next time step.
- If $R_N + R_L > \lambda$, select a less accurate solution method, e.g., decrease the fill level of ILU(k).
- If $R_N + R_L < -\lambda$ or $R_I > \beta$, select a more accurate solution method, e.g., increase the fill level of ILU(k).

The values of the user-specified parameters $\lambda = 0.2$, $w = 3$, and $\beta = 2$ were selected experimentally for another application and then used without modification in these experiments.

The graphs in Figure 7 compare several base preconditioner methods (ILU(0), ILU(1), ILU(2), ILU(3)), with an adaptive scheme that changes the level of fill of an incomplete factorization preconditioner based on the heuristic described

above; all cases used GMRES(30). We plot convergence rate (in terms of residual norm) versus both time step number (left-hand graph) and time (right-hand graph). The switching points for the adaptive scheme are marked on these graphs and indicate that the algorithm began using ILU(0), switched to ILU(2) at the thirteenth time step, and finally switched to ILU(3) at the twentieth time step. We see that the overall time to solution for the adaptive solver is within 10% of the best base method, ILU(3), and the adaptive solver significantly outperforms the other base methods, being twice as fast as the slowest, ILU(0). Perhaps more importantly, the adaptive solver enables the simulation to begin with a method that is relatively fast yet not so robust, and then to switch automatically to more powerful preconditioners when needed to maintain a balance in the linear and nonlinear convergence rates.

5 Conclusions and Future Work

Our preliminary results demonstrate the potential of multi-method composite and adaptive linear solvers for improving the performance of PDE-based simulations. We continue to develop and evaluate multi-method solvers for large-scale applications from several problem-domains. Additionally, multi-method schemes can also be constructed for several other computational problems in large-scale simulations, for example, discretization and Jacobian evaluation. We are also considering the effective instantiation of such schemes through advanced software frameworks that allow dynamic composition of software components [1, 17].

Acknowledgments

We gratefully acknowledge Barry Smith, who enhanced PETSc functionality to facilitate our work and participated in several discussions on the role and development of multi-method solvers.

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, 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 2.1.3, Argonne National Laboratory, 2002 (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. *Journal of Computational and applied Mathematics*, 74, (1996) pp. 91-110.
6. 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.
7. 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. Also under review, *Future Generation Computer Systems*,
8. 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).
9. T. S. Coffey, C. T. Kelley, and D. E. Keyes, Pseudo-Transient Continuation and Differential-Algebraic Equations, submitted to the open literature, 2002.
10. I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*, Clarendon Press, Oxford, 1986.
11. 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.
12. R. Freund, G. H. Golub, and N. Nachtigal, *Iterative Solution of Linear Systems*, Acta Numerica, Cambridge University Press, (1992) pp. 57-100.
13. B. Fryxell, K. Olson, P. Ricker, F. Timmes, M. Zingale, D. Lamb, P. MacNeice, R. Rosner, J. Truran, and H. Tufo, FLASH: an Adaptive-Mesh Hydrodynamics Code for Modeling Astrophysical Thermonuclear Flashes, *Astrophysical Journal Supplement*, 131 (2000) pp. 273-334, (see www.asci.uchicago.edu).
14. C. T. Kelley and D. E. Keyes, Convergence analysis of pseudo-transient continuation. *SIAM Journal on Numerical Analysis* 1998; 35:508-523.
15. L. McInnes, B. Norris, S. Bhowmick, and P. Raghavan, Adaptive Sparse Linear Solvers for Implicit CFD Using Newton-Krylov Algorithms, To appear in the Proceedings of the Second MIT Conference on Computational Fluid and Solid Mechanics, Massachusetts Institute of Technology, Boston, USA, June 17-20, 2003.
16. J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer-Verlag, New York, 1999.
17. 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.
18. X. Z. Tang, G. Y. Fu, S. C. Jardin, L. L. Lowe, W. Park, and H. R. Strauss, Resistive Magnetohydrodynamics Simulation of Fusion Plasmas, PPPL-3532, Princeton Plasma Physics Laboratory, 2001.