The Parallel Performance of a Groundwater Flow Code on the Cray T3D*

S. F. Ashby† R. D. Falgout† S. G. Smith† A. F. B. Tompson‡

Abstract

This paper summarizes some preliminary results on the parallel performance of ParFlow, a new code for the numerical simulation of groundwater flow through three-dimensional heterogeneous porous media. At present, this code is capable of simulating groundwater flow for large sites (10^7 spatial zones) on a variety of distributed memory MIMD machines. The computational kernels are described briefly, and their parallel performance is examined on the Cray T3D massively parallel computer using Cray's implementation of the PVM message-passing library.

1 Introduction

Cleaning up polluted groundwaters is a daunting task facing government and industry alike. The U.S. Department of Energy, for instance, spends nearly one third of its budget on the cleanup of its contaminated sites; individual companies are spending tens of millions of dollars per year on the cleanup of contaminated facilities. The numerical simulation of subsurface fluid flow and contaminant migration plays an increasingly important role in the design and management of engineered remediation procedures [8]. Such simulations can be used, for example, to choose the best cleanup strategy for a given site, and then, once a scheme is chosen, to manage it in the most cost effective fashion.

Many of the computer codes in use today make unrealistic assumptions about the nature of the subsurface medium and the associated flow behavior. For example, many codes assume that the subsurface is homogeneous in composition and spatial distribution. In reality, the subsurface is three-dimensional and heterogeneous. These heterogeneities result in *preferential flow channels*, which can have a dramatic impact on flow and contaminant transport [1]. Consequently, the heterogeneous nature of the subsurface must be taken into account if one is to draw reliable conclusions from the numerical simulations.

The size of the site to be modeled (typically several square kilometers) and the need to resolve heterogeneities (on the order of meters) leads to computational domains with upwards of one billion spatial zones. Of course, the subsurface is impossible to characterize in this detail, and so hydrogeologists typically employ geostatistical techniques such as *turning bands* (see below) to create statistically accurate *realizations* of key subsurface properties, particularly the hydraulic conductivity [7]. Monte Carlo and optimization techniques can be used to quantify the inherent uncertainty and enable site managers to perform more realistic risk assessments. These techniques also can be used to evaluate various remediation strategies, say, determining the optimal pumping configuration in a pump-and-treat scheme.

---

*This work was supported in part by the Department of Energy (DOE) Defense Programs Technology Transfer Initiative, the Laboratory Directed Research and Development program at Lawrence Livermore National Laboratory (LLNL), the Army Corps of Engineers (SERDP), and the Applied Mathematical Sciences program of the DOE Office of Scientific Computing. It was performed at LLNL under contract W-7405-ENG-48.

†Center for Computational Sciences & Engineering, LLNL, L-316, Livermore, CA 94551

‡Earth Sciences Division, LLNL, L-206, Livermore, CA 94551

2 ParFlow Simulator

Computational environmental remediation is compute-intensive and quickly overwhelms all but the largest of conventional supercomputers. The need to repeatedly run large, time-dependent simulations necessitates the use of massively parallel processing power. Toward this end, we are building a parallel flow simulator called ParFlow. It is designed to be portable across a variety of distributed memory MIMD machines with message passing, ranging from workstation clusters to large MPPs. To achieve the best possible performance, we are building the code from scratch rather than porting an existing code. In addition to portability, the code is scalable (as demonstrated below) and extensible (via the use of C with some object abstraction).

Our initial focus has been on the accurate and efficient simulation of single phase flow problems. This is essential because these problems will constitute the main computational cost of the multiphase, time-dependent simulations in which we are ultimately interested. The mathematical model for single phase (saturated) steady-state flow is derived from a mass conservation equation and Darcy's law, and is given by

$$-\nabla \cdot (K\nabla(h+z)) - Q \;=\; 0 \tag{1}$$

where h is the pressure head (from which the flow velocity field is determined), K is the hydraulic conductivity (generated via a turning bands algorithm, for example), and Q is a source term (used to represent pumping wells, for example). At present, the problem domain is assumed to be a parallelepiped; the boundary conditions may be Dirichlet, Neumann, or mixed.

