NAS Technical Report; NAS-07-007
September 2007

Testing Parallel Linear Iterative Solvers for Finite Element Groundwater Flow Problems

Fred T. Tracy and Thomas C. Oppe
U.S. Army Engineer Research and Development Center Major Shared Resource Center (ERDC MSRC)
Vicksburg, MS
{Fred.T.Tracy, Thomas.C.Oppe}@erdc.usace.army.mil

Sharad Gavali
NASA Ames Research Center, Moffett Field, CA
gavali@nas.nasa.gov

Abstract

*The modeling of groundwater flow using three-dimensional finite element discretizations creates a need to solve large systems of sparse linear equations (***Ax** = **b***) at each of several nonlinear iterations. These linear systems can be difficult to solve because of the ill-conditioning of the matrix* **A** *resulting from the widely varying magnitudes of its coefficients. Because the meshes may contain millions of nodes, iterative solvers are typically used to perform the* **Ax** = **b** *solution. Since 80 percent or more of the computational time is spent in the iterative solver part of the computer program, choosing the most efficient solver for each application can dramatically reduce the total solution time. This paper tests 12 Krylov subspace parallel linear iterative solvers with 5 preconditioners (60 scenarios) on linear systems of equations resulting from a finite element study of remediation of a military site using pump-and-treat technology. Both symmetric, positive-definite matrices, resulting from a Picard linearization of the nonlinear equations for the steady-state case, and nonsymmetric matrices, arising from a Newton linearization of the nonlinear equations of a transient case, are studied. The Portable, Extensible Toolkit for Scientific Computation (PETSc) library was used in this study on the Engineer Research and Development Center Major Shared Resource Center SGI O3K and Cray XT3 computers. Matrix data corresponding to each subdomain in a parallel groundwater finite element program were first written to a file in a compressed sparse column format. A separate program was then written in FORTRAN to read these data, renumber the nodes, call the PETSc routines to load* **A** *and* **b** *and then solve for* **x***, and finally compute error norms. Solver time, iteration count, 2-norm and ∞-norm of the residual after convergence, weak speedup, and strong speedup are tabulated in this paper for the different scenarios and then analyzed.*

1. Introduction

The modeling of groundwater flow using the finite element method with three-dimensional (3-D) meshes creates a need to solve large systems of linear equations,

$$\mathbf{Ax} = \mathbf{b} \qquad (1)$$

at each of several nonlinear iterations. Here, **A** is the coefficient matrix, **b** is the known right-hand-side vector, and **x** is the unknown vector to be computed. Widely varying material properties of the media (e.g., hydraulic conductivity of sand and clay) and the presence of unsaturated flow can give rise to ill-conditioned matrices having coefficients that vary in size by several orders of magnitude. Because the meshes may contain millions of nodes, iterative solvers are often used to solve Equation 1. Since 80 percent or more of the computer time is spent in the iterative solver part of the computer program, choosing the most efficient solver for each application can dramatically reduce the total solution time. The purpose of this work is to test several iterative parallel linear solvers to help determine the best one for groundwater flow applications. Because of the nature of the matrices, the findings may be applicable to other application areas as well.

2. Test Problem

The test problem consists of a finite element model of a pump-and-treat system for cleaning up a military site. Figure 2 shows a top view of the entire mesh. Figure 3 shows a magnified portion of the mesh showing wells and trenches. Figure 4 shows a further magnification of the mesh surrounding two wells. Finally, Figure 5 shows a lateral view of the mesh showing the refinement chosen for the various soil layers. The original mesh was discretized using 102,996 nodes and 187,902 elements, while a 2-fold refinement utilized 197,409 nodes and 375,804 elements, and an 8-fold refinement utilized 763,887 nodes and 1,503,216 elements. Two linear systems from this test problem were tested: (1) the steady-state run at the tenth nonlinear iteration using a Picard linearization, producing a symmetric, positive-definite (SPD) linear system and (2) a transient run at the tenth nonlinear iteration of the first time-step using a Newton linearization, producing a nonsymmetric linear system.

