

Scaling Behavior of Linear Solvers on Large Linux Clusters

John Fetting¹, Wai-Yip Kwok¹, Faisal Saied¹

¹National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign

Abstract

As high performance Linux clusters enter mainstream supercomputing, it is important to understand how well these architecture scale for important kernels in computational science applications, such as linear solvers, eigenvalue solvers, multidimensional FFT's. This paper attempts to characterize the scalability of state of the art linear solvers on large Linux clusters. Our studies focus on two families of algorithms, Krylov subspace methods and multigrid methods. We include results up to 256 processors on a Linux cluster, with problem sizes up to 64 million unknowns.

1 Introduction

Large sparse linear systems of equations arise in many computational science and engineering disciplines. One common scenario occurs when implicit methods are used for time-dependent partial differential equations, and a nonlinear system of equations has to be solved at each time step. With a Newton-type approach, this in turn leads to a linear system at each Newton iteration. In such situations, the linear solver can account for a large part of the overall simulation time. Demonstrating the effectiveness of linear

solvers on large Linux clusters will help establish the usefulness of these platforms for applications in such diverse areas as CFD, supernova simulation, Bio-physics, earthquake engineering, and structural mechanics.

In this paper, we investigate the effectiveness of Linux clusters for state of the art software for parallel linear solvers such as PETSc and *hypre*. In particular, we study scaling to large processor counts, and large system sizes. The scalability of the algorithms, of the implementation, and of the cluster architecture are investigated. We focus on iterative Krylov subspace solvers like Conjugate Gradient, GMRES(k) and BiCGStab, coupled with parallel preconditioners, such as block Jacobi, domain decomposition methods (such as Additive Schwarz), and multigrid methods.

The structure of the remainder of this paper is as follows. We describe the Linux clusters at NCSA that were used in the tests reported here. The results are primarily for a Pentium III cluster, using up to 256 processors. Some data is included for an Itanium 1 Linux cluster, and for the SGI Origin 2000. We give a brief overview of the methods used in the scaling studies, namely Krylov subspace methods and multigrid methods, and of the software used in the study, PETSc and *hypre*. After describing the problems used in the symmetric and non-symmetric scaling tests, we present the results of our tests, for these two cases. In the process of investigating the scalability of linear solvers on Linux clusters, we include some comparisons of different methods and a comparison across architectures. We end with a summary of our findings, and future directions for investigation.

2 Architectures

The tests were performed on three different architectures. Two of the architectures are Linux based clusters and one is Irix based. The IA32 cluster, Platinum[1], is a distributed memory Linux cluster. Each node in the 484 node cluster is a dual processor IBM eServer x330 thin server. They are built with Intel Pentium III 1 GHz processors which have a 256 KB full-speed L2 cache. The peak performance is therefore 1 Gflop. The nodes are connected with a high speed Myrinet 2000 switched network to support the MPI programming model. The operating system installed on the nodes is RedHat Linux version 7.2 (2.4.9 kernel). The compilers used in the tests is the Intel C compiler, with the Intel Fortran77/90/95 and C++ compilers also available.

The IA64 cluster used in these tests, Titan[2], is also a distributed memory Linux cluster. It consists of 128 IBM IntelliStation Z Pro 6894 workstations, each of which contains two Intel Itanium-I 800 MHz processors with a 4 MB full-speed L3 cache. Each node has 2 GB of memory and is capable of 3.2 Gflops per processor. Like Platinum, Titan is connected via a switched Myrinet 2000 network which is the backbone for the MPI programming model for these systems. Titan has RedHat Linux 7.1 (2.4.16 kernel) installed, as well as the Intel Fortran77/90/95, C, C++ compilers.

In order to gain some perspective on the results found on the Linux clusters, the tests were also run on a SGI Origin 2000[3]. These shared memory architecture machines provides up to 128 MIPS R10000 250 MHz processors per machine, with each processor giving a peak performance of 500 Mflops. The programming model used in

these tests is MPI, but other parallel programming models are available on the Origins due to the shared memory architecture. The Origins are running SGI IRIX 6.5, and use the SGI Fortran77/90/95, C, C++ compilers.

3 Algorithms for Solving Linear Systems

In this paper we are interested in methods for solving large sparse linear systems that arise from the discretization of partial differential equations, denoted by

$$Ax = b,$$

where A is an $N \times N$ matrix. In this section we give a brief overview of the methods we have used in our scaling studies.