We solve for the pressure head on a discrete mesh. In the numerical investigations to date, we have used a single uniform grid (but with different steplengths, Δx, Δy, and Δz), and so load balancing is easy. A standard 7-point finite volume spatial discretization is used. In the case of saturated flow (our immediate interest), the discretized equations are linear, and so we obtain the pressure head from the solution of a large, sparse system of linear equations, $Ah = f$. The coefficient matrix A is symmetric positive definite and has the usual seven stripe pattern. The matrix has order $N = n_x \times n_y \times n_z$, where the n_i are the number of grid points in x, y, and z directions, respectively. For problems of interest, N is typically in the millions; the large number is dictated by the size of the physical site and the need to resolve heterogeneities adequately. We solve this system using preconditioned conjugate gradients, and then difference the pressure head to obtain the velocity field. This field is then passed to a transport code to simulate contaminant migration.

3 Parallel Implementation

The problem data is distributed across a virtual 3D process grid consisting of $P = p \times q \times r$ processes. The grid points within a process are arranged as a 3D subgrid, and the code uses a nested loop to access these points. This loop is our key computational kernel, and its efficient implementation is crucial as we will see in the next section. The computations are organized so as to avoid explicit data redistribution, thereby improving the code's efficiency. (This is one of the benefits of writing the code from scratch.) Although each process has a piece of the problem domain, we are *not* doing domain decomposition in the algorithmic sense. We are solving the *full* problem rather than independent subproblems. We have overlapped most of the code's communication with computations, thereby enhancing scalability on machines with message-passing macros that permit this.

We are using message-passing to realize portability across a variety of distributed memory MIMD computing platforms. At present, we are using AMPS (*a*nother *m*essage *p*assing *s*ystem), a message-passing layer derived from Zipcode [6] with additional input/output capabilities (e.g., parallel file read/write). This layer eventually will be replaced with the emerging MPI standard. By layering AMPS on top of PVM, Chameleon, the Reactive Kernel, and the IRIX IPC library, we have successfully run ParFlow (in various incarnations) on the following platforms: a single Sparcstation, a cluster of Sparcstations, a multiprocessor SGI Onyx, an nCUBE/2, an IBM SP-1, and the Cray T3D.

The generation of the hydraulic conductivity realization is central to the problem definition. Here we are using Tompson's turning bands algorithm [7], which is a technique for computing a spectral random field with given statistical properties (mean μ, variance σ^2, and correlation lengths

λ_x, λ_y, and λ_z). The algorithm has two phases: the generation of lines and the projection of grid points onto those lines. In the first phase, a number of rays through the origin (typically 50-100) are created, and one-dimensional random fields are generated along these. This requires cosines, which are more expensive than multiplications on RISC machines. (On the Cray T3D, the cosine function is about ten times slower.) The degree of parallelism here depends on the process grid topology: If the topology is too skewed, then each process may compute nearly an entire line, which degrades parallelism. Ideally, the process grid topology should match the topology of the discretized domain (see Table 1 below). In the second phase, these 1D fields are combined to produce a 3D random field. Specifically, each point is projected onto the lines, and these values are combined to determine the value of the 3D field at the given point. This projection phase is fully parallelizable, meaning that the computations are distributed equally across all processes with no redundancy. In contrast, some of the line generation computations may be replicated across one or more processes depending on the problem and the virtual process grid topology.

We have implemented several *preconditioned conjugate gradient* algorithms for solving the large linear systems that result from the discretization of (1). The algorithm is well known [4] and will not be repeated here; its key components are one matrix-vector multiplication, one preconditioning step, three vector updates, and two inner products. In this paper, we present results for diagonal, 2-step Jacobi, and multigrid preconditionings; the algorithms are denoted DSCG, J2CG, and MGCG, respectively. The multigrid preconditioner consists of a single V-cycle with semi-coarsening [2].

