

IMPLEMENTATION AND PERFORMANCE ANALYSIS OF A PARALLEL MULTICOMPONENT GROUNDWATER TRANSPORT CODE[†]

*G. Mahinthakumar, Center for Computational Sciences, Oak Ridge National Laboratory., phone:
423-241-5628, fax: 423-241-2850, e-mail: kumar@ornl.gov.*

*F. Saied, National Center for Supercomputing Applications (NCSA), Urbana, IL,
phone: 217-244-9481, fax: 217-244-2909, e-mail: fsaied@ncsa.uiuc.edu.*

ABSTRACT

We describe the implementation and analyze the performance of a parallel finite-element multi-component groundwater transport code on a variety of parallel architectures. The code exhibits characteristics that are typical to many simulation codes such as explicit communication, global reduction operations, sparse matrix operations, and parallel I/O. The parallel implementation is based on domain decomposition with explicit message passing using MPI. We analyze performance on architectures such as the Intel Paragon, IBM SP, Origin 2000, Cray T3E, and Convex Exemplar SPP-2000. One of our goals is to investigate performance metrics that are based on an entire application code as opposed to kernels that perform specific operations. For the results presented here the performance analysis is mainly limited to scalability and cache effects. We emphasize here that this paper primarily focuses on the parallel performance issues and details regarding the application has been presented elsewhere.

Our results show that the implementation is scalable on architectures which have a good ratio (> 2) of communication bandwidth (MB/sec) to peak performance (Mflops). The single node performance is mainly affected by memory bandwidth and secondary cache size because sparse matrix operations dominate the computations. On machines such as the Origin 2000 which have a good sized (4 MB) secondary cache we achieve better percentage of the peak for small to moderate size problems than machines which have no secondary cache. We have implemented certain computational and memory saving features into this code which enables enhanced resolution of the model with rapid solution times. We are now able to solve nonlinear coupled partial differential equation systems with more than 100 million degrees of freedom in about 5-10 seconds per time step on machines such as the 1024-processor Intel Paragon XPS/150. To our knowledge, solution to groundwater transport problems of this size has not been reported before in literature.

BACKGROUND

Distributed memory implementation of finite-element codes is generally based on domain decomposition [e.g. Fox, 1988; Gropp and Keyes, 1988] or element-by-element strategies [Tezduyar and Liou, 1989]. Element-by-element strategies which benefit from reduced communication and higher floating point performance at the expense of increased storage and floating point operations are mainly adapted for unstructured applications [e.g. Tzeduyar et al., 1993, Mahinthakumar and Hoole, 1990]. We apply the domain decomposition strategy for parallelization of the structured finite-element application that is considered here. A similar strategy that was used in the flow module of the same suite of codes has been published in Saied and Mahinthakumar, 1998. Our other past related work includes cross-platform solver performance analysis for the single-component transport problem [Mahinthakumar et al., 1997], and parallel I/O study of the same suite of codes [Mackay

[†]An updated version of this paper will be available for viewing at <http://www.ccs.ornl.gov/staff/kumar/siam99.html>

et al., 1998]. The work that is being reported here is different in many aspects in that we are investigating multicomponent transport as opposed to single component transport, improved memory and computational savings features for multicomponent transport, and we are analyzing the performance of the entire code as opposed to solver or I/O performance.

The multicomponent groundwater transport code that is the focus of this investigation is being used in a variety of complex groundwater transport and remediation applications [Mahinthakumar, 1998]. The general system of equations describing transport of nc dissolved components undergoing reactions in saturated porous media is defined by a nonlinear time-dependent coupled partial differential equation (p.d.e) system given by

$$\frac{\partial C_i}{\partial t} = \nabla \cdot (\mathbf{D} \cdot \nabla C_i) - \nabla \cdot (C_i \mathbf{v}) + \frac{q}{\theta}(C_i - C_{0i}) - R_i \quad i = 1, 2, 3, \dots, nc \quad (1)$$