One of the leading families of methods for linear systems are iterative solvers known as Krylov subspace methods [6], [8], [9], [12]. These methods are characterized by the subspaces in which the iterates lie. The j -th Krylov subspace for a given matrix A and vector r_0 is

$$K_j(A, r_0) = \{r_0, Ar_0, A^2r_0, \dots, A^{j-1}r_0\}.$$

The j -th iterate of a Krylov subspace method lies in the shifted affine subspace

$$x_j \in x_0 + K_j(A, r_0).$$

Different Krylov subspace methods differ in the criteria they use in selecting a vector in the subspace. The Conjugate Gradient method picks x_j to be the vector that minimizes the A -norm of the error over the Krylov subspace. This method can only be used for problems where the matrix is symmetric and positive definite.

A number of Krylov subspace methods have been developed for the case when the matrix is not symmetric, positive definite. The GMRES method [11] picks x_j to be the vector that minimizes the two norm of the residual over the Krylov subspace. In its original form GMRES requires one additional vector of storage for each iteration, for the Arnoldi process, that applies Gram-Schmidt orthogonalization to the Krylov basis. In practice, GMRES is usually implemented in its restarted form, GMRES(k), with a fixed number of vectors stored. Another popular Krylov subspace method for non-symmetric problems is the BiCGStab (stabilized BiConjugate Gradient) method [13].

The main operations in a Krylov subspace method are (i) matrix-vector products (ii) dot products and norm computations, and (iii) vector updates.

The convergence rate of a Krylov subspace method can be improved by using the method in conjunction with a preconditioner, which we denote by M . The iterative method is applied to the preconditioned system

$$M^{-1}Ax = M^{-1}b.$$

Applying the preconditioner to a vector r is symbolically denoted by $z = M^{-1}r$. In selecting a preconditioner one needs to balance the cost of applying the preconditioner and the improvement in convergence obtained by preconditioning (which is related to how well M approximates A). Some of the simplest preconditioners are Jacobi and block Jacobi. Other preconditioners are ILU (incomplete LU factorization) and approaches based on domain decomposition, such as Additive Schwartz.

Another method that can be used on its own, or as a preconditioner in a Krylov subspace method, is the multigrid method [7]. In its usual (geometric) form, multigrid works with a discretization of the PDE on a hierarchy of progressively coarser grids. The idea is to apply a few iterations of a relaxation method at a given level to get an approximation whose error is smooth and can be represented on a coarser grid. Then the residual of this approximation is restricted to the next coarser grid, and the process repeated until the coarsest grid is reached. Here a coarse grid solve is performed, and the solution (the coarse grid correction) is interpolated up to the finer grid. After adding the coarse grid solution to the existing approximation, a few more iterations of the smoother are applied.

The user of a multigrid code needs to make a number of choices to get a specific form of the method. These choices include the smoother, the number of pre- and post-smoothing iterations, the restriction and interpolation operators, coarse grid matrices for each grid, the coarse grid solver, and the cycling scheme. The preconditioned Conjugate gradient method requires the preconditioning matrix to be symmetric. Since the multigrid iteration matrix is not symmetric (unless special choices are made to ensure symmetry), we use non-symmetric Krylov subspace methods in conjunction with multigrid preconditioners.

4 Software Packages

Two of the leading packages which provide parallel routines for solving large sparse linear systems are PETSc[4], produced at Argonne National Laboratory, and *hypre*[10], produced at Lawrence Livermore National Laboratory. Both of these packages were designed with scalability on a message passing architecture in mind, and so they are prime candidates for study and use on Linux clusters. Both of these packages use MPI and is readily available for Linux as well as other platforms.

4.1 PETSc

PETSc, the portable, extensible toolkit for scientific computation, provides library routines for solving large sparse linear systems through the Scalable Linear Equations Solvers (SLES) component[5]. This component allows the user to easily solve linear systems that are set up in a PETSc format. These formats include a standard sparse storage format (AIJ), a block diagonal format, and an interface to enter information about a structured grid. Once the system is set up in PETSc format, there is a call to `SLESSolve()` which solves the system.

Some of the options to the call to `SLESSolve()` allow you to select from a broad range of Krylov Subspace Methods to use, in conjunction with a preconditioner. The available Krylov Subspace Methods include GMRES(k) and BiCGStab. The preconditioners available from within PETSc include Jacobi, Block Jacobi, Additive Schwarz, and Multigrid. Multigrid must be used on a structured grid from within PETSc, but it is also available for unstructured grids through an interface to *hypr*.