The coefficient matrix A is viewed as a stencil distributed across the processes analogous to the data distribution. To compute the matvec result at a given grid point (i, j, k), we "apply" the stencil to the grid: For each neighboring grid point specified by the stencil, we multiply the vector value at that point by the corresponding stencil coefficient, and then sum these products. This is equivalent to multiplying a row of the matrix by the vector x. By viewing the matrix-vector multiplication in this way, it is readily apparent which data needs to be communicated, namely, process boundary data. We therefore exchange interprocess boundary data at the start of a matvec. To facilitate this, there is a single layer of ghost points for storing the interprocess data; a single layer suffices because we have a 7-point stencil. Once each process has the data it needs, intraprocess matvecs are carried out in parallel. Of course, it is possible to overlap some of the communication with computation. Specifically, the intraprocess matvec can work on the process' internal mesh points first, and then update the boundary mesh points after the communication step is complete. For large problems on machines that allow overlapping of communication and computations, this communication will be concluded before the internal mesh points have been updated.

4 Parallel Performance

We now describe the parallel performance of the PARFLOW simulator on the Cray T3D. Our machine has 128 nodes, each consisting of a 150MHz DEC Alpha processor and 64MB of memory. The operating environment requires that one use a power of two number of processors, and we used the -O3 compiler optimization option. In these experiments, AMPS was layered on top of PVM, and this implementation performs poorly. We expect much better results when we replace PVM with Cray's SHMEM library, or when we replace AMPS with a native MPI port.

Our model problem corresponds to a physical domain of size $1260 \times 1260 \times 31$ m^3. The hydraulic conductivity field has parameters $\mu = 4$, $\sigma = 1.5$, and $\lambda_x = 40$, $\lambda_y = 40$, and $\lambda_z = 2$. We impose piecewise linear Dirichlet boundary conditions (for the pressure head) on the four vertical sides of the domain, and no flow conditions on the top and bottom. The PCG algorithm was halted once the 2-norm of the relative residual was less than 10^{-9}. Although this is a contrived test problem, it serves to illustrate the parallel performance of our algorithms. We are currently working with colleagues in the Laboratory's Environmental Protection Department to incorporate their model of the LLNL site into PARFLOW.

Importance of x-vector length

We first remark that code performance, especially that of the matvec, is extremely dependent on the number of data points per process in the x direction, which we call the x-vector length. This

TABLE 1

Cray T3D (64 processors) MFLOP Rates

TABLE 1A. $n_x \times n_y \times n_z = 129 \times 129 \times 65$

$p \times q \times r$	TBands	J2CG	MGCG
$1 \times 2 \times 32$	11.4	270.0	n/a
$1 \times 4 \times 16$	9.7	314.4	206.2
$1 \times 8 \times 8$	9.0	346.3	262.1
$4 \times 2 \times 8$	6.2	141.8	128.4
$4 \times 4 \times 4$	5.1	157.7	149.4
$8 \times 1 \times 8$	8.2	58.5	61.0
$8 \times 2 \times 4$	5.8	64.5	71.1
$8 \times 8 \times 1$	5.9	65.2	77.2

TABLE 1B. $p \times q \times r = 4 \times 4 \times 4$

$n_x \times n_y \times n_z$	J2CG	MGCG
$1025 \times 33 \times 33$	304.8	256.1
$257 \times 65 \times 65$	271.1	225.3
$129 \times 129 \times 65$	157.4	150.1
$129 \times 65 \times 129$	155.9	155.5
$65 \times 257 \times 65$	64.6	72.3
$65 \times 129 \times 129$	66.3	74.9
$65 \times 65 \times 257$	64.5	74.3

is because node performance depends on the number of points in the innermost loop of the matvec routine, and this corresponds to the x direction (recall the discussion above). We believe that this sensitivity is a result of caching issues on the T3D, but further investigation is needed. Thus, for best matvec and PCG node performance (ignoring communication), we want $p = 1$. Although this choice skews the process grid topology and results in more data being communicated than would a topology ratio of 1:1:1, it minimizes PVM overhead because there are fewer calls to pack routines.

