ORNL/TM-13441

Center for Computational Sciences

MULTIGRID AND KRYLOV SOLVERS FOR LARGE SCALE FINITE ELEMENT GROUNDWATER FLOW SIMULATIONS ON DISTRIBUTED MEMORY PARALLEL PLATFORMS

Kumar Mahinthakumar [‡]
Faisal Saied [⋆]

[‡] Center for Computational Sciences
Bldg 4500N, MS 6203
Oak Ridge National Laboratory
Oak Ridge, TN 37831.

[⋆] Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801.

Date Published: July 1997

Prepared by the
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831
managed by
Lockheed Martin Energy Research Corp.
for the
U.S. DEPARTMENT OF ENERGY
under Contract No. DE-AC05-96OR22464

Contents

List of Tables

List of Figures

Acknowledgements

This work was sponsored by the Center for Computational Sciences of the Oak Ridge National Laboratory managed by Lockheed Martin Energy Research Corporation for the U.S. Department of Energy under contract number DE-AC05-960R22464. We acknowledge the support by the Center of Computational Sciences at Oak Ridge National Laboratory, the Mathematics and Computer Science Division at the Argonne National Laboratory and the National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign for the use of their supercomputing facilities.

MULTIGRID AND KRYLOV SOLVERS FOR LARGE SCALE FINITE ELEMENT GROUNDWATER FLOW SIMULATIONS ON DISTRIBUTED MEMORY PARALLEL PLATFORMS

Kumar Mahinthakumar

Faisal Saied

Abstract

In this report we present parallel solvers for large linear systems arising from the finite-element discretization of the three-dimensional steady-state groundwater flow problem. Our solvers are based on multigrid and Krylov subspace methods. The parallel implementation is based on a domain decomposition strategy with explicit message passing using NX and MPI libraries. We have tested our parallel implementations on the Intel Paragon XP/S 150 supercomputer using up to 1024 parallel processors and on other parallel platforms such as SGI/Power Challenge Array, Cray/SGI Origin 2000, Convex Exemplar SPP-1200, and IBM SP using up to 64 processors. We show 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 70 million nodes in less than a minute. This is more than 25 times faster than the diagonal preconditioned conjugate gradient method which is one of the more popular methods for large sparse linear systems. Our results also 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 subspace method such as the conjugate gradient method. We show that even for extremely heterogeneous systems the multigrid preconditioned conjugate gradient method is atleast 10 times faster than the diagonally preconditioned conjugate gradient method.

1. Background

In order to determine flow fields in a groundwater aquifer, a partial differential 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

$$\nabla \cdot (K \nabla h) - q = 0 \tag{1}$$

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 (on the order of kilometers) a fine discretization is required (on the order of a few meters). For such problems finite-element or finite-difference discretizations give rise to very large linear systems (on the order of 10's of millions of nodes) that need to be solved.

The matrices that result from the discrete approximation of Equation (1) are sparse, symmetric and positive definite[8]. The preconditioned conjugate gradient method is a popular Krylov method (see next section) commonly used to solve such systems [13], [15]. 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 to some extent the roughness of the coefficients [1], [4], [14], [2]. 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. Ashby and co workers [2], [3] implemented multigrid preconditioned solvers for a finite-difference formulation of the groundwater flow problem on Cray T3D and J90 architectures. In this work we implement parallel multigrid based solvers for a finite-element formulation of the same problem on various distributed memory platforms. The performance of these solvers are then compared with a diagonally preconditioned conjugate gradient solver. Performance is measured in terms of raw solution time, scalability, parallel efficiency, and Megaflop rate (A Megaflop/s stands for 10^6 floating point operations per second). Efficiency of multigrid methods for increasing problem sizes and increasing roughness is also compared.

2. 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

$$K_j(A, r_0) = \text{span}\{r_0, Ar_0, \ldots, A^{j-1}r_0\}$$

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 [8], [17], [7]. Of these, PCG is used for only symmetric positive definite systems.