One of the strengths of PETSc is the degree of customization. There are a plethora of options available for solving linear systems, which gives a lot of flexibility within the package to adjust the way you are solving a particular system based on knowledge of the system. There are a number of monitors and logging facilities built in to PETSc to help you determine whether or not PETSc is doing a good job of solving your system. The results shown in this paper utilize PETSc's ability to monitor itself and give detailed breakdowns of function calls and timings.

4.2 *hypr*

Designed with multigrid methods in mind, *hypr* is a package that provides many high performance preconditioners. It provides another option in scalable software for solving large, sparse linear systems of equations on massively parallel computers. *hypr* has an interface for structured-grid based systems, semi-structured-grid based systems, finite element systems, and linear-algebraic systems. This paper tests the scalability of the structured-grid system interface.

Like PETSc, *Hypr* has the more popular Krylov Subspace Methods built-in. These include GMRES and Conjugate Gradient. Additionally, a multigrid method is available as a solver and a preconditioner. Other preconditioners available include Jacobi, diagonal scaling, BoomerAMG (*hypr*'s algebraic multigrid), and ParaSAILS (*hypr*'s sparse approximate inverse).

5 Test Problems

The scaling studies reported in this paper were done for one symmetric positive definite problem, and a non-symmetric one.

5.1 The Poisson Model Problem

The symmetric linear system is derived by a standard 7-point discretization of the Poisson equation, with homogeneous Dirichlet boundary conditions, using a uniform grid[8].

$$-\nabla^2 u(x) = -\frac{\partial^2 u(x)}{\partial x^2} - \frac{\partial^2 u(x)}{\partial y^2} - \frac{\partial^2 u(x)}{\partial z^2} = f(x) \quad \text{in } \Omega = (0,1)^3$$

$$u(x) = 0 \quad \text{on } \partial\Omega.$$

The total number of unknowns is $N = nx \times ny \times nz$.

The sparse matrix resulting from this discretization can be stored in a standard compressed row sparse storage format (the AIJ format), or alternatively, by using the structured grid infrastructure in PETSc or Hypre.

5.2 The Convection-Diffusion Equation

A simple modification of the Poisson equation, namely the addition of a first derivative term in the p.d.e., leads to a non-symmetric matrix, and can be used to test the non-symmetric solver capabilities of the software. The parameter α controls the degree of non-symmetry in this test problem.

$$-\nabla^2 u(x) + \alpha \left(\frac{\partial u(x)}{\partial x} + \frac{\partial u(x)}{\partial y} + \frac{\partial u(x)}{\partial z} \right) = f(x), \quad \text{in } \Omega.$$

The sparsity structure of resulting matrix is the same as in the Poisson problem. In particular, we can use either the AIJ format or the structured grid features of PETSc and Hypre.

6 Results

The results are divided into two sections, first the results for the symmetric model problem, then the results for the non-symmetric model problem. There is also a distinction between results from PETSc and the results from *hypre*. Note that some of the scaling studies were done with a fixed total problem size as the number of processors grew, and some were done with a fixed problem size per processor. It will be made clear in each case which scaling study is being represented. All of the tests used one processor per node on the linux clusters, allowing for full utilization of the memory bandwidth and the processor interconnect by that cpu.

6.1 Symmetric Problem

Previously it was mentioned that PETSc has a number of options for the multigrid preconditioner. Figure 1 shows the difference in scalability between two options for the coarse grid solve when using multigrid as a preconditioner to GMRES(30) on a structured grid on the IA32 cluster. Keeping the number of unknowns per processor fixed at $\frac{1}{4}$ million, we would see a horizontal line for ideal scaling. The orange plot shows a direct method, LU factorization, for the coarse grid solve. The blue plot shows an iterative method using the preconditioner only which is set to Block Jacobi. LU does better than the iterative method until 128 processors, when the time for the coarse grid solve blows up to around 700 seconds with LU. This is most likely the result of the poor scaling of the direct solver. Multigrid methods can tolerate an approximate (and cheaper) coarse grid solver and so we will use the block Jacobi preconditioner on the coarse grid for all other PETSc results presented here.

Doing the same scaling study on IA32 using the more standard preconditioning methods implemented in PETSc on an unstructured grid, it becomes more evident how important a good preconditioner is. Figure 2 shows that Jacobi doesn't work very well as a preconditioner on the model problem, but Block Jacobi does. CG preconditioned with Block Jacobi is the best choice from these options.