3. Testing Iterative Solvers Using PETSc

This paper tests 12 Krylov subspace parallel linear iterative solvers[1,2,4,6] with 5 preconditioners (60 scenarios) on the two linear systems described above. The Portable, Extensible Toolkit for Scientific Computation (PETSc) library[5] was used in this study on the Engineer Research and Development Center Major Shared Resource Center SGI O3K and Cray XT3 computers. The solvers are

1. Conjugate Gradient (CG)
2. Generalized Minimum Residual (GMRES)
3. Biconjugate Gradient (BiCG)
4. Biconjugate Gradient Stabilized (BiCGSTAB)
5. Conjugate Gradient Squared (CGS)
6. Transpose-Free Quasi-Minimal Residual, version 1 (TFQMR1)
7. Transpose-Free Quasi-Minimal Residual, version 2 (TFQMR2)
8. Conjugate Residual (CR)
9. Flexible GMRES (FGMRES)
10. Minimum Residual (MINRES)
11. Symmetric LQ (SYMMLQ)
12. Biconjugate Gradient Stabilized, degree k (BiCGSTAB(k))

The preconditioners are

1. None
2. Jacobi
3. Block Jacobi (Bjacobi)
4. Additive Swartz method (ASM)
5. successive overrelaxation (SOR)

Figure 1 shows a generic parallel version of the Conjugate Gradient solver algorithm for a finite element program with each processing element (PE) assigned to a portion of the mesh. The preconditioning matrix **K** is chosen so that it approximates **A** in some sense and because the auxiliary linear system

$$\mathbf{Kz} = \mathbf{r} \tag{2}$$

is much easier to solve than the original linear system and can be solved efficiently on parallel architectures. Ghost nodes for the vector **p** are updated prior to calculating the vector $\mathbf{q} = \mathbf{Ap}$, and parallel reduction operations are required to calculate the inner products $\mathbf{b}^T\mathbf{b}$, $\mathbf{z}^T\mathbf{r}$, $\mathbf{p}^T\mathbf{q}$, and $\mathbf{z}^T\mathbf{r}$. $\mathbf{x}_0$ is an initial guess to the solution **x**.

```
□ = 0; p = 0
r = b - A * $x_0$; nmax = 20000
□ = $10^{-15}$; eps = □ * sqrt($b^Tb$)
! || reduction needed for $b^Tb$
n = 0
do
  n = n + 1
! Apply preconditioner
  Solve K * z = r for z
  □  = $z^Tr$
! || reduction needed for $z^Tr$
  if (n > 1) □ = □ / $□_{sav}$
  p = z + □ * p
! Ghost node update needed for p
! || reduction needed for $p^Tq$
  q = A * p; □ = □ / $p^Tq$
  x = x + □ * p; r = r - □ * q
  $□_{sav}$ = □
! || reduction needed for $r^Tr$
  if (n > nmax .or.
       sqrt($r^Tr$) < eps) exit
end do
```

Figure 1: Parallel Conjugate Gradient algorithm

3.1. Saved Data

For each subdomain, the following data were written to a file from a parallel groundwater finite element program:

1. Number of global nodes, number of "owned" nodes (i.e., subdomain nodes), number of "local" nodes, which is the union of owned nodes and "ghost" nodes (i.e., nodes in other subdomains that are connected to an owned node), number of compressed columns, and number of PEs.
2. A one-dimensional (1-D) array containing the global node numbers for the local nodes.
3. A two-dimensional (2-D) array in compressed column format containing the local node numbers corresponding to the coefficients of **A** for the owned rows of **A**. Zeroes are used to pad the array to simplify the reading and writing of these data.

