DISTRIBUTED MEMORY IMPLEMENTATION OF MULTIGRID METHODS FOR GROUNDWATER FLOW PROBLEMS WITH ROUGH COEFFICIENTS

G. Mahinthakumar
Center for Computational Sciences
Oak Ridge National Laboratory
Oak Ridge, TN 37831–6203
email: kumar@ornl.gov

F. Saied*
Department of Computer Science
University of Illinois at Urbana–Champaign
Urbana, IL 61801
email: saied@cs.uiuc.edu

KEYWORDS

Hydrology, Multiprocessors, Numerical methods, Partial differential equations, Multigrid method.

ABSTRACT

In this paper we present parallel solvers for large linear systems arising from the finite–element discretization of three–dimensional groundwater flow problems. We have tested our parallel implementations on the Intel Paragon XP/S 150 supercomputer using up to 1024 parallel processors. Our solvers are based on multigrid and Krylov subspace methods. Our goal is to combine powerful algorithms and current generation high performance computers to enhance the capabilities of computer models for groundwater modeling. We demonstrate that multigrid can be a scalable algorithm on distributed memory machines. We demonstrate the effectiveness of parallel multigrid based solvers by solving problems requiring more than 64 million finite–elements in less than a minute. Our results show that multigrid as a stand alone solver works best for problems with smooth coefficients, but for rough coefficients it is best used as a preconditioner for a Krylov method.

BACKGROUND

In order to determine flow fields in a groundwater aquifer, a partial difference equation (p.d.e) commonly referred to as the groundwater flow equation needs to be solved. For the steady–state saturated case, this equation is an elliptic p.d.e given by

$$\nabla(K\nabla h) - q = 0 \qquad (1)$$

where K is the hydraulic conductivity tensor, h is the head field, and q represents the source/sink terms coming from injection/pumping wells. In general, finite–element or finite–difference techniques are used to discretize Equation (1).

For many realistic problems, the groundwater flow equation involves rough coefficients (tensor K) resulting from heterogeneous hydraulic conductivity fields (or K–fields). In order to resolve fine–scale heterogeneity effects on large–scale regional models a fine

discretization is required (e.g. in the order of few meters) even for regional models (e.g. in the order of kilometers). For such problems finite–element or finite–difference discretizations give rise to very large linear systems (in the order of 100's of millions) that need to be solved.

The matrices that result from the discrete approximation of Equation (1) are sparse, symmetric and positive definite. The preconditioned conjugate gradients is a popular Krylov method (see next section) commonly used to solve such systems (Meyer *et al* 1989; Mahinthakumar and Valocchi, 1993). For methods such as preconditioned conjugate gradients, the number of iterations required for convergence increases with the problem size and the degree of heterogeneity when traditional preconditioners such as diagonal scaling or incomplete Cholesky are used. However, we can improve on this behavior by using a multigrid method, either on its own, or as a preconditioner in a Krylov subspace method. By using multigrid techniques we can make the convergence behavior less dependent on the problem size and the roughness of the coefficients (Alcouffe *et al* 1981; Behie and Forsyth 1983; Mckeon and Chu 1987; Ashby and Falgout 1995). But the difficulty in implementing multigrid techniques on distributed memory machines has prevented this method from gaining popularity on machines such as the Intel Paragon.

In this work we implement parallel multigrid based solvers on the Intel Paragon and compare their performance with diagonally preconditioned conjugate gradients. Performance is measured in terms of raw solution time, scalability, parallel efficiency, and Megaflop rate. Efficiency of multigrid methods for increasing problem sizes and increasing roughness is also compared.

KRYLOV SUBSPACE METHODS

Krylov subspace methods for solving a linear system $Ax = b$ are iterative methods that pick the j–th iterate from the following affine subspace

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

where x_0 is the initial guess, r_0 the corresponding residual vector and the Krylov subspace $K_j(A, r_0)$ is defined as

$$K_j(A, r_0) \equiv \text{span}\{r_0, Ar_0, \dots, A^{j-1}r_0\}.$$

These methods are very popular for solving large sparse linear systems because they are powerful and yet offer considerable savings in both computation and storage. Some of the more popular Krylov methods are Preconditioned Conjugate Gradients (PCG), Bi–Conjugate Gradient Stabilized (Bi–CGSTAB), Generalized Minimal Residual (GMRES), Quasi–Minimal Residual (QMR), and Adaptive Chebychev (Barret *et al.* 1994; Dongarra *et al.* 1991). Of these, PCG is used for only symmetric positive definite systems.