Figure 3 shows the difference between GMRES preconditioned with multigrid with Block Jacobi on the coarse grid, and CG preconditioned with Block Jacobi. Both are implemented in PETSc on IA32. Indeed, the multigrid preconditioned method scales much better than the others. This demonstrates the better algorithmic scalability of multigrid.

Figure 4 shows *hypr* solving a structured grid problem on two different architectures. As before, scalability is tested using $\frac{1}{4}$ million unknowns per processor up to 256 processors. The architectures show similar single-processor performance, although the theoretical peak flop rate of the Linux cluster is higher than that of the SGI Origin2000. The slope of the curves is small for the Linux cluster, meaning that it scales well. However, the wall clock time goes up more for the SGI Origin2000 as number of processors increases. This is probably due to the better network connection in the Linux clusters. Comparing the two numerical solvers, GMRES with multi-grid preconditioners is performing better than multi-grid alone. The results from the IA64 cluster have been omitted from the plotted results. The reason for this is that the IA64 clusters are not presently showing improved times when compared to the IA32 clusters. We are investigating what might be causing this.

6.2 Non-Symmetric Problem

Figure 5 shows what happens to scalability when the total problem size is fixed and the number of processors is varied using PETSc on IA32. Ideal scaling would look like $\frac{1}{P}$. Total problem size is fixed at 8 million unknowns and P ranges from 1 to 256. Again, BiCGStab preconditioned with Block Jacobi is the best choice.

Figure 6 shows a cross-platform comparison between the IA32 cluster and the Origin2000 with fixed total problem size scaling. The much smaller theoretical peak flop rate of the Origin2000 is most evident in the single processor case and is less pronounced as P increases.

Figure 7 is the non-symmetric version of figure 3. Since this is a non-symmetric problem, the best non-multigrid method is now BiCGStab preconditioned with block Jacobi. Again it is evident that the multigrid preconditioned method scales far better than the others because of the superior algorithmic scaling of multigrid.

In Figure 8 the Linux cluster shows better single-processor performance compared with SGI Origin2000 (about 15 to 20% faster). The Linux cluster scales very well as the number of processors increases, while the SGI Origin2000 does not scale as well. Comparing the two numerical techniques, GMRES with multi-grid preconditioner show better single-processor performance. However, multigrid scales better. Therefore as number of processors increases, multigrid becomes more effective.

Table 1 demonstrates the scalability of different components of the solver using information from PETSc's built-in instrumentation. The tests were run on a non-

symmetric problem using BiCGStab preconditioned with block Jacobi on IA32. Using the log files generated by PETSc, we have tabulated the percent of time spent in subset of the function calls made during a linear solve. PETSc inserts a barrier call in the dot-product function when profiling is turned on. The VecDotBarrier times reflect the synchronization delay in the dot-products. The MatMult scales well with the total solve time, but the VecDotBarrier does not. This indicates that as the number of processors increases, the VecDotBarrier is increasingly becoming the bottleneck for good performance and scaling. On the other hand, the MatSolve portion of the solve generally decreases as the number of processors grows due to the almost embarrassingly parallel nature of the solves on the individual blocks of the block Jacobi preconditioner.

7 Conclusions

The results presented in this paper show that using state of the art algorithms and software for large sparse linear systems, it is possible to achieve very good scaling on Linux clusters. In particular, on 256 processors of a Pentium 3 Linux cluster at NCSA, we were able to solve a linear system with 64 million unknowns in under 23 seconds. Our scaled speedup results show that the 400^3 problem on 256 processors took only 1.3 times longer than the 64^3 problem on one processor in the symmetric case, and 1.25 times longer for the non-symmetric case.

In the future we will extend the current studies to include Itanium 2 based Linux clusters. We plan to use matrices from an application code for supernova simulations. We are exploring the possibility of using the *hypr* solvers as preconditioners through PETSc. This will include the algebraic multigrid and sparse approximate inverse preconditioners in Hypr.

Acknowledgements

This work was funded in part by a Department on Energy grant DE FG0201ER41204. The computations were performed at the National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign, which is funded through the PACI Program at the National Science Foundation.