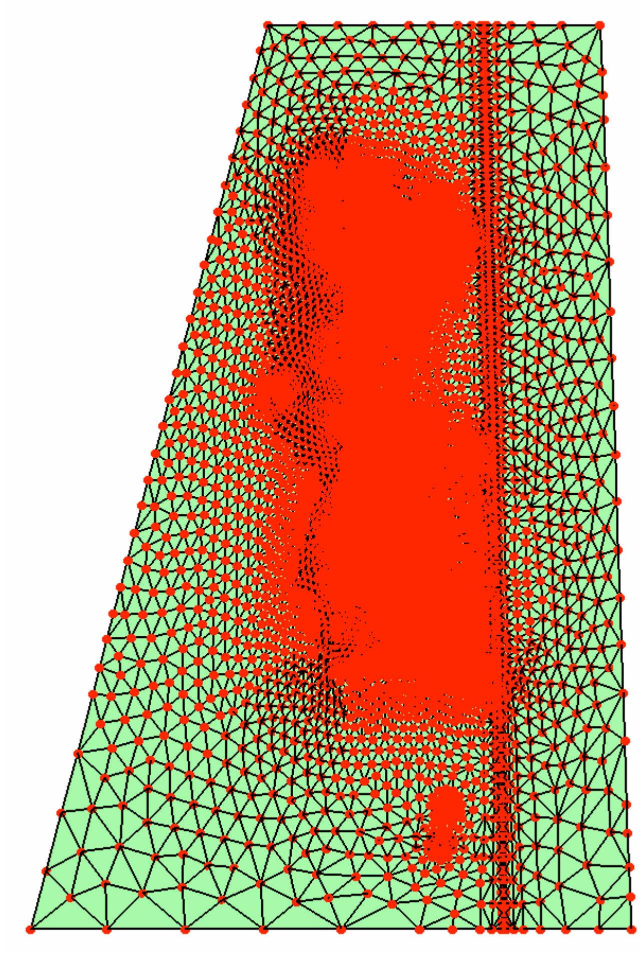

Figure 2: Top view of mesh

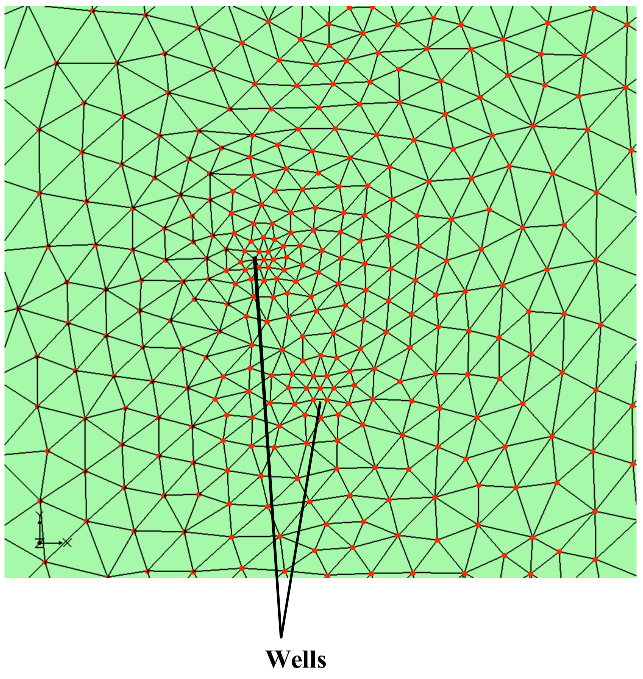

Figure 4: Further magnification showing fine resolution of the mesh for modeling wells