3. Multigrid Methods

Multigrid methods for partial differential equations use multiple grids for resolving various features of the solution on the appropriate spatial scales [5], [6], [10], [9]. They derive their efficiency by not attempting to resolve coarse scale features on the finest grid.

The basic idea of multigrid is depicted in Figure 1, for the two-grid version. Starting with an 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. 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, I_h^{2h}) and coarse to fine (prolongation or interpolation, I_{2h}^h) 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 performance of multigrid can be "tuned" through an appropriate choice of parameters like the number of levels, or the smoothing sweeps (ν_1, ν_2).

4. Algorithmic Framework

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

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

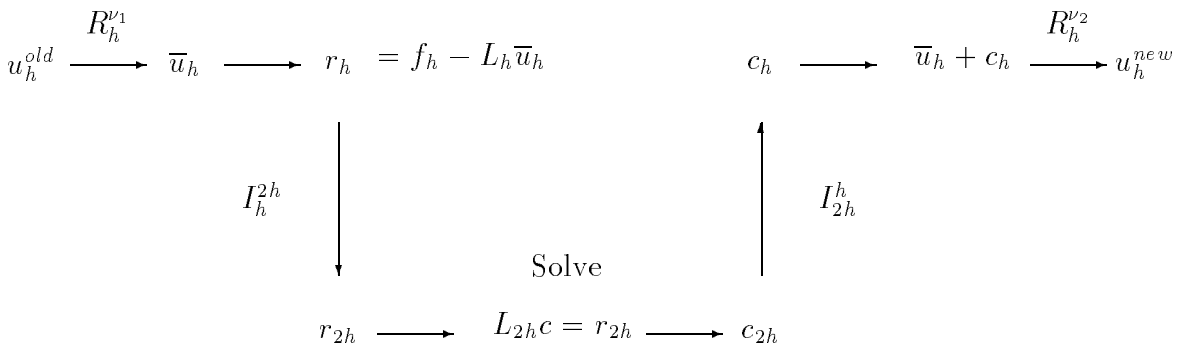

Figure 1: The Two-Grid Version of Multigrid

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 [12], [11]. This discretization results in a matrix equation of the form $Ax = b$, where A is a sparse, symmetric positive definite matrix. For a logically 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 (interpolation) operator 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 gradient method), MG (stand alone multigrid solver), MGCG (multigrid preconditioned conjugate gradient method), and MGBiCGSTAB (multigrid preconditioned Bi-CGSTAB). The results for BiCGSTAB with multigrid preconditioning were very similar to those for MGCG, and will not be presented here. We also note here that our multigrid implementations are currently restricted to uniform rectangular grids.

5. Parallel Implementation

Our parallel implementation was originally targeted for the Intel Paragon machines using the Paragon's native `NX` message passing library. The code was then ported to the SGI/Power Challenge Array, SGI/Cray Origin 2000, Convex Exemplar and IBM SP systems using an `MPI` (Message Passing Interface) implementation. Since most of our tests were performed on the Intel Paragon XPS/150 we briefly describe this architecture below.

The Intel Paragon XP/S 150 (1024 MP-nodes) at the Oak Ridge National Laboratory's Center for Computational Sciences (CCS)[1] has 1024 MP (multiple thread) nodes connected in a 16×64 rectangular mesh configuration. Each node has a local memory of 64 Megabytes. The native message passing library on the

[1]For more details, see the CCS web page at http://www.ccs.ornl.gov

Paragon is called `NX`. The inter-node message bandwidth is about 152 Mb/s for long messages ($\approx$ 1Mb) with a zero-length latency of 35 ms.

For parallelization we used a two-dimensional (2-D) domain decomposition in the x and y directions as depicted in Fig 2. 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 would require communication with up to 26 neighboring processors.

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 a 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 limits the number of multigrid levels that can be used because even the coarsest grid problem has to be distributed across all processors.