References

- [1] NCSA IA32 cluster home page
<http://www.ncsa.uiuc.edu/UserInfo/Resources/Hardware/IA32LinuxCluster/>, 2002.
- [2] NCSA IA64 cluster home page
<http://www.ncsa.uiuc.edu/UserInfo/Resources/Hardware/IA64LinuxCluster/>, 2002.

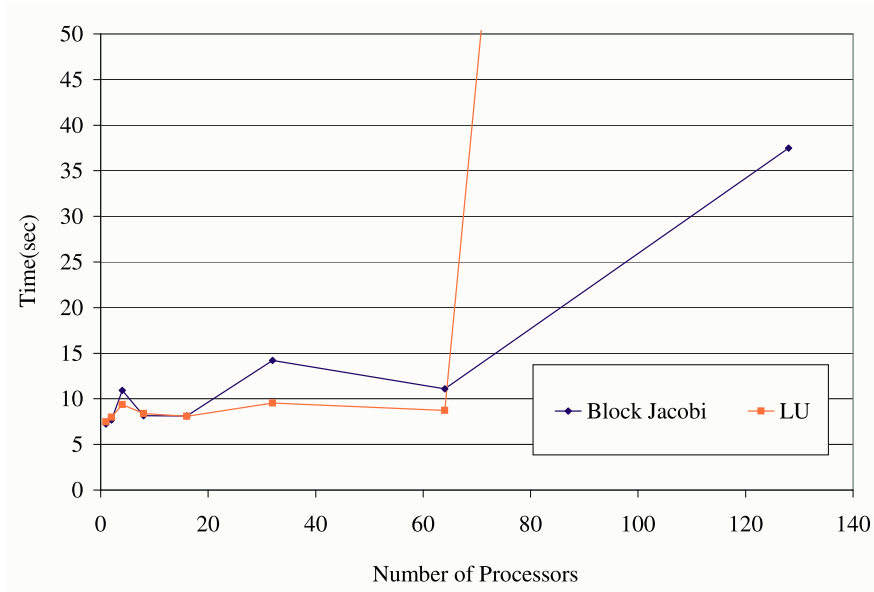


Figure 1: Fixed problem size ($\frac{1}{4}$ million unknowns) per processor with a symmetric problem solved by PETSc with two different coarse grid solvers.

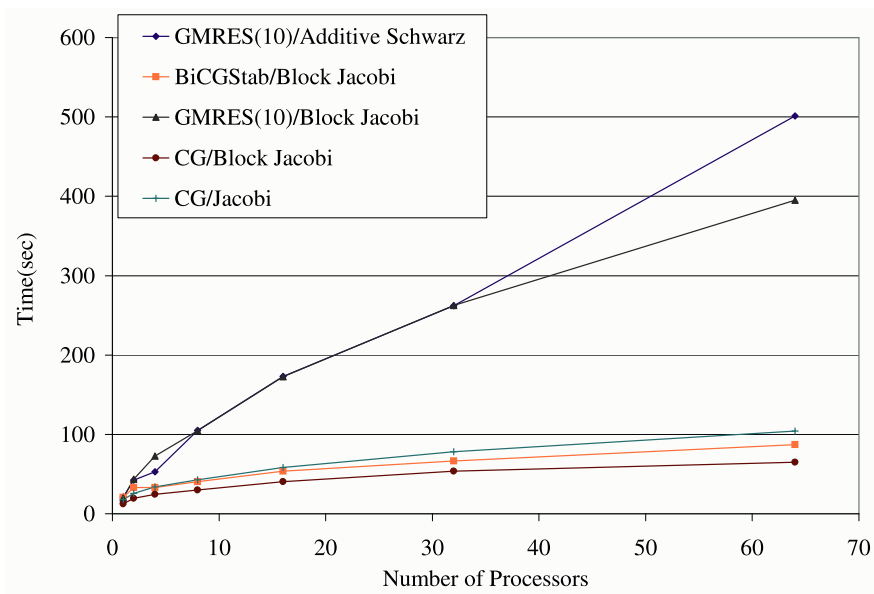


Figure 2: Fixed problem size ($\frac{1}{4}$ million unknowns) per processor with a symmetric problem solved by PETSc's standard preconditioned Krylov Subspace methods.