MULTIGRID METHODS

Multigrid methods for partial differential equations use multiple grids for resolving various features of the solution on the appropriate spatial scales (Brandt 1977; Briggs 1988; Holst and Saied 1993). They derive their efficiency by not attempting to resolve coarse scale features on the finest grid. The basic idea of multigrid is depicted in Fig 1, for the two–grid version.
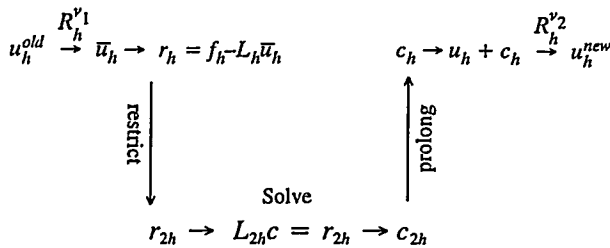
$$u_h^{old} \xrightarrow{R_h^{\nu_1}} \overline{u}_h \rightarrow r_h = f_h - L_h \overline{u}_h \qquad c_h \rightarrow u_h + c_h \xrightarrow{R_h^{\nu_2}} u_h^{new}$$

restrict $\downarrow$ prolong $\uparrow$

Solve

$$r_{2h} \rightarrow L_{2h}c = r_{2h} \rightarrow c_{2h}$$

Fig 1: The Two–Grid Version of Multigrid

Starting with and initial guess, u_h^{old}, on the finest grid, we apply ν_1 iterations of a smoothing method (R_h), such as weighted Jacobi or Gauss–Seidel and form the residual r_h of the resulting grid vector $\overline{u}_h$. This is "restricted" down to the coarse grid, where it is used as the right hand side (r_{2h}) of the coarse grid correction equation, $L_{2h}c = r_{2h}$, where L_{2h} is an appropriately defined coarse grid operator. The solution to this problem (c_{2h}) is interpolated back to the fine grid where it is added to the current approximation. Finally an additional ν_2 sweeps of the smoother are applied to the corrected approximation, to obtain u_h^{new}. The grid transfers involve fine to coarse (restriction) and coarse to fine (prolongation or interpolation) stages. At the coarsest level, a full matrix solve is performed before moving up to the next finer level. The coarse–grid solve is usually done by PCG or banded Gaussian elimination.

In practice, the two–grid algorithm is applied recursively. The most common approach is the V–cycle, where an initial guess must be supplied on the finest grid. The V–cycle can be used on its own or as a preconditioner to a Krylov method. The Full Multigrid

(FMG) method goes one step further and starting from the coarsest grid, "bootstraps" itself up to the finest grid, before doing the V–cycle. In this sense, FMG generates its own (usually very good) initial guess on the finest grid. In Fig 2 we schematically illustrate both these approaches (the finest grid is at the top).
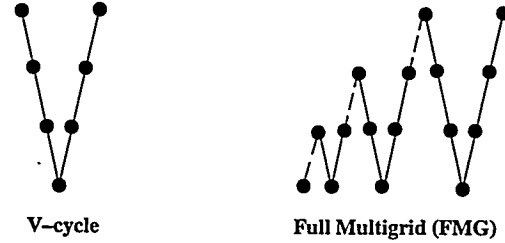

V–cycle   Full Multigrid (FMG)

Fig 2: V–Cycle and the Full Multigrid

ALGORITHMIC FRAMEWORK

For the three–dimensional isotropic case, Equation (1) reduces to

$$\frac{\partial}{\partial x}\left(K(x,y,z)\frac{\partial h}{\partial x}\right) + \frac{\partial}{\partial y}\left(K(x,y,z)\frac{\partial h}{\partial y}\right) + \frac{\partial}{\partial z}\left(K(x,y,z)\frac{\partial h}{\partial z}\right) = q \qquad (2)$$

where $K(x,y,z)$ is the hydraulic conductivity value at location (x,y,z). To solve Equation (2) we employ the Galerkin finite element discretization using eight–node linear brick elements (Istok 1989; Huyakorn and Pinder 1983). This discretization results in a matrix equation of the form $Ax = b$, where A is a sparse, symmetric positive definite matrix. For a rectangular grid structure and "natural ordering" of unknowns matrix A has a 27–diagonal banded non–zero structure. If the non–zero entries of the matrix are stored by diagonals, vectorizing compilers can generate extremely efficient code for operations like a matrix vector product, which are used in multigrid and Krylov methods. In our implementation we exploit symmetry and store only the 14 super–diagonals of the matrix.