In the experiments below, we choose $p = 1$ to optimize overall performance of the PCG algorithm. The parallelism of the turning bands line generation phase suffers when $p = 1$, but it is called just once per simulation, whereas the linear solve must be done at each time step (when we move to the time-dependent realm). Of course, it might be possible to rewrite the matvec routine (i.e., the innermost loop) and the communication routines to perform better, and we are exploring these options.

To see this behavior, examine Table 1A. Here we fix the problem size to be $129 \times 129 \times 65$ and vary the process grid topology for turning bands, J2CG, and MGCG. Notice the variation in MFLOP rates. The best rates are obtained when $p = 1$, which gives the longest possible x-vector lengths. We see the same effect in Table 1B, where we fix the process grid topology ($4 \times 4 \times 4$) and vary n_x, n_y, and n_z (but keep the product constant). Again, we see the importance of having long x-vectors. Although the MGCG MFLOP rate is less than that for J2CG, it converges much more rapidly. For example, on 64 processors, MGCG is an order of magnitude faster than J2CG. See [2] for a description of this algorithm and its performance.

Efficiency and scaled speedup graphs

In the next series of experiments we report efficiency and scaled speedup for the key components of ParFlow: turning bands generator, matrix-vector multiplication, multigrid preconditioning, and each of the PCG algorithms, DSCG, J2CG, and MGCG. In calculating the efficiency, we fixed the total problem size at $N = 64 \times 64 \times 32$ and measured the time, T_P, as the number of processes P increased. The efficiencies (given by T_1/PT_P) drop off sharply because the problem size per process is too small for larger P. (When $P = 128$, we have only 1024 grid points per process, and there is too little work for the amount of communication.) Scaled speedup is defined to be T_1/T_P, where now the total problem size is allowed to grow with P. In our experiments, each processor was given $2^{17} \approx 10^5$ grid points. To calculate the efficiency and scaled speedup graphs, it was necessary to vary the $p : q : r$ and $n_x : n_y : n_z$ ratios. As discussed above, this has a dramatic impact on node performance, which (at least in part) explains the erratic performance graphs. Since our goal is to minimize time to solution (rather than maximize MFLOP rates), we fixed $p = 1$ and let q and r vary as needed. If we had chosen $p : q : r$ to match $n_x : n_y : n_z$, which would have minimized communication, we probably would have seen smoother graphs, but the results would have had less relevance. Despite this, the algorithms are fairly scalable, meaning that the time per iteration remains constant as we increase the problem size and number of processes in tandem.

In Figures 1 and 2, we report efficiencies and scaled speedups for the turning bands, matrix-vector multiplication, and multigrid preconditioning routines. The turning bands performance is hindered by the line generation phase, which is a consequence of choosing $p = 1$. Recall that this phase is less parallel than the projection phase, and also requires cosines. We remark that matvec is the dominant cost in both DSCG and J2CG, but this is not true of MGCG. Here, the preconditioning step consists of a single V-cycle with semi-coarsening (to ameliorate anisotropy and/or skewed grid aspect ratios), and there is more communication than in DSCG and J2CG.

In Figures 3 and 4, we report efficiencies and scaled speedups for DSCG, J2CG, and MGCG. The poor efficiencies are attributable to two sources: (i) the inefficiency of the message-passing routines, and (ii) the fact that for large P, the size of the subproblem per process is too small. The scalability curves of DSCG and J2CG are similar; both drop off as the number of processes increases. MGCG, on the other hand, appears to be more scalable. We believe that we can improve the performance (i.e., flatten and raise the scalability curves) of all three algorithms by using a faster communication library (e.g., native MPI).