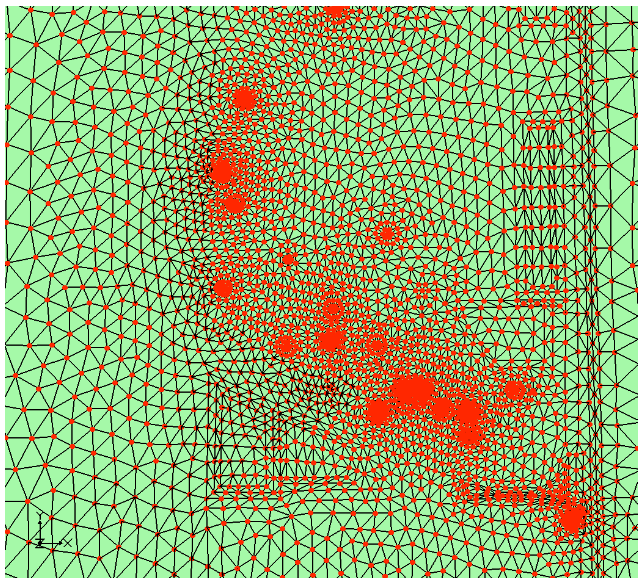

Figure 3: Magnified view of a portion of the mesh

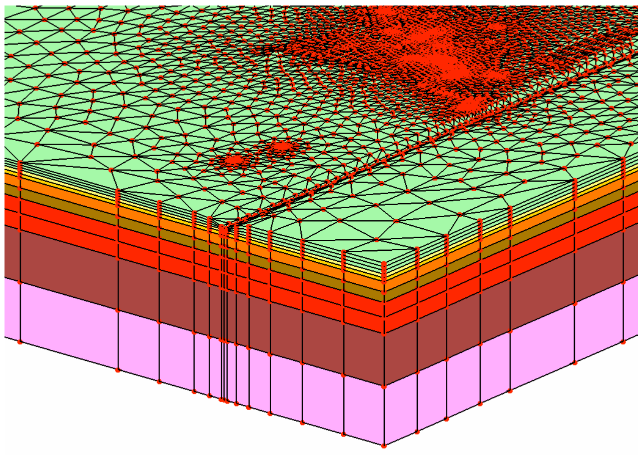

Figure 5: Side view showing strata layers

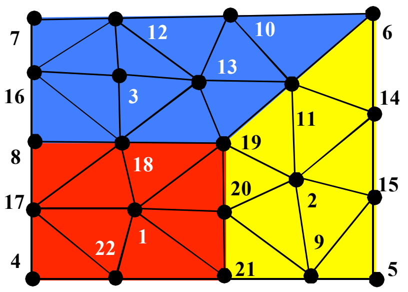

Figure 6: Small finite element mesh

4. A 2-D array in compressed column format containing the coefficients of **A** for the owned rows of **A**. Zeroes are used to pad the array.
5. A 1-D array containing the owned portion of **b**.
6. A 1-D array containing the owned portion of the solution vector **x** obtained independently.

A separate program was then written in FORTRAN to read these data, renumber the nodes, call the PETSc routines to load **A** and **b** and then solve for **x**, and finally compute error norms. Solver time, iteration count, 2-norm and ∞-norm of the residual after convergence, weak speedup, and strong speedup were tabulated for the different scenarios and then analyzed.

3.2. Renumbering the Nodes

ParMETIS[3] was used to compute the original partitioning of the mesh. Unfortunately, the resulting global numbering of the nodes was very inefficient when input directly into PETSc, which used a block partitioning of the matrix **A** by rows. To illustrate the difficulty, Figure 6 shows a sample finite element mesh containing 22 nodes and partitioned into three PEs. The node assignment is

```
PE 0    4 22 21 17  1 20  8 18 19
PE 1    9  2 11  5 15 14  6
PE 2   16  3 13  7 12 10
```

If the same number of nodes per PE is maintained, the PETSc partitioning is

```
PE 0    1  2  3  4  5  6  7  8  9
PE 1   10 11 12 13 14 15 16
PE 2   17 18 19 20 21 22
```

With the ParMETIS partitioning, node 1 has no ghost nodes; node 2 has ghost nodes 19 and 20; node 3 has ghost nodes 18; etc. However, with the PETSc partitioning, node 1 has ghost nodes 17, 18, 19, 20, 21, and 22; node 2 has ghost nodes 9, 11, 14, 15, 19, and 20; node 3 has ghost nodes 12, 13, 16, and 18; etc. To eliminate the communication cost of the additional ghost nodes, the global nodes were renumbered consecutively within each ParMETIS partition. `npetsc` is a mapping vector from the original global node numbering to the new numbering.

3.3. PETSc FORTRAN Code

To see an example of how the input data are used with PETSc, consider the code to load the array `a` with values from the matrix **A** (Figure 7). Definitions of the major variables are as follows:

`nown`	number of owned nodes
`ncol`	number of compressed columns
`ai`	original **A** matrix
`a`	PETSc version of the **A** matrix
`jloc`	local node number from the local row i and local compressed column j
`ii`	new global row number in zero-based numbering system
`jj`	new global column number in zero-based numbering system

```
do i = 1, nown
 ii = npetsc(i) - 1
 do j = 1, ncol
  jloc = id(i, j)
  if (jloc .ne. 0) then
   jj = npetsc(jloc) - 1
   v = ai(i, j)
   call MatSetValues (a, 1, ii, 1, &
        jj, INSERT_VALUES, ierr)
  end if
 end do
end do

call MatAssemblyBegin (a, &
     MAT_FINAL_ASSEMBLY, ierr)
call MatAssemblyEnd (a, &
     MAT_FINAL_ASSEMBLY, ierr)
```

Figure 7: Loading A into PETSc

The **b** vector is loaded in a similar fashion. After options are set, a call to `KSPSolve` completes the solution. Table 1 shows times for the O3K and XT3 for loading the data into PETSc after allocating sufficient memory for the arrays.

PEs	Nodes	Elements	Time (sec)
8	102996	187902	0.29
16	102996	187902	0.15
32	197409	375804	0.29
8	102996	187902	0.08
16	102996	187902	0.04
32	197409	375804	0.06
64	763887	1503216	0.17

Table 1: Load times of the PETSc data for the O3K (white) and XT3 (shaded) for the SPD matrix

4. Test Results

Table 2 shows results for the 60 scenarios for the SPD matrix for the original mesh of 102,996 nodes and 187,902 elements using 8 PEs on the O3K and XT3. In all the runs, the convergence criterion of