For the multigrid implementation we use a V–cycle for each multigrid iteration. In order to construct the restriction operator within each V–cycle, we implemented three methods: simple injection, half weighting (7–point), and full weighting (27–point). For the prolongation operator (or interpolation) within each V–cycle, we use a linear interpolation scheme. The coarse grid operator for each level is simply the finite–element global matrix at these levels. For cases with rough coefficients, the elemental hydraulic conductivity values at the coarser levels are obtained by a local averaging scheme. We implemented three options to perform this averaging: arithmetic, geometric and harmonic averaging. For most of our test cases, simple injection and arithmetic averaging proved to be the best options. For the

coarse grid solve we used the diagonally preconditioned conjugate gradient method.

For the smoothing operation we chose the weighted (or underrelaxed) Jacobi, which, for $Ax = b$ and $A = D-L-U$, is defined by

$$x^{(n+1)} = [(1-\omega)I + \omega D^{-1}(L + U)]x^{(n)} + \omega D^{-1}b$$

where ω is the weighting factor. Although Jacobi is less powerful than methods such as Gauss–Seidel, it is easily parallelized and is generally adequate as a smoother.

We also implemented options to use multigrid as a preconditioner for CG and BiCGSTAB methods. Summarizing, our parallel solvers consisted of the following methods: DPCG (diagonally preconditioned conjugate gradients), MG (stand alone multigrid solver), MGCG (multigrid preconditioned conjugate gradients), and MGBiCGSTAB (multigrid preconditioned Bi–CGSTAB).

PARALLEL IMPLEMENTATION

Our parallel implementation is currently restricted to the Intel Paragon architecture. We used the Intel Paragon machines at the Oak Ridge National Laboratory's Center for Computational Sciences (CCS) for our code development and testing. There are three Intel Paragon machines at CCS: XP/S 5 (66 GP–nodes), XP/S 35 (512 GP–nodes), and XP/S 150 (1024 MP–nodes). Although we used all 3 machines in our development phase, the results that will be presented here pertain to the XP/S 150. The XP/S 150 has 1024 MP (multiple thread) nodes connected by a 16 row by 64 column rectangular mesh configuration[**]. In our implementation we used these nodes only in single threaded mode. In single threaded mode, each node is theoretically capable of 75 Mflops (in double–precision arithmetic). Each node has a local memory of 64 Mb (execpt for 64 "fat" nodes which have 128 Mb each). The native message passing library on the Paragon is called nx. The inter–node message bandwidth is about 152 Mb/s for long messages ($\sim$ 1Mb) with a zero–length latency of 35 μs.

For the parallel implementation we used a two–dimensional (2–D) domain decomposition in the x and y directions as shown in Fig 3. A 2–D decomposition is generally adequate for groundwater problems because common groundwater aquifer geometries involve a vertical dimension which is much shorter than the other two dimensions. For the finite–element discretization such decomposition involves communication with at most 8 neighboring processors. We note here that a 3–D decomposition in this case will require communication with up to 26 neighboring processors.

[**] see CCS web page at http://www.ccs.ornl.gov
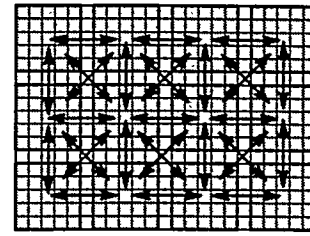
☐ overlapping processor regions ☐ individual processor regions

(arrows show communication pattern)

Fig 3: Plan View of Two–Dimensional Domain Decomposition

We overlap one layer of processor boundary elements in our decomposition to avoid additional communication during the assembly stage at the expense of some duplication in element computations. There is no overlap in node points. In order to preserve the 27–diagonal band structure within each processor submatrix, we perform a local numbering of the nodes for each processor subdomain. This resulted in non–contiguous rows being allocated to each processor in the global sense. For local computations each processor is responsible only for its portion of the rows which are locally contiguous. However, such numbering gives rise to some difficulties during explicit communication and I/O stages. For example, in explicit message passing, non–contiguous array segments had to be gathered into temporary buffers prior to sending. These are then unpacked by the receiving processor. This buffering contributes somewhat to the communication overhead. When the solution output is written to a file we had to make sure that the proper order is preserved in the global sense. This required non–contiguous writes to a file resulting in I/O performance degradation particularly when a large number of processors were involved.