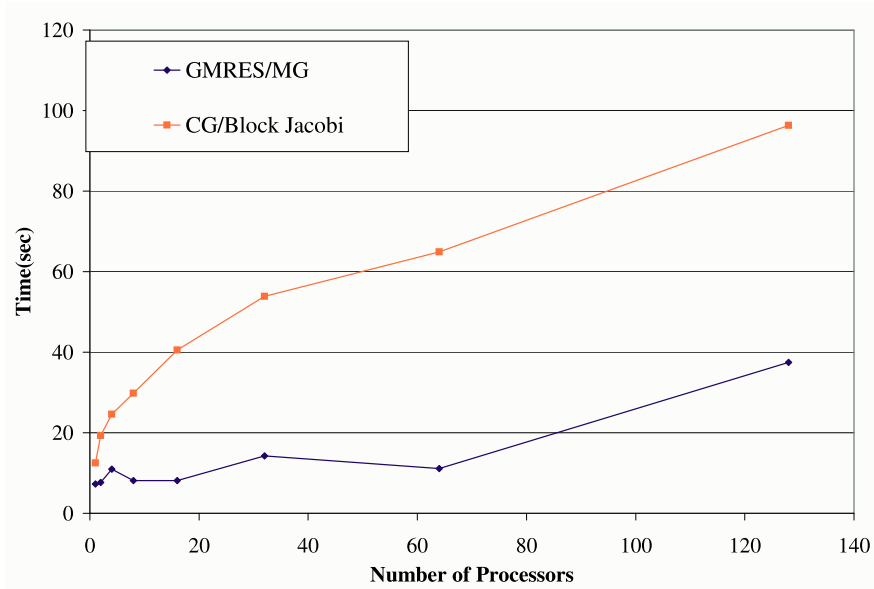


Figure 3: Fixed problem size ($\frac{1}{4}$ million unknowns) per processor with a symmetric problem solved by PETSc, comparing the multigrid preconditioner with a standard one.

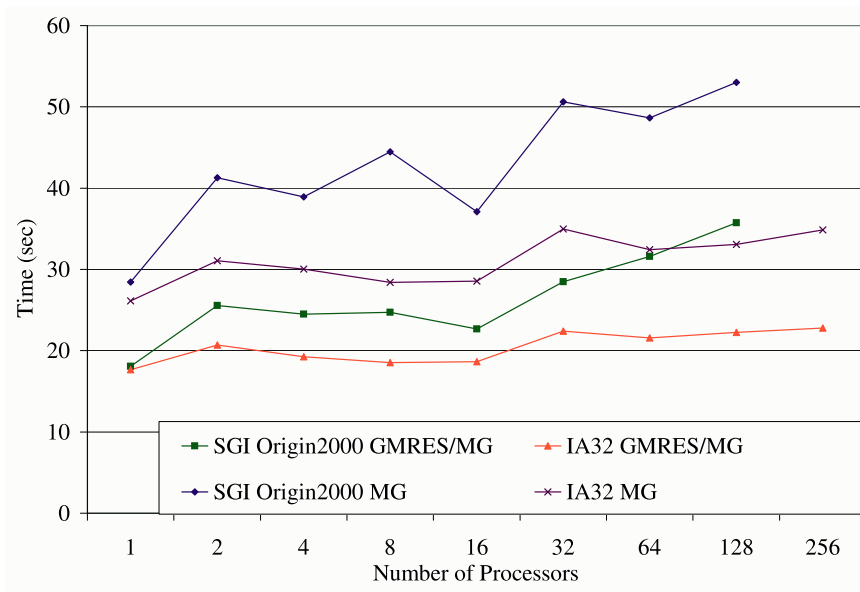


Figure 4: Cross-platform comparison with *hypr*'s GMRES/MG and multigrid alone on a symmetric problem with fixed problem size ($\frac{1}{4}$ million unknowns) per processor.