Acknowledgements

We acknowledge the valuable participation of Charles Baldwin, John Bell, William Bosl, Thomas Fogwell, Gwen Loosmore, and John Ziagos in the PARFLOW project. In particular, we thank Dr. Thomas Fogwell for his helpful comments and guidance concerning multigrid.

The Cray T3D experiments described in this paper were run on the 128-node machine located in the Center for Computational Sciences and Engineering at Lawrence Livermore National Laboratory as part of the H4P technology transfer initiative funded by DOE Defense Programs. Early code development was done on the Cray T3D located at the Pittsburgh Supercomputer Center.

References

[1] R. ABABOU, D. B. MCLAUGHLIN, L. W. GELHAR, AND A. F. B. TOMPSON, *Numerical simulation of three-dimensional saturated flow in randomly heterogeneous porous media*, Transport in Porous Media, 4 (1989), pp. 549–565.

[2] S. F. ASHBY, R. D. FALGOUT, S. G. SMITH, AND T. W. FOGWELL, *Multigrid preconditioned conjugate gradients for the numerical simulation of groundwater flow on the CRAY T3D*, Tech. Report UCRL-JC-118622, Lawrence Livermore National Laboratory, 1994. Submitted to the proceedings of the ANS International Conference on Mathematics and Computations, Reactor Physics, and Environmental Analyses, April 30–May 4, 1995, Portland, OR.

[3] S. F. ASHBY, R. D. FALGOUT, S. G. SMITH, AND A. F. B. TOMPSON, *Modeling groundwater flow on MPPs*, in Proc. 1993 Scalable Parallel Libraries Conference, A. Skjellum and D. Resse, eds., IEEE Computer Society Press, 1994, pp. 17–25. Held at Mississippi State University, October 6–8, 1993. Also available as LLNL technical report UCRL-JC-115602.

[4] P. CONCUS, G. H. GOLUB, AND D. P. O'LEARY, *A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations*, in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, New York, 1976, pp. 309–332.

[5] 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 Res., 25 (1989), pp. 1440–1446.

[6] A. SKJELLUM, S. G. SMITH, N. E. DOSS, A. P. LEUNG, AND M. MORARI, *The design and evolution of Zipcode*, Parallel Computing, 20 (1994), pp. 565–596.

[7] A. F. B. TOMPSON, R. ABABOU, AND L. W. GELHAR, *Implementation of of the three-dimensional turning bands random field generator*, Water Resources Res., 25 (1989), pp. 2227–2243.

[8] A. F. B. TOMPSON, S. F. ASHBY, R. D. FALGOUT, S. G. SMITH, T. W. FOGWELL, AND G. A. LOOSMORE, *Use of high performance computing to examine the effectiveness of aquifer remediation*, in Proc. X International Conference on Computational Methods in Water Resources, A. Peters, G. Wittum, B. Herrling, U. Meissner, C. Brebbia, W. Gray, and G. Pinder, eds., vol. 2, Kluwer Academic Publishers, Dordrecht, 1994. Held in Heidelberg, Germany, July 19-22, 1994. Also available as LLNL technical report UCRL-JC-115374.