For simplicity we use the same static decomposition at all multigrid levels. This strategy puts an upper limit on the number of multigrid levels that can be used because even the coarsest grid problem has to be distributed across all processors. However, this strategy is not always bad since the convergence of the multigrid method deteriorates with increasing number of levels.

All explicit communications between neighboring processors were performed using asynchronous nx calls. System calls were used for global communication operations such as those used in dot products. The codes are written in FORTRAN using double–precision arithmetic.

MODEL PROBLEM

For all the test simulations we setup a model problem as shown in Fig 4. This setup corresponds to a contamination scenario where the contaminant leaches from a single rectangular source into a naturally flowing groundwater aquifer.
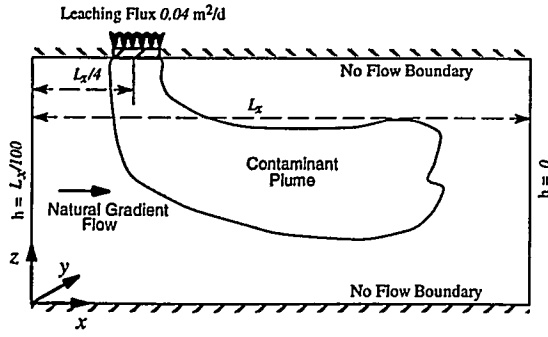
Fig 4: Vertical Cross–Section of Model Problem

The flow field generated from such simulation can be used as an input to a transport simulator to generate the contaminant plume (Mahinthakumar 1995). Boundary conditions for this setup are as follows: Fixed heads of $h = L_x/100$ and $h = 0$ at the faces of $x = 0$ and $x = L_x$ respectively, a rectangular patch of $L_x/8 \times L_y/8$ centered at $(x = L_x/4, y = L_y/2, z = L_z)$ with a uniformly distributed flux of 0.04 m²/d, and no flow boundaries else where. For tests involving heterogeneous K–fields (i.e. rough coefficients), we obtained the spatially correlated random K–fields by using a parallelized version of the turning bands code (Tompson *et al* 1989). The degree of heterogeneity is measured by the parameter σ, which is an input parameter to the turning bands code.

PERFORMANCE RESULTS AND DISCUSSION

In this section we present and compare the performance of our implementations with respect to problem size, scalability, raw floating point performance, and roughness of coefficients. The following selections were used for all performance tests unless otherwise stated:

- convergence criteria for matrix solution: two–norm of relative residual $< 10^{-8}$

- coarse grid solve: DPCG with tolerance set to 10^{-4}

- smoothing: $v_1 = v_2 = 3$, weighted Jacobi with $\omega = 0.95$

- homogeneous K–field (constant coefficient case)

- timings are for matrix solution only

- number of multigrid levels within each V–cycle = 4

where v_1, v_2 are the number of pre– and post–smoothing operations respectively. Timings were obtained by the dclock() system call. Timings reported are for the processor that takes the maximum time.

Increasing Problem Size

In Table 1 we present the iteration counts and solution timings for MG and DPCG methods as the problem size increases. As the problem size doubled in either x or y directions, the 2–D processor configuration was also changed accordingly. NP is the total number of processors used. The processor mesh configuration is $NP_x \times NP_y$.