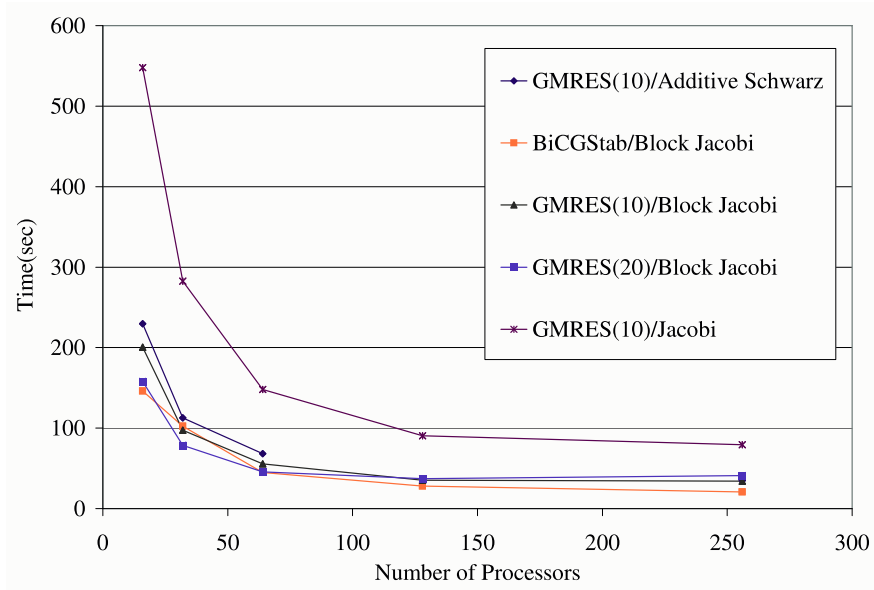


Figure 5: Fixed total problem size (8 million unknowns) on a non-symmetric problem solved by PETSc using standard preconditioned Krylov Subspace methods.

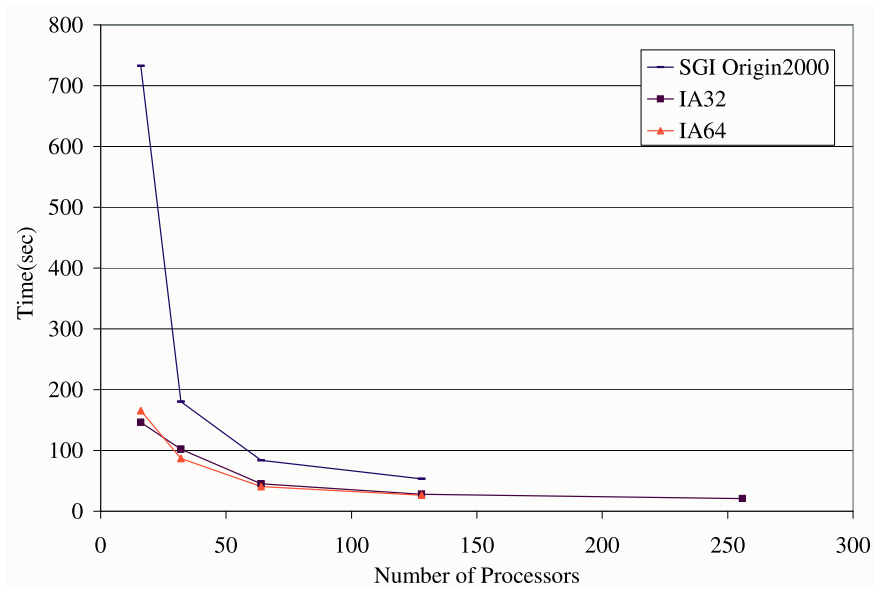


Figure 6: Fixed total problem size (8 million unknowns) on a non-symmetric problem solved by PETSc. Cross-platform comparison of BiCGStab/block Jacobi.

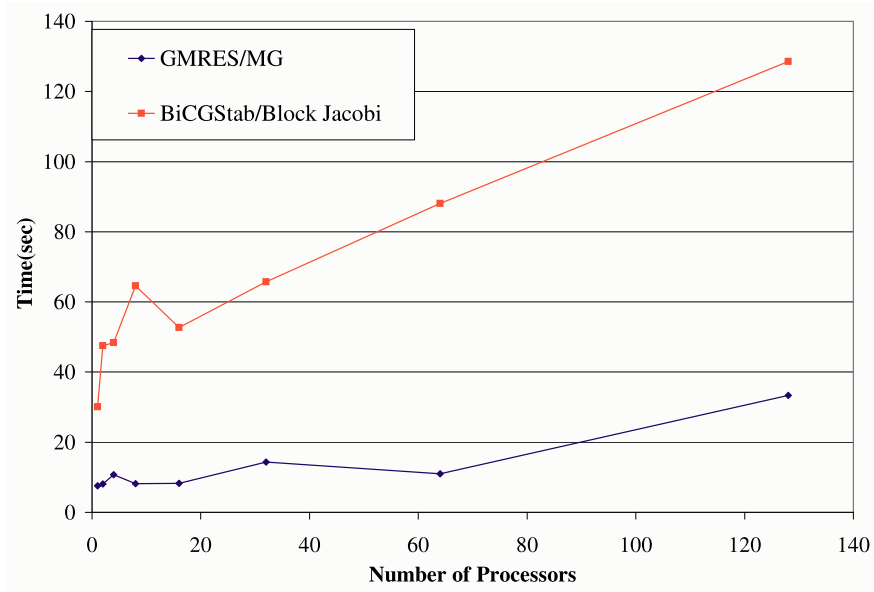


Figure 7: Fixed problem size ($\frac{1}{4}$ million unknowns) per processor with a non-symmetric problem solved by PETSc, comparing the multigrid preconditioner with a standard one.