All explicit communications between neighboring processors were performed using asynchronous `NX` or `MPI` calls. System calls were used for global commu-

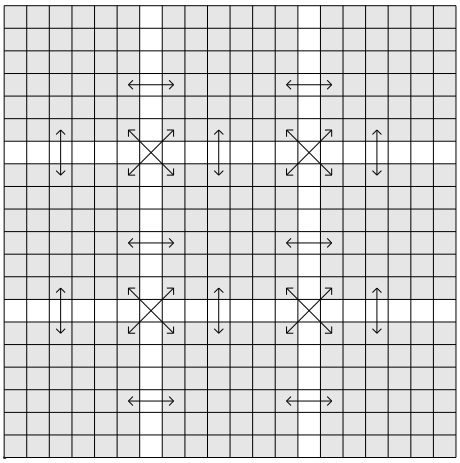

Figure 2: Plan View of Two-Dimensional Domain Decomposition. Each gray region belongs to a processor; the white regions are overlapped. The arrows show the communication pattern.

nication operations such as those used in dot products. The codes are written primarily in FORTRAN (except for some I/O routines which are in C) using double-precision arithmetic. Although each MP node on the XPS/150 is capable of using up to three parallel threads the results presented in this report are only for the single threaded mode. Our initial attempts at using multiple threads on the Paragon was not successful since the application is highly memory bandwidth limited. We note here that assembler level coding would be required to exploit even two processors of an MP node in this application and this level of effort is beyond the scope of this study.

6. 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.

The flow field generated from such simulation can be used as an input to a transport simulator to generate the contaminant plume [13]. Boundary conditions

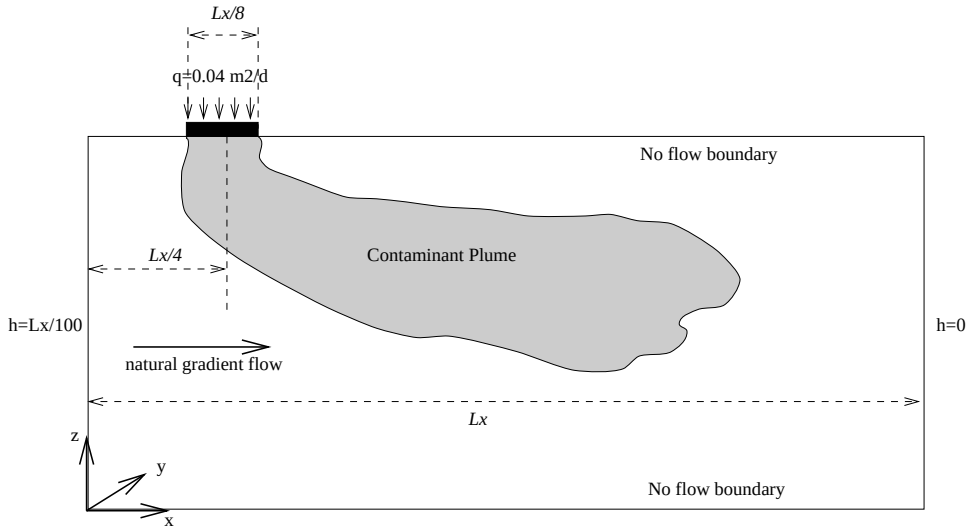

Figure 3: Vertical Cross-Section of Model Problem.

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 m^2/d, and no flow boundaries elsewhere. 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 [16]. The degree of heterogeneity is measured by the parameter σ, which is an input parameter to the turning bands code.

7. 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}
- homogeneous K-field (constant coefficient case)
- timings are for matrix solution only

In the following, P denotes the number of processors. All performance analyses are for the Intel Paragon XP/S 150 except section which deals with multiple platforms. Timings are obtained by `dclock()` (for the Paragon) or `MPI_wtime()` (for other platforms) system calls. Timings reported are for the processor that takes the maximum time.