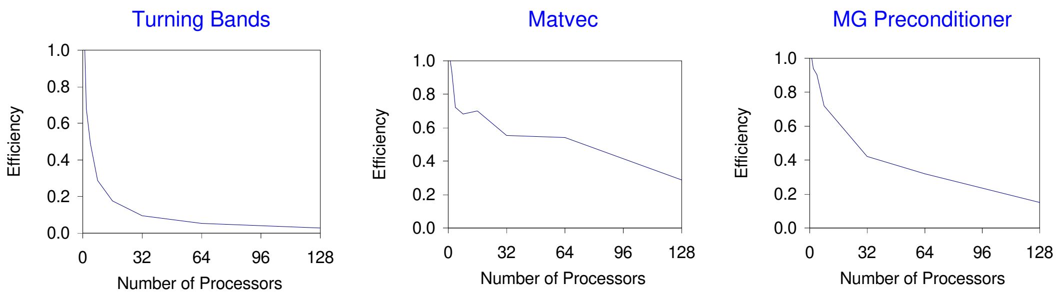

FIG. 1. *Parallel efficiency on the Cray T3D.*

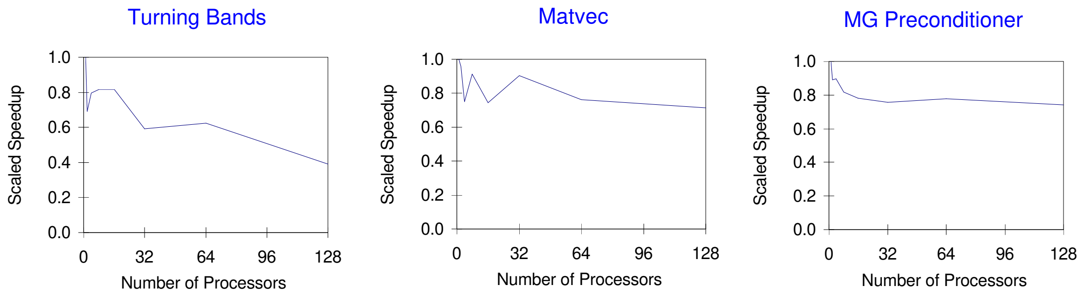

FIG. 2. *Scaled speedup on the Cray T3D.*

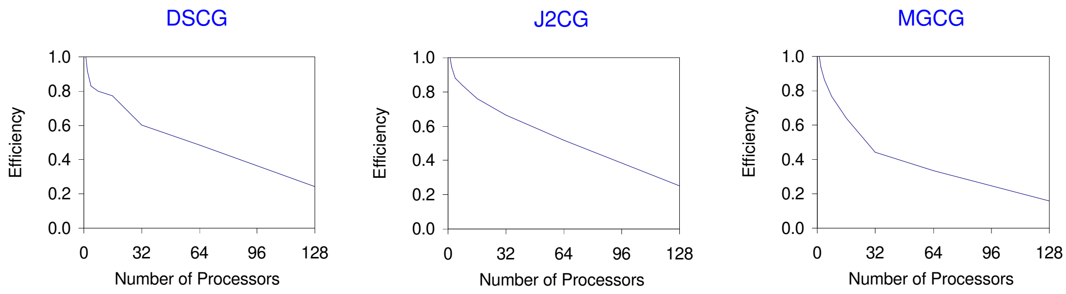

FIG. 3. *Parallel efficiency on the Cray T3D.*

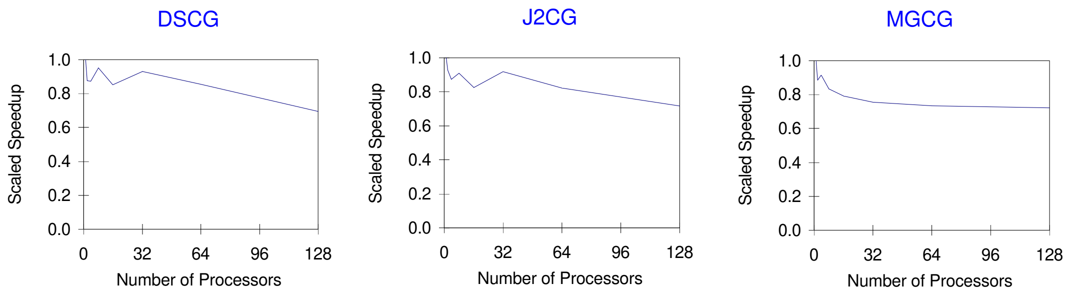

FIG. 4. *Scaled speedup on the Cray T3D.*