where \mathbf{v} is the 3x1 velocity field vector, \mathbf{D} is the 3x3 dispersion tensor dependent on \mathbf{v} , and C_i is the dissolved concentration of component i . The term $q(C_i - C_{0i})/\theta$ represents the source term with volumetric flux q , medium porosity θ , and injected concentration C_{0i} (e.g. from injection wells). R_i is the rate of mass loss of component i due to sorption and bioremediation reactions and is the main coupling term for the system of equations. The term R_i may contain many terms and can be nonlinear. For example, if only bioremediation reactions are present then R_i is given by

$$R_i = \mu_{max} F_i X \prod_{j=1}^{j=nc} f_{ji} \left(\frac{C_j}{K_j + C_j} \right) \quad i = 1, 2, 3, \dots, nc \quad (2)$$

Where F_i is the stoichiometric ratio, X is the biomass concentration, μ_{max} is the maximum utilization rate, and f_{ji} is a factor controlling component j 's contribution to component i 's biodegradation process. If $f_{ji} = 0$ then component j does not participate in component i 's biodegradation process.

IMPLEMENTATION

The system of equations (1) are discretized using the Galerkin finite element method with 8-node linear hexahedral elements. A logically rectangular grid structure is assumed but irregular geometries are supported using distorted elements. A Crank-Nicolson approximation (central finite-difference) is used for the time derivative terms. A lumped mass formulation [Huyakorn and Pinder, 1983] is used for all time-derivative and non-derivative (zeroth spatial derivative) terms. The coupled nonlinear system described by (1) is solved using a modified form of the Sequential Iterative Algorithm (SIA) with a semi-explicit method to decouple the system [Mahinthakumar et al., 1999]. In each iteration a full matrix solve of the decoupled system is performed for the linear system arising from equations (1). Iterations are performed until the fully coupled system is satisfied (typically 4-5 iterations per time step for a 6 component system). A diagonal storage format is used for the global matrices arising for each component of the decoupled system. We have found that the matrix data storage format had different effects on different architectures and here we have adapted a format that performed reasonably well on most architectures. We have implemented BiCGSTAB, GMRES(k), ORTHOMIN(k), and CGS iterative solvers for the matrix solution and the comparison of these solvers are discussed in Mahinthakumar et al. (1997). Our performance results in this paper will be based on the BiCGSTAB solver which performs reasonably well for most problems.

Because we use a lumped mass formulation to describe the coupling terms (e.g. equation (2)), the coupling terms occur only on the main diagonals of the off diagonal blocks of the full matrix. We have

achieved considerable memory and computational savings by not replicating the storage or computation of matrix entries that are common to all components and updating only those entries which change from time step to time step. Based on the user input data the code will determine which matrix entries are common to all components and which entries change for the subsequent time step. In most multicomponent transport scenarios tremendous savings can be achieved by carefully tracking these needs. For example, for the six component transport system we used for our simulations we were able save a factor of 4 in matrix storage and a factor of 2 in matrix computations compared to the standard approach which caters to the worst case scenario. This implementation has enabled us rapid solution of a large number of components at a much higher resolution than previously possible. The combination of lumped mass formulations, decoupling of the full matrix, and other special treatments for memory and computational savings has one minor drawback: this restricts us from reordering the full sparse matrix to form dense sub blocks which may result in better floating point performance. Therefore we lose some of the advantages that are traditionally associated with a vector p.d.e (multicomponent transport) as opposed to a scalar p.d.e (e.g. single component transport).

Parallel Implementation

For the parallel implementation we use a two-dimensional (2-D) domain decomposition in the x and y directions (for more details see Saied and Mahinthakumar [1998]). A 2-D decomposition is generally adequate for groundwater problems because common groundwater aquifer geometries involve a vertical dimension that 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. 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 some I/O performance degradation particularly when a large number of processors were involved. All explicit communications between neighboring processors are performed using asynchronous MPI calls. System calls were used for global communication operations such as those used in dot products. Parallel I/O is performed using MPI-IO calls (ROMIO) or optionally have a single processor read/write data through message passing and a temporary buffer. The single processor I/O option was built for portability on systems where MPI-IO could not be used with native MPI. Parallel I/O using MPI-IO has been tested on the Origin 2000 and Intel Paragon systems. Recently we have incorporated Open MP directives in the solver portion of the code. Options have been built to use the code in a pure Open MP mode, a pure MPI mode, or a hybrid mode. All three modes have been tested on the Origin 2000 and a 4-processor shared memory Intel Xeon system. We will not be presenting these results here since the results are

very preliminary. The codes are mainly written in Fortran 77 (with some C) using double-precision arithmetic.