7.1. Scalability of Multigrid and DPCG

We analyze the scalability of multigrid and DPCG by increasing the problem size with a corresponding increase in the number of processors (i. e. N/P is fixed). The results of this experiment are presented in Table 1. The grid sizes ranged from $33 \times 33 \times 65$ for a single processor to $1025 \times 1025 \times 65$ for the 1024 processors. The most striking result in Table 1 is that the multigrid iterations remain fixed, while the DPCG iterations grow as we scale up the problem size. Furthermore, we see that the multigrid solution time for the largest problem (approximately 68 M nodes) on 1024 processors is about twice that for the smallest problem (approximately 70 K nodes) on 1 processor. In particular, the 68 million node problem was solved in under 35 seconds on 1024 processors.

The multigrid data from Table 1 is plotted in Fig 4. The total multigrid solution time is broken down into the coarse grid solve time and the rest. 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, 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 4 that all phases of the V-cycle other than the coarse grid solve show very good scalability.

Table 1: Scaling Behavior of Multigrid and DPCG. Homogeneous K field, four grid levels, and $\nu_1 = \nu_2 = 3$. P is the number of processors.

P	Px × Py (= P)	nx × ny × nz	MG Iter	MG Time	DPCG Iter	DPCG Time
1	1 × 1	33 × 33 × 65	4	20.38	114	43
2	2 × 1	65 × 33 × 65	4	20.57	168	63
4	2 × 2	65 × 65 × 65	4	20.89	177	67
8	4 × 2	129 × 65 × 65	4	21.33	283	106
16	4 × 4	129 × 129 × 65	4	21.59	287	108
32	8 × 4	257 × 129 × 65	4	22.23	442	165
64	8 × 8	257 × 257 × 65	4	23.02	551	205
128	16 × 8	513 × 257 × 65	4	23.02	843	313
256	16 × 16	513 × 513 × 65	4	23.95	1085	406
512	32 × 16	1025 × 513 × 65	4	26.91	1658	646
1024	32 × 32	1025 × 1025 × 65	4	31.17	2142	907

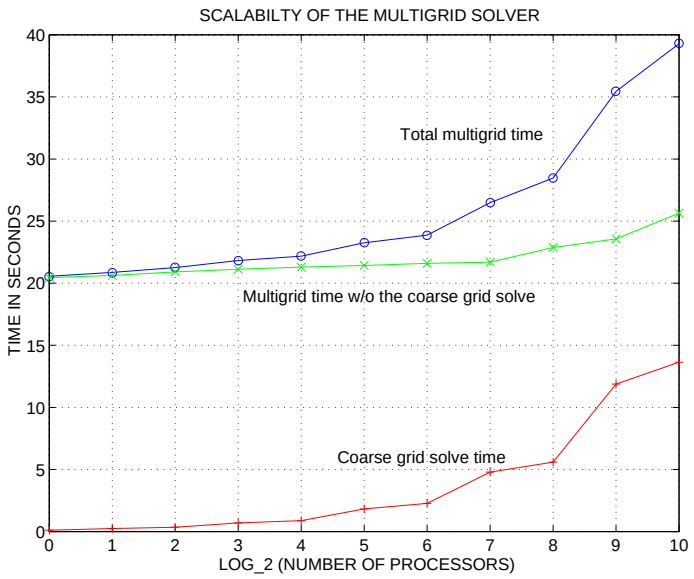

Figure 4: Scaling Behavior of Multigrid.

7.2. Parallel Performance for Fixed Problem Size

In Fig 5 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 ($257 \times 257 \times 65$) using the MG solver. The number of levels was three and $\nu_1 = \nu_2 = 3$. The total time includes initial setup, finite-element matrix assembly, matrix solution and I/O.