CG

PC	2-Norm $\times 10^{-9}$	∞-Norm $\times 10^{-10}$	Iterations	Time (sec)
None	56.4	23.1	7096	24.47
··	··	··	··	13.14
Jacobi	18.3	8.44	768	2.78
··	··	··	··	1.50
Bjacobi	9.91	5.16	224	1.94
··	··	··	··	1.03
ASM	-	-	-	-
··	-	-	-	-
SOR	11.4	5.89	306	2.06
··	··	··	··	1.59

GMRES

PC	2-Norm $\times 10^{-9}$	∞-Norm $\times 10^{-10}$	Iterations	Time (sec)
None	2.00	1.18	215247	1478.09
··	1.99	1.75	236216	645.88
Jacobi	1.72	1.16	2639	18.81
··	··	0.673	2419	6.64
Bjacobi	1.72	1.08	586	7.97
··	1.71	1.16	587	3.22
ASM	1.69	1.64	514	9.22
··	1.70	0.764	515	3.30
SOR	1.62	0.946	808	8.66
··	1.63	··	809	5.02

BiCG

PC	2-Norm $\times 10^{-9}$	∞-Norm $\times 10^{-10}$	Iterations	Time (sec)
None	58.7	29.8	7426	56.06
··	58.3	25.0	7466	25.81
Jacobi	18.2	8.66	769	6.06
··	··	9.90	··	2.72
Bjacobi	9.82	4.61	224	4.22
··	9.97	6.23	··	2.00
ASM	9.86	4.56	220	5.22
··	9.82	6.69	··	2.35
SOR	-	-	-	-
··	-	-	-	-

BiCGSTAB

PC	2-Norm $\times 10^{-9}$	∞-Norm $\times 10^{-10}$	Iterations	Time (sec)
None	83.3	33.7	8140	55.84
··	··	··	··	29.63
Jacobi	52.9	32.6	509	3.69
··	··	··	··	1.87
Bjacobi	35.2	16.2	154	2.96
··	··	··	··	1.39
ASM	23.6	9.80	141	3.53
··	··	··	··	1.53
SOR	23.9	11.8	214	2.91
··	··	··	··	2.19

CGS

PC	2-Norm $\times 10^{-9}$	∞-Norm $\times 10^{-10}$	Iterations	Time (sec)
None	-	-	-	-
··	-	-	-	-
Jacobi	2699.	24700.	516	3.75
··	··	··	··	1.90
Bjacobi	-	-	-	-
··	-	-	-	-
ASM	-	-	-	-
··	-	-	-	-
SOR	1620.	2640.	214	2.94
··	··	··	··	2.21

TFQMR1

PC	2-Norm $\times 10^{-9}$	∞-Norm $\times 10^{-10}$	Iterations	Time (sec)
None	38500.	159000.	5225	39.34
··	47700.	198000.	··	19.72
Jacobi	3020.	3430.	512	3.94
··	2990.	3170.	··	1.97
Bjacobi	464000.	3260000.	180	3.84
··	308000.	1610000.	··	1.66
ASM	22600.	95800.	144	3.84
··	21200.	90200.	··	1.59
SOR	1800.	2300.	214	3.03
··	1790.	2350.	··	2.23

TFQMR2

PC	2-Norm $\times 10^{-9}$	∞-Norm $\times 10^{-10}$	Iterations	Time (sec)
None	-	-	-	-
··	-	-	-	-
Jacobi	-	-	-	-
··	-	-	-	-
Bjacobi	39700.	41500.	413	12.00
··	49900.	29200.	445	7.94
ASM	8080.	4270.	377	15.81
··	8180.	3570.	376	6.24
SOR	1670.	1200.	748	16.88
··	1790.	1640.	576	9.14

CR

PC	2-Norm $\times 10^{-9}$	∞-Norm $\times 10^{-10}$	Iterations	Time (sec)
None	55.5	37.2	6682	23.66
··	··	··	··	13.05
Jacobi	17.7	8.80	744	2.81
··	··	··	··	1.48
Bjacobi	9.93	5.24	222	2.34
··	··	··	··	1.06
ASM	-	-	-	-
··	-	-	-	-
SOR	11.5	5.75	303	2.09
··	··	··	··	1.61

FGMRES

PC	2-Norm $\times 10^{-9}$	∞-Norm $\times 10^{-10}$	Iterations	Time (sec)
None	2.00	1.18	215247	1527.46