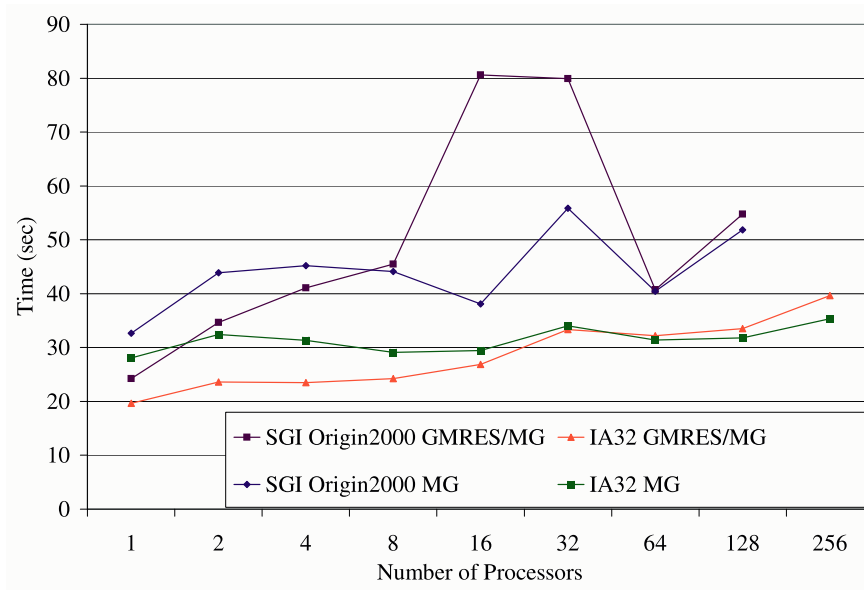


Figure 8: Cross-platform comparison with *hypr*'s GMRES/MG and multigrid alone on a non-symmetric problem with fixed problem size ($\frac{1}{4}$ million unknowns) per processor.

- [3] NCSA SGI Origin2000 home page
<http://www.ncsa.uiuc.edu/UserInfo/Resources/Hardware/Origin2000/>, 2002.
- [4] Satish Balay, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Lois Curfman McInnes, and Barry F. Smith. PETSc home page.
<http://www.mcs.anl.gov/petsc>, 2001.
- [5] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. PETSc users manual. Technical Report ANL-95/11 - Revision 2.1.3, Argonne National Laboratory, 2002.
- [6] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, V. Pozo, C. Romine, and H. van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, Philadelphia, 1994.
- [7] A. Brandt. Multilevel adaptive solutions to boundary-value problems. *Math. Comp.*, 31:311–329, 1977.
- [8] J. W. Demmel. *Applied Numerical Linear Algebra*. SIAM, Philadelphia, 1997.
- [9] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. van der Vorst. *Numerical Linear Algebra for High-Performance Computers*. SIAM, Philadelphia, 1998.
- [10] hypre: High performance preconditioners. <http://www.llnl.gov/CASC/hypre/>.
- [11] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7(3):856–869, 1986.
- [12] L. N. Trefethen and D. Bau. *Numerical Linear Algebra*. SIAM, Philadelphia, 1997.
- [13] H. A. van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of non-symmetric problems. *SIAM J. Sci. Stat. Comput.*, 13:631–645, 1992.

	16 processors	32 processors	64 processors	128 processors
VecDotBarrier	16.13	19.46	26.33	32.57
VecDot	4.15	4.82	7.39	12.09
MatMult	50.9	52.63	56.0	50.64
VecAXPY	5.3	3.64	3.22	2.82
MatSolve	26.7	28.36	21.95	15.9

Table 1: Percent of total solve time spent in selected function calls during PETSc solve using BiCGStab with block Jacobi preconditioning on a non-symmetric problem with fixed problem size (8 million unknowns) on IA32.