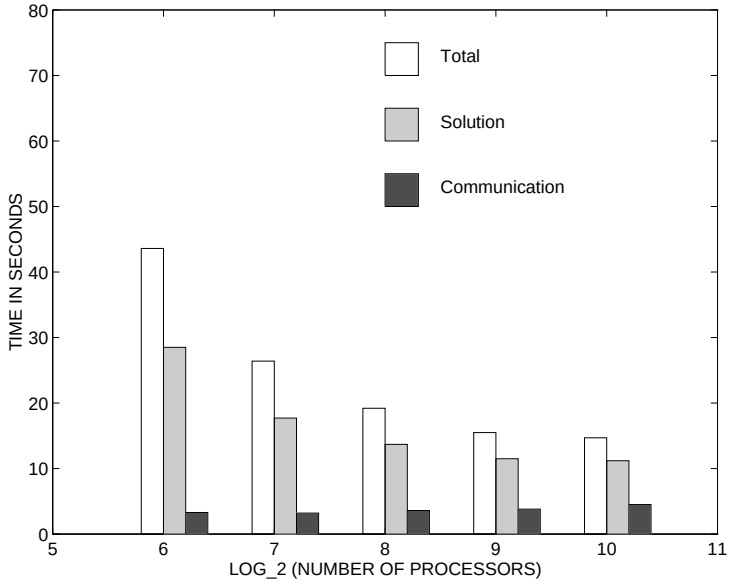

Figure 5: Parallel Performance for Fixed Problem Size ($257 \times 257 \times 65$). The number of multigrid levels was three and $\nu_1 = \nu_2 = 3$.

From Fig 5 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 P. 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.

7.3. Results on Multiple Platforms

In this section we compare the performance of the stand alone multigrid solver on a variety of parallel platforms for a fixed problem size of $129 \times 129 \times 17$. The comparison was done for IBM SP (4, 8, 16, 32, and 64 processors), Intel Paragon XPS/150 (4, 8, 16, 32, and 64 processors), Cray/SGI Origin 2000 (4, 8, 16, and 32 processors), SGI Power Challenge Array (4, 8, and 16 processors), and Convex Exemplar SPP-1200 (4 and 8 processors). In this test case we used a homogeneous K field, 3 multi grid levels, and 5 pre- and post- smoothings. In all cases we used `MPI` for message passing. Our aim here is to evaluate whether our implementation is satisfactory for machines with varying message passing, memory bandwidth, and floating point properties. We note here that on all systems except the Intel Paragon we simply used the default `'-O'` compilation optimization flag. On the Paragon we used the `'-Mvect -O3 -Knoieee'` flags. The results are shown in Table 2.

P	IBM SP	XPS/150	SGI-PC	Origin	Convex
4	8.7	21.2	11.2	6.0	29.8
8	4.6	11.6	6.2	2.4	18.8
16	2.8	7.0	3.7	1.2	-
32	2.1	4.5	-	1.0	-
64	1.8	3.5	-	-	-

Table 2: Comparison of MG solution time (in seconds) for various parallel systems. The Problem Size is fixed at $129 \times 129 \times 17$. The number of multigrid levels was three and $\nu_1 = \nu_2 = 5$. P is the number of processors.

It is evident from Table 2 that all machines exhibit the expected speedup behavior for the moderate number of processors tested here. In Figure 6 we compare the performance of the three bigger systems, namely, the IBM SP, XPS/150, and the Origin 2000 in terms of the total, solution, and communication times. This comparison is for a 257x129x65 problem with 4 grid levles and $\nu_1 = \nu_2 = 5$.

From Figure 6 we see that except for Origin 2000 the communication overhead is acceptable even though we use the same parallel decomposition on all four multigrid levels. We could not determine the cause for the slightly higher