··	1.99	1.75	236216	651.07
Jacobi	2.06	0.873	2147	15.85
··	2.05	0.946	2130	5.97
Bjacobi	2.07	1.18	600	8.82
··	2.06	1.20	··	3.30
ASM	2.00	1.16	501	8.99
··	2.07	0.815	500	3.17
SOR	2.06	1.24	768	8.41
··	2.05	1.05	769	4.80
MINRES				
PC	**2-Norm $\times 10^{-9}$**	**∞-Norm $\times 10^{-10}$**	**Iterations**	**Time (sec)**
None	290.	107.	6814	32.31
··	··	··	··	15.98
Jacobi	102.	39.6	737	3.59
··	··	··	··	1.75
Bjacobi	23.1	19.4	221	2.72
··	··	··	··	1.14
ASM	-	-	-	-
··	-	-	-	-
SOR	26.3	10.7	300	2.50
··	··	··	··	1.72
SYMMLQ				
PC	**2-Norm $\times 10^{-9}$**	**∞-Norm $\times 10^{-10}$**	**Iterations**	**Time (sec)**
None	63.9	105.	7261	33.71
··	··	··	··	16.49
Jacobi	-	-	-	-
··	-	-	-	-
Bjacobi	-	-	-	-
··	-	-	-	-
ASM	-	-	-	-
··	-	-	-	-
SOR	-	-	-	-
··	-	-	-	-
BiCGSTAB(k)				
PC	**2-Norm $\times 10^{-9}$**	**∞-Norm $\times 10^{-10}$**	**Iterations**	**Time (sec)**
None	77.6	41.5	6760	49.03
··	··	··	··	25.89
Jacobi	43.9	22.2	492	3.65
··	··	··	··	1.91
Bjacobi	27.4	13.6	154	3.22
··	··	··	··	1.43
ASM	28.1	29.9	142	3.69
··	··	··	··	1.56
SOR	20.1	8.00	202	2.88
··	··	··	··	2.10

Table 2: Test results for iterative solvers and preconditioners (PC) using 8 PEs on the O3K (white) and XT3 (shaded) for the SPD matrix

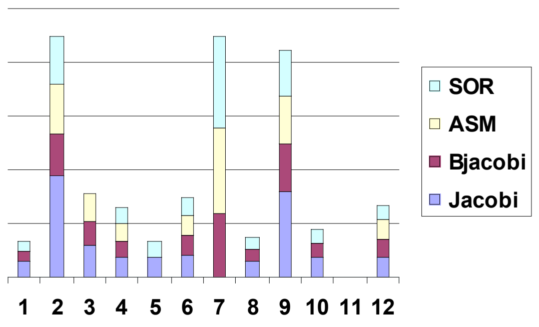

Figure 8: O3K solver times for the SPD matrix

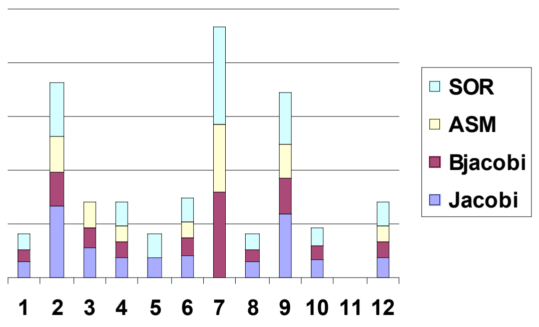

Figure 9. XT3 solver times for the SPD matrix

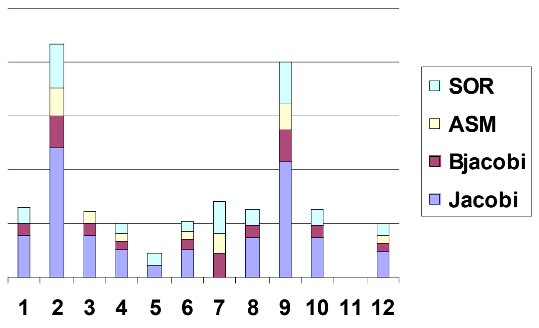

Figure 10: XT3 iteration counts for the SPD matrix

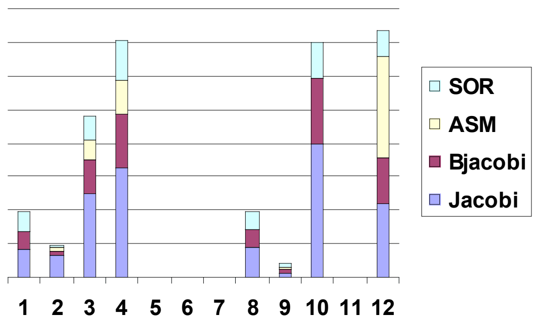

Figure 11: XT3 $||r||_\infty$ for the SPD matrix

CG – Jacobi

PEs	Nodes	Elems	Iters	Time (sec)	Strong SP	Weak SP
8	102996	187902	768	2.78		
16	102996	187902	768	1.56	1.78	
32	197409	375804	1095	2.38		0.66
8	102996	187902	768	1.50		
16	102996	187902	768	0.85	1.76	
32	197409	375804	1095	1.25		0.68
64	763887	1503216	3652	7.88		0.19