NP	$NP_x \times NP_y$ (= NP)	$nx \times ny \times nz$	MG Iter	MG Time	DPCG Iter	DPCG Time
1	1 x 1	33 x 33 x 65	6	23.0	98	42.3
2	2 x 1	65 x 33 x 65	6	24.1	147	61.0
4	2 x 2	65 x 65 x 65	6	25.2	151	63.6
8	4 x 2	129 x 65 x 65	6	26.3	245	98.7
16	4 x 4	129 x 129 x 65	6	27.2	242	98.4
32	8 x 4	257 x 129 x 65	6	28.4	358	142
64	8 x 8	257 x 257 x 65	6	29.9	429	171
128	16 x 8	513 x 257 x 65	6	32.9	664	260
256	16 x 16	513 x 513 x 65	6	32.7	756	299
512	32 x 16	1025 x 513 x 65	6	40.4	1203	469
1024	32 x 32	1025 x 1025 x 65	6	45.0	1612	670

Table 1: Effect of Increasing Problem Size

As the problem size increased, the DPCG iterations increased dramatically while the multigrid iterations remains fixed. It is interesting to note here that DPCG is also sensitive to the aspect ratio of the problem while MG is not. For example, the increase in iteration count for DPCG is larger for the 257 x 257 x 65 to 513 x 257 x 65 transition than the 513 x 257 x 65 to 513 x 513 x 65 transition. This phenomena can be observed for all cases. Although we do not show it here, we note that both MGCG and MGBiCGSTAB behaved similarly to MG. We would like to note here that using MG we were able to solve a 68 million node problem in only 45 seconds.

Scalability of Multigrid

In this section we analyze the scalability of MG by increasing the problem size with corresponding increase in the number of processors. In fact we used the same test results that were obtained for the previous section. For perfect scalability, the solution time should remain fixed as we increase the problem size accordingly with the number of processors. In Fig 5 we show the scaling behavior of our parallel multigrid implementation. Note that a horizontal line on this plot would correspond to perfect scalability.

A first glance at Fig 5 may indicate that multigrid does not scale very well at all. The solution time for the largest problem (approximately 68 M nodes) on 1024 processors is approximately twice that for the smallest problem (approximately 70 K nodes) on 1 processor. A closer inspection of our timings revealed that most of the loss in scalability is due to the coarse grid solve which is performed by DPCG. Even though the multigrid iterations remain the same throughout the scaling process (see Table 1), the DPCG coarse grid solve iterations increase because the coarse grid problem becomes larger as we scale. By the same token we can see from Fig 5 that all phases of the V–cycle other than the coarse grid solve show very good scalability.
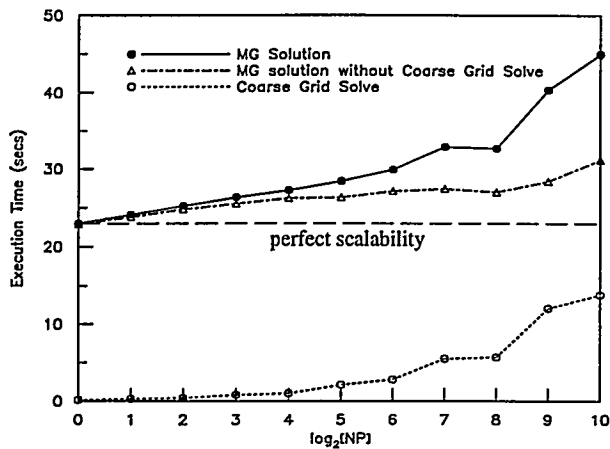
Fig 5: Scaling Behavior of Multigrid

Convergence Behavior for Large Problems

The convergence behavior of each method for the 68 M node problem is shown in Fig 6. Note that in this figure we plot the norm of the relative residual against the total time (includes initial setup, I/O, matrix computation and assembly, and matrix solution time). It is evident that all MG based methods show very good and monotonic convergence rates compared to DPCG. It is interesting to note that MGCG is performing well even though our V–cycle preconditioner is not symmetric.
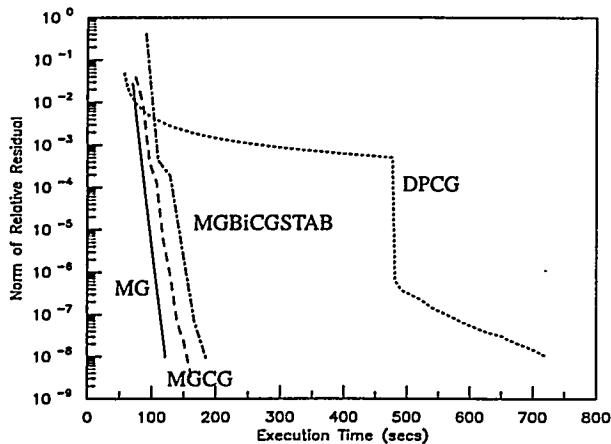
Fig 6: Covergence Behavior as Function of Total Time
for the 1025 x 1025 x 65 Problem