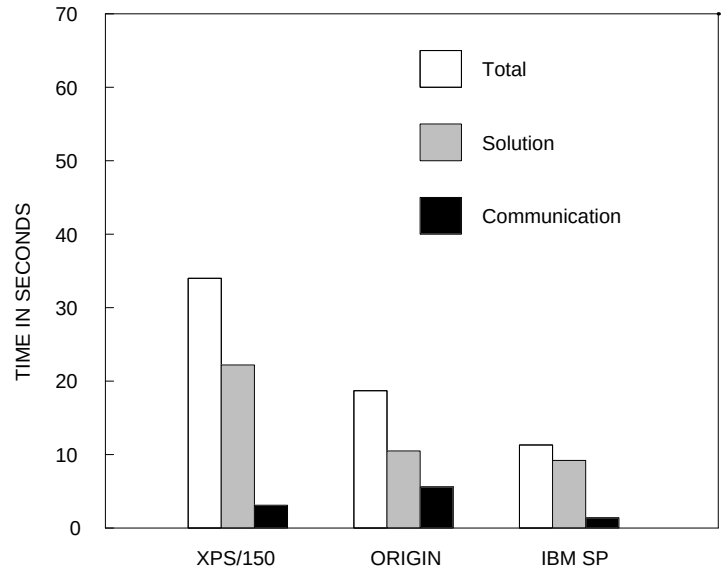

Figure 6: Comparing the overall parallel performance for a bigger problem. The Problem Size is fixed at $257 \times 129 \times 65$. The number of multigrid levels was three and $\nu_1 = \nu_2 = 5$. 32 processors was used in all cases.

communication overhead for the Origin 2000. We note here that the advertised peak internode message bandwidth of 156 Mb/s for a 32 processor Origin 2000 is comparable to the Intel Paragon numbers. Latency would have played a smaller role in the communication overhead since this problem is significantly larger than the one presented for 32 processors in Table 2.

7.4. Roughness of Coefficients

In this section we investigate the performance of our solvers for problems with rough coefficients. The roughness of the coefficients of Equation (2) is measured by a parameter σ^2 (variance of the log of K-field) which represents the degree of heterogeneity of the K-field. In these tests, $\sigma^2 = 0.0$ corresponds to a homogeneous K-field and $\sigma^2 \geq 3.0$ correspond to extremely heterogeneous K-fields. In Table 3 we show the effect of increase in σ^2 on the convergence behavior of our solvers. The tests were performed up to $\sigma^2 = 4.0$ which is considered extremly

high heterogeneity not common to many groundwater aquifers. The results we present are for a $1025 \times 1025 \times 65$ problem on 1024 processors. The multigrid-based methods used 5 levels and 5 pre- and post- smoothings. The results show that multigrid is best used as a preconditioner when the heterogeneity is high. Examining Table 3 reveals that the convergence of MGCG and DPCG are less affected by σ^2 than MG. However, for practical values of σ^2 we see that MGCG still outperfroms DPCG by at least a factor of 10. This is consistent with the findings of [2] who succesfully applied MGCG for heterogeneous groundwater flow problems using a finite difference implementation. Our success with MGCG is somewhat surprising since we did not use operator-based restriction and prolongation for the multigrid implementation. We believe this is related to the robustness of our coarse grid operator which is based on coefficient averaging and a finite element discretization.

Table 3: Effect of Varying Heterogeneity. 1024 Processors, $1024 \times 1024 \times 64$ mesh. The numbers in the table are times in seconds and in parentheses, the number of iterations. $\sigma^2 = 0.0$ corresponds to a homogeneous K-field.

σ^2	Multigrid	DPCG	MGCG
0.0	34.81 (4)	875.93 (2035)	44.78 (4)
0.5	68.2 (9)	1130.3 (2662)	63.0 (7)
1.0	110.7 (15)	1184.2 (2789)	76.1 (9)
2.0	203.2 (28)	1343.3 (3165)	113.3 (14)
3.0	353.1 (49)	1554.0 (3662)	141.6 (18)
4.0	568.3 (79)	1819.7 (4288)	178.6 (23)

7.5. Floating Point Performance

Since most of our tests were performed on the Intel Paragon, the performance results discussed in this section are only for this architecture. 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. Even though these numbers represent only a fraction of the theoretical peak for the Intel Paragon, we consider these reasonable since most of our operations involve sparse matrices or level 1 BLAS operations which are limited by memory bandwidth rather than the CPU speed. 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. In order to get a higher fraction of the peak, assembler level coding has to be exploited which is beyond the scope of this study.

7.6. Tuning the Performance of Multigrid

The performance of multigrid solvers can be tuned by varying parameters that control the multigrid V-cycle. For example, by selecting optimal values for the number of smoothings and the number of levels we can improve the performance of the solver for a given problem size and processor count.