CG – Bjacobi

PEs	Nodes	Elems	Iters	Time (sec)	Strong SP	Weak SP
8	102996	187902	224	1.94		
16	102996	187902	257	0.98	1.98	
32	197409	375804	594	2.17		0.45
8	102996	187902	224	1.03		
16	102996	187902	257	0.59	1.75	
32	197409	375804	594	1.36		0.43
64	763887	1503216	1378	6.43		0.16

GMRES – Bjacobi

PEs	Nodes	Elems	Iters	Time (sec)	Strong SP	Weak SP
8	102996	187902	586	7.97		
16	102996	187902	584	2.67	2.99	
32	197409	375804	1043	4.99		0.54
8	102996	187902	587	3.22		
16	102996	187902	584	1.56	2.06	
32	197409	375804	1043	2.78		0.56
64	763887	1503216	3892	22.81		0.14

GMRES – ASM

PEs	Nodes	Elems	Iters	Time (sec)	Strong SP	Weak SP
8	102996	187902	514	9.22		
16	102996	187902	563	4.72	1.95	
32	197409	375804	943	7.93		0.60
8	102996	187902	515	3.30		
16	102996	187902	563	1.96	1.68	
32	197409	375804	944	3.42		0.57
64	763887	1503216	3892	22.81		0.14

BiCGSTAB – Bjacobi

PEs	Nodes	Elems	Iters	Time (sec)	Strong SP	Weak SP
8	102996	187902	224	4.22		
16	102996	187902	170	1.21	3.49	
32	197409	375804	386	2.94		0.42
8	102996	187902	224	2.00		
16	102996	187902	170	0.75	2.67	
32	197409	375804	386	1.72		0.44
64	763887	1503216	848	7.74		0.29

BICGSTAB – ASM

PEs	Nodes	Elems	Iters	Time (sec)	Strong SP	Weak SP
8	102996	187902	141	3.53		
16	102996	187902	144	1.55	2.28	
32	197409	375804	310	3.59		0.43
8	102996	187902	141	1.53		
16	102996	187902	144	0.88	1.74	
32	197409	375804	310	1.97		0.45
64	763887	1503216	881	10.76		0.14

CR – Bjacobi

PEs	Nodes	Elems	Iters	Time (sec)	Strong SP	Weak SP
8	102996	187902	222	2.34		
16	102996	187902	253	0.97	2.41	
32	197409	375804	572	2.21		0.41
8	102996	187902	222	1.06		
16	102996	187902	253	0.60	1.77	
32	197409	375804	572	1.35		0.44
64	763887	1503216	1312	6.29		0.17

CR – SOR

PEs	Nodes	Elems	Iters	Time (sec)	Strong SP	Weak SP
8	102996	187902	303	2.09		
16	102996	187902	328	1.17	1.79	
32	197409	375804	711	2.65		0.44
8	102996	187902	303	1.61		
16	102996	187902	328	0.87	1.85	
32	197409	375804	711	1.85		0.47
64	763887	1503216	-	-	-	-

MINRES – Bjacobi

PEs	Nodes	Elems	Iters	Time (sec)	Strong SP	Weak SP
8	102996	187902	221	2.72		
16	102996	187902	252	1.08	2.52	
32	197409	375804	568	2.36		0.46
8	102996	187902	221	1.14		
16	102996	187902	252	0.63	1.81	
32	197409	375804	568	1.41		0.45
64	763887	1503216	2440	12.49		0.09

MINRES – SOR

PEs	Nodes	Elems	Iters	Time (sec)	Strong SP	Weak SP
8	102996	187902	300	2.50		
16	102996	187902	325	1.30	1.92	
32	197409	375804	9489	38.78		0.03
8	102996	187902	300	1.72		
16	102996	187902	325	0.91	1.89	
32	197409	375804	9489	25.90		0.04
64	763887	1503216	-	-	-	-