Parallel Performance for Fixed Problem Size

We measured the parallel performance for a fixed problem of size 257 x 257 x 65 (3.9 M nodes) by increasing the number of parallel processors from 64 to 1024. The results are shown in Fig 7. For the multigrid based methods 3 multigrid levels and 2 pre– and post– smoothing passes were used.
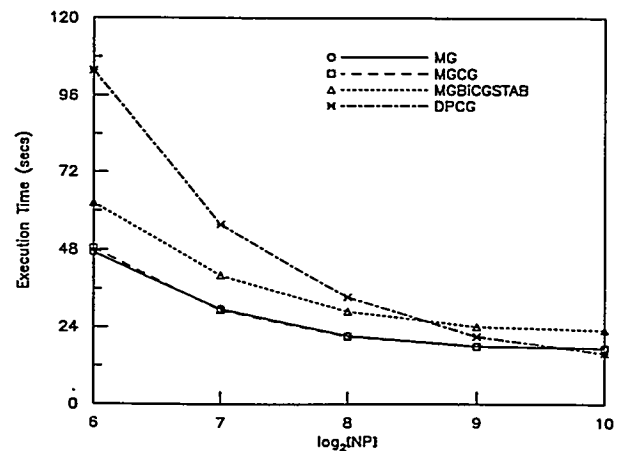

Fig 7: Parallel Performance for Fixed Problem Size

From Fig 7 we can see that the MG based methods do not speed up as well as DPCG. We attribute this behavior to greater overheads in the coarse grid operations as the problem size per processor becomes smaller. We would like to note here that even though DPCG has better parallel efficiency it is still slower than MG for all processor numbers except for NP=1024.

In Fig 8 we compare the parallel efficiency of the total time to the matrix solution and explicit inter–processor communication times. Timings are for the fixed size problem (257 x 257 x 65) using the MG solver. The total time includes initial setup, finite–element matrix assembly, matrix solution and I/O. From Fig 8 we can observe that even though the MG solution has subpar parallel efficiency, the total time has a reasonable speed up behavior. The explicit communication time decreases slightly in the beginning and then starts to gradually increase as we increase NP. We attribute the initial drop in communication time to messages becoming shorter (message bandwidth limited) and the increase near the end to the latency overhead.
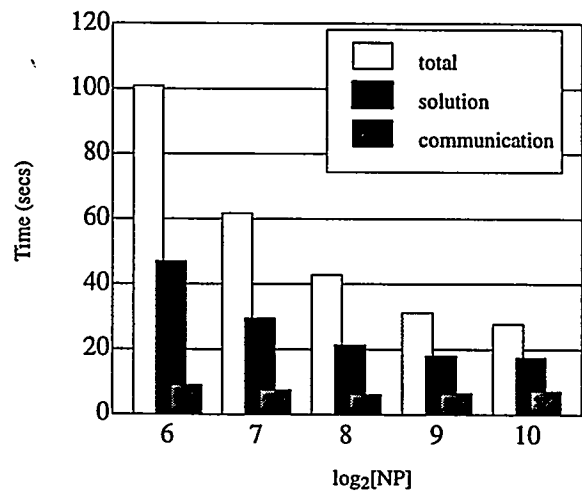

Fig 8: Overall Parallel Efficiency

Roughness of Coefficients

The roughness of the coefficients of Equation (2) is measured by parameter σ which represents the degree of heterogeneity of the K-field. In Table 2 we show the effect of increase in σ on the convergence behavior of our solvers. The results we present are for a 257 x 257 x 65 problem on 256 processors. The multigrid based methods used 3 levels and 2 pre- and post- smoothings. The results show that multigrid is best used a preconditioner when the heterogeneity is high. Examining Table 2 reveals that the convergence of MGCG and MGBiCG-STAB is less affected by σ than DPCG. This is interesting because we did not use operator based restriction and prolongation for the MG based methods in our implementation. We believe this is somewhat related to the robustness of our coarse grid operator.