ARCHITECTURES

Our results are presented for the following architectures: Intel Paragon XPS/150 at ORNL, Cray/SGI Origin 2000 at NCSA, Cray T3E at NERSC, IBM SP at ANL, and Convex Exemplar SPP-2000 at NCSA. The following table shows the main features of each architecture.

Feature	Paragon	Origin 2000	Cray T3E	IBM SP	Convex
clock speed (MHZ)	50	250	450	62.5	180
peak performance per node (Mflops)	75 or 150	500	900	125	360
memory bandwidth (MB/sec)	170	720	1200	NA	NA
communication bandwidth (MB/sec)	152	NA	600	35	NA
communication latency (μ s)	35	NA	NA	63	NA
number of processors available to user	1024	128	512	64	16
data cache (L1)	16 KB	32 KB	8 KB	32 KB	1 MB
instruction cache (L1)	16 KB	32 KB	8 KB	32 KB	1 MB
secondary cache (L2)	none	4 MB	96 KB	NA	none
Memory per node (MB)	64	256	256	128	256

Table 1: Main features of each architecture (NA = not available).

PERFORMANCE RESULTS

Our performance results are based on simulations with model parameters taken from a field scale bioremediation problem reported by Sempirini and McCarty (1992) involving the aerobic biodegradation of six organic components (TCE, VC, t-DCE, c-DCE, DO, and CH_4). The problem sizes chosen here is purely for performance analysis purposes and is not representative of the field problem. The results presented here will be limited to scalability and cache effects. Detailed comparisons of different solvers used in this code and a parallel I/O performance analysis have been presented elsewhere [Mahinthakumar et al., 1997, Mackay et al., 1998]. For tests performed here we have used options so that very minimal I/O is performed (no check pointing and concentration field output only at the final timestep).

Overall Scalability

In Figure 1 we compare the scalability of the entire code up to 64 processors. The Origin 2000 numbers are for the 250 Mhz processor (see Table 1). The total time includes I/O, matrix assembly, and matrix solution. The local problem size is fixed at $41 \times 41 \times 11$. Note that perfect scalability would correspond to a horizontal line. Scalability is very good on the Intel Paragon, and Cray T3E and reasonable on the IBM SP. The Paragon results did not surprise us since it has a very good communication to CPU performance ratio. Even though the Origin 2000 performed very well in the single-node performance category (to be addressed later), it did not exhibit good scalability at 64 processors. We believe this is due to increasing message contention which was not handled well by the Origin 2000.

Detailed analysis of communication times indicated that the scalability and parallel performance are not affected by latency on all the machines. Furthermore, scalability on the Origin 2000 dramatically improved (results not presented here) when the local problem size was reduced, thus indicating that message sizes and message contention (affected by bandwidth) has a more profound impact on communication performance than the number of messages (affected by latency).