Table 3: Iteration count and speedup (SP) values for preconditioner/solver combinations for the O3K (white) and XT3 (shaded)

$$\|\mathbf{r}\|_2 < \varepsilon\|\mathbf{b}\|_2, \quad \varepsilon = 10^{-15} \qquad (3)$$

was used. This is a stringent criterion that could tax some solver/preconditioner combinations. But since $\|\mathbf{b}\|_2 = 1.36$ (10^6) for the original mesh, the absolute convergence criterion of 1.36 (10^{-9}) is within acceptable limits of machine accuracy. In fact, SYMMLQ with the Jacobi, block Jacobi, or SOR preconditioners was the only additional method to converge when the convergence criterion was increased to 10^{-13}. For the SPD matrix A, Figures 8 and 9 show the elapsed times for the 12 solvers on the O3K and XT3, respectively. Figure 10 shows the solver iteration counts for the XT3, and Figure 11 shows the infinity norm of the final computed residual vector. For the SPD matrix, Table 3 shows the elapsed times and speedups for certain solvers when solving the linear systems corresponding to larger meshes. Finally, Figures 12 and 13 show the elapsed times when solving the nonsymmetric linear system corresponding to the original mesh.

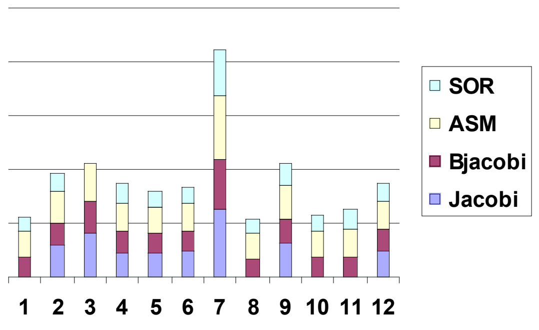

Figure 12. O3K solver times for the nonsymmetric matrix

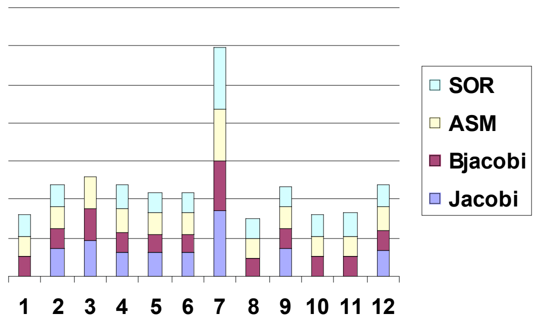

Figure 13. XT3 solver times for the nonsymmetric matrix

5. Conclusions

Conclusions observed from this study are as follows:

1) The times for loading the matrices and vectors into PETSc are small compared with the solver time if enough memory for the arrays is allocated in the initialization process,
2) the load times can be hundreds of times larger than the solver times if space for the **A** matrix is allocated dynamically,
3) the XT3 was approximately twice as fast as the O3K,
4) the GMRES solver was the slowest to achieve a given convergence criterion but produced the most accurate solution,
5) the successive over-relaxation preconditioner performed much better on the O3K than on the XT3,
6) the overall best solvers for these linear systems were Conjugate Gradient and Conjugate Residual using the block Jacobi preconditioner,
7) some solvers gave identical results on the O3K and XT3, while others did not with even the number of iterations being different, and
8) a superlinear speedup was observed for some solvers for small processor counts on the O3K. For example, GMRES using the block Jacobi preconditioner gave a speedup of 2.99 on the O3K when doubling the number of PEs from 8 to 16. Here, the iteration count was 586 on 8 PEs and 584 on 16 PEs.

Acknowledgment

This work was supported in part by a grant of computer time from the DoD High Performance Computing Modernization Program at the ERDC MSRC, Information Technology Laboratory, Vicksburg, MS.

References

1. Dongarra, J.J, I.S. Duff, D.C. Sorensen, and H.A. van der Vorst, *Numerical Linear Algebra for High-Performance Computers*, SIAM, Philadelphia, 1998.

2. Greenbaum, Anne, *Iterative Methods for Solving Linear Systems*, SIAM, Philadelphia, 1997.

3. Karypis Lab, http://glaros.dtc.umn.edu/gkhome/views/metis/metis/main.html, 2007.

4. Kelley, C.T., *Iterative Methods for Linear and Nonlinear Equations*, SIAM, Philadelphia, 1995.

5. PETSc, http://www-unix.mcs.anl.gov/petsc/petsc-as/index.html, 2007.

6. Rheinboldt, W.C., *Methods for Solving Systems of Nonlinear Equations, Second Edition*, SIAM, Philadelphia, 1998.