In Figure 7, the effect of varying (ν_1, ν_2) is examined for the homogeneous case. Recall that ν_1 and ν_2 are the number of pre-smoothings and post-smoothings, respectively. For this experiment we chose $\nu_1 = \nu_2$. N/P, the number of unknowns per processor, was kept fixed for all the cases, $P = 1$, $P = 256$ and $P = 1024$. Note that the pay-off for doing more smoothings is greater for $P = 1024$ than for the single processor case. The reason is that the number of V-cycles, and hence the number of coarse grid solves, is reduced as ν_1, ν_2 are increased. This reduces the impact of the coarse grid solve which is the least efficient component of the parallel multigrid algorithm.

We also studied the performance of the code by varying the number of levels used in the multigrid algorithm. We note here that our code is limited to five levels on the Paragon, because we require the coarse grid problem to be distributed across all processors. Although we do not present the results here, for large problems, it pays to use all five levels because this cuts down the fraction of the

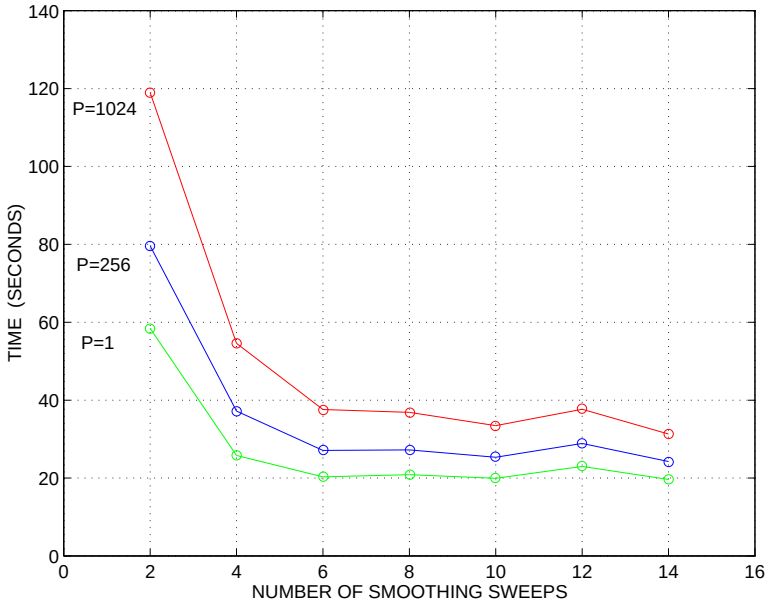

Figure 7: Effect of Varying (ν_1, ν_2). For this experiment we chose $\nu_1 = \nu_2$. N/P was kept fixed for all three cases, $P = 1$, $P = 256$ and $P = 1024$.

time spent in the coarse grid solver. However, the improvement in time decreased as we increased the number of levels (e.g. the improvement in time by going from 4 to 5 levels is less than that going from 3 to 4 levels). This implies that by going beyond 5 levels at the expense of additional coding and load imbalance overhead may not improve the performance appreciably.

8. Conclusions

We have implemented parallel solvers based on multigrid and conjugate gradient methods for the solution of finite-element equations for the 3-D groundwater flow problem on distributed memory machines. Our study indicates that multigrid based solvers are very efficient for solving very large steady-state groundwater flow problems involving rectangular grids. For example, for the 1K $\times$ 1K $\times$ 65 node problem, DPCG would have to run at 150 Gflops to solve the problem as quickly as multigrid. Our results further indicate that while the standard multigrid V-

cycle solver (MG) is very efficient for large problems with homogeneous or mildly heterogeneous K-fields, the multigrid preconditioned conjugate gradient solver (MGCG) is better suited for problems with stronger K-field heterogeneity. These findings are consistent with the findings of [2] which are for a finite-difference implementation of the groundwater flow problem.