σ	MG		MGCG		MGBiCG		DPCG	
	Iter	Time	Iter	Time	Iter	Time	Iter	Time
0.0	9	18.7	8	21.7	4	20.4	256	33.4
0.5	10	23.2	8	24.2	5	26.9	366	49.2
1.0	13	30.7	10	28.1	5	27.0	609	81.1
1.5	25	58.6	13	34.6	7	36.0	1043	137
2.0	Diverged		20	50.6	10	49.7	1489	195
2.5	Diverged		29	71.3	16	77.5	1844	242

Table 2: Effect of K-field Heterogeneity

Floating Point Performance

We estimated the Mflop rates for our solvers using a MATLAB routine which computes the number of floating point operations as a function of various V-cycle parameters. The peak performance for the MG solver is about 4.2 Gflops compared to 10.3 for DPCG. These numbers are for the largest problem shown in Table 1. For the MG solver the Mflop per processor ranged from 7.8 for the single processor problem in Table 1 to 4.1 for the largest problem on 1024 processors. We should keep in mind that these numbers are for sparse matrix operations that usually do not give good floating point performance. For double precision floating point operations involving sparse matrices, 15 Mflops per processor is usually considered very good for the Portland Group Fortran compiler on the i860 chip.

CONCLUSIONS

We have implemented multigrid for the solution of the finite-element equations for the 3-D groundwater flow problem on distributed memory machines. Of the solvers we have implemented we conclude that multigrid solvers are the most efficient for solving very large groundwater flow problems on the Paragon. For example, for the 68 M node problem, DPCG would have to run at 150 Gflops to solve the problem as quickly as multigrid. The performance of our multigrid solvers could be further improved by optimizing the single processor

performance of major loops. For example by using calls to optimized BLAS routines for the saxpy and dot product operations. The robustness of our multigrid solvers with respect to increasing heterogeneity might be enhanced by using operator based interpolation and semi-coarsening. That is, for increasing heterogeneity the number of V-cycles that is required for convergence will not change appreciably.

In this work we have demonstrated that by combining powerful algorithms and current generation high performance computers the capabilities of computer models for groundwater modeling can be substantially enhanced.

REFERENCES

Alcouffe, R.E., A. Brandt, J.E. Dendy, and J.W. Painter. 1981. "The multigrid method for the diffusion equation with strongly discontinuous coefficients." *SIAM Journal of Scientific and Statistical Computing.* no. 2: 430-454.

Brandt, A. 1977. "Multilevel adaptive solutions to boundary value problems." *Mathematics of Computation.* vol. 31: 311-329.

Behie, A. and P. Forsyth, Jr. 1983. "Multigrid solution of three-dimensional problems with discontinous coefficients." *Applied Mathematics and Computation.* 13: 229-240.

Ashby, S.F., and R.D. Falgout. 1995. "A Parallel Multigrid Preconditioned Conjugate Gradient Algorithm for Groundwater Flow Simulations." Technical Report UCRL-JC-122359, Lawrence Livermore National Laboratory, CA (Oct.).

Briggs, W.L., 1988. *A Multigrid Tutorial.* SIAM, Philadelphia, P.A.

Istok, J.D., 1989. *Groundwater modeling by the finite element method.* Water Resources Monograph 13, American Geophysical Union, Washington, D.C.

Holst, M. and F. Saied 1993. "Multigrid solution of the Poisson-Boltzmann equation." *Journal of Computational Chemistry.* vol. 14, no. 1: 105-113.

Huyakorn, P.S., and G.F. Pinder 1983. *Computational Methods in Subsurface Flow,* Academic Press, New York, N.Y.

Mahinthakumar, G., and A. J. Valocchi. 1992. "Application of the Connection Machine to flow and transport problems in three-dimensional heterogenous aquifers." *Advances in Water Resources.* vol. 15: 289-302.

McKeon, T.J., and W.-C. Chu. 1987. "A multigrid model for steady flow in partially saturated porous media." *Water Resources Research.* vol. 23, no. 4 (Apr.): 542-550.

Tompson, A.F.B., R. Aboubu, and L.W. Gelhar. 1989. "Implementation of the three-dimensional turning bands random field generator." *Water Resources Research.* vol. 25, no. 10 (Oct.): 2227-2243.

DISCLAIMER