9. References

[1] R. E. Alcouffe, A. Brandt, J. E. Dendy, Jr., and J. W. Painter. The multigrid method for the diffusion equation with strongly discontinuously coefficients. *SIAM J. Sci. Stat. Comput.*, 2:430–454, 1981.

[2] S. F. Ashby and R. D. Falgout. A parallel multigrid preconditioned conjugate gradient algorithm for groundwater flow simulations. Technical Report UCRL-JC-122359, Lawrence Livermore National Laboratory, CA, October 1995.

[3] S. F. Ashby, R. D. Falgout, S. G. Smith, and T. W. Fogwell. Multigrid preconditioned conjugate gradient algorithm for the numerical simulation of groundwater flow on the cray T3D. Technical Report UCRL-JC-118622, Lawrence Livermore National Laboratory, CA, September 1994.

[4] A. Behie and P. Forsyth, Jr. Multigrid solution of three-dimensional problems with discontinuous coefficients. *Appl. Math. Comp. Soc. Petr. Eng. J.*, 13:229–240, 1983.

[5] A. Brandt. Multilevel adaptive solutions to boundary-value problems. *Math. Comp.*, 31:311–329, 1977.

[6] W. L. Briggs. *A Multigrid Tutorial*. SIAM, Philadelphia, 1988.

[7] R. Barrett et al. *Templates for the Solution of Linear Syatems: Building blocks for iterative methods*. SIAM Publications, Philadelphia, 1993.

[8] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, 2nd edition, 1989.

[9] M. Holst, R. Kozack, F. Saied, and S. Subramaniam. Treatment of electrostatic effects in proteins: Multigrid-based Newton iterative method for solution of the full nonlinear Poisson-Boltzmann equation. *Proteins: Structure, Function, and Genetics*, 18(3):231–245, 1994.

[10] M. Holst and F. Saied. Multigrid solution of the Poisson-Boltzmann equation. *J. Comp. Chem.*, 14(1):105–113, 1993.

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

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

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

[14] T. J. McKeon and W.-C. Chu. A multigrid model for steady flow in partially saturated porous media. *Water Resources Research*, 23:542–550, 1987.

[15] P. D. Meyer, A. J. Valocchi, S. F. Ashby, and P. E. Saylor. A numerical investigation of the conjugate gradient method as applied to three-dimensional groundwater flow problems in randomly heterogeneous porous media. *Water Resources Research*, 25:1440–1446, 1989.

[16] A. B. F. Tompson, R. Aboubu, and L. W. Gelhar. Implementation of the three-dimensional turning bands random field generator. *Water Resources Res.*, 25(10):2227–2243, 1989.

[17] 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.

ORNL/TM-13441

INTERNAL DISTRIBUTION

1. A. S. Bland
2. E. F. D'Azevedo
3. J. P. Gwo
4. G. K. Jacobs
5. K. L. Kliewer
6. D. R. Mackay
7–11. G. Mahinthakumar
12. C. E. Oliver
13. B. A. Riley
14. O. Yasar
15. Laboratory Records - RC
16–17. Laboratory Records Department/OSTI
18. Central Research Library

EXTERNAL DISTRIBUTION

19. Daniel A. Hitchcock, ER-31, Acting Director, Mathematical, Information, and Computational Sciences Division, Office of Computational and Technology Research, Office of Energy Research, Department of Energy, Washington, DC 20585
20. Frederick A. Howes, ER-31, Mathematical, Information, and, Computational Sciences Division, Office of Computational and Technology Research, Office of Energy Research, Department of Energy, Washington, DC 20585
21. Tom Kitchens, ER-31, Mathematical, Information, and, Computational Sciences Division, Office of Computational and Technology Research, Office of Energy Research Department of Energy, Washington, DC 20585
22. David B. Nelson, ER-30, Associate Director, Office of Energy Research, Director, Office of Computational and Technology Research, Department of Energy, Washington, DC 20585

23–27. Faisal Saied, Department of Computer Science, University of Illinois, Urbana, IL 61801

28. Albert J. Valocchi, Department of Civil Engineering, University of Illinois, Urbana, IL 61801