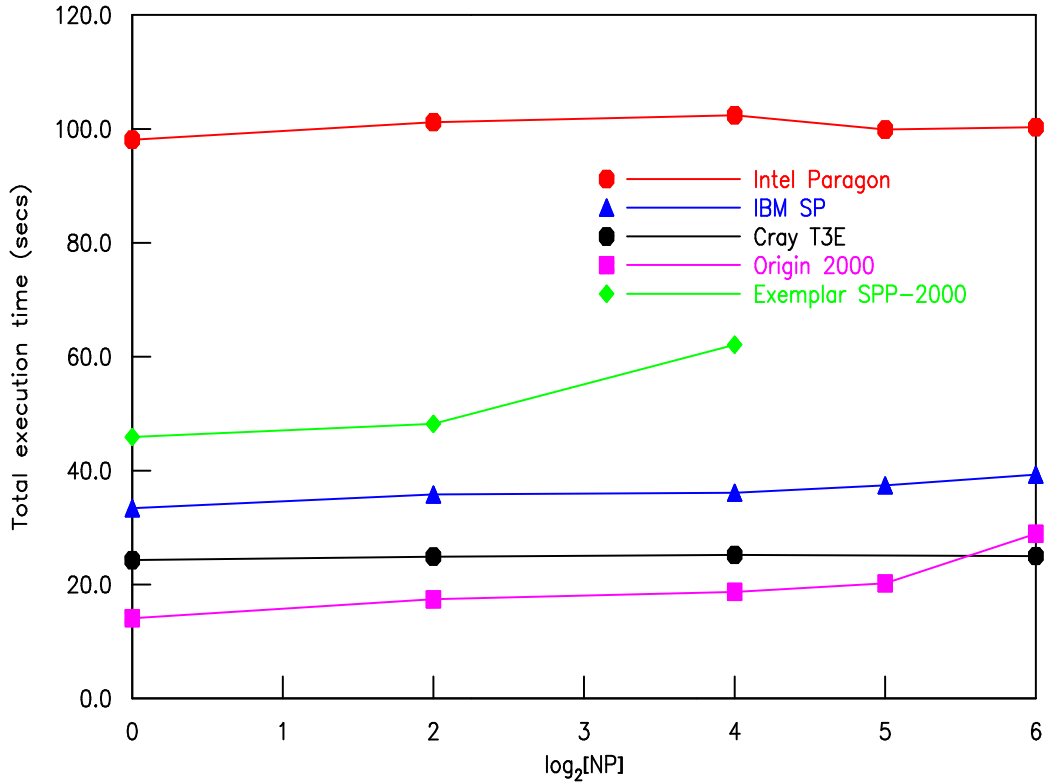


Figure 1: Cross-platform scalability up to 64 processors. NP = number of processors. Local problem size is fixed at 41x41x11. Six transporting species are simulated to 10 time steps with approximately 110,000 degrees of freedom per processor. Total execution time includes all operations of the code.

Solver performance

The megaflop performance was not measured for the entire code but only for the BICGSTAB solver. The 'ssrun' utility on the Origin 2000 indicated that the solver performance is indicative of the overall performance for moderate number of processors if minimal I/O is performed. The measured 64 processor solver performance is as follows: 890 Mflops on the Intel Paragon, 1.87 Gflops on the IBM SP, 4.25 Gflops on the Origin 2000, and 2.78 Gflops on the T3E. This is about 10–15% of the peak performance except for the T3E which is about 5%. The 10–15% range is not unusual for sparse matrix applications. It should be noted that we did not perform any special tuning other than compiler optimization for the single node performance of the code for each architecture.

Intel Paragon results for large number of processors

In Figure 2 we show the scalability up to 1024 processors of the Intel Paragon XPS/150. We note here that the code is run in single threaded mode on the MP-nodes of the Paragon due to memory bandwidth limitations in the sparse matrix operations. The peak performance in single-threaded mode is about 75 Gflops on 1024 nodes. Excellent scalability is observed up to 512 processors and

reasonable scalability up to 1024 processors. Here we also show timings for the dominating components of the code, matrix solution (BiCGSTAB solver) and matrix assembly/computation. Communication time which includes explicit message passing and global reduction operations is also shown. In general, explicit message passing takes about 65% of the total communication time. In the explicit message passing time we have included temporary buffering operations (see parallel implementation section) where memory to memory copies are made. Since the results shown here are only up to 10 time steps, matrix assembly and computation takes about 34% of the total time and matrix solution takes about 57% of the total time. As we increase the number of time steps matrix solution time begins to dominate even more (up to 72%) and matrix assembly is less dominant (20%). This is primarily because in our implementation most of the assembly computations are performed in the first time step and in subsequent time steps only updates of changing entries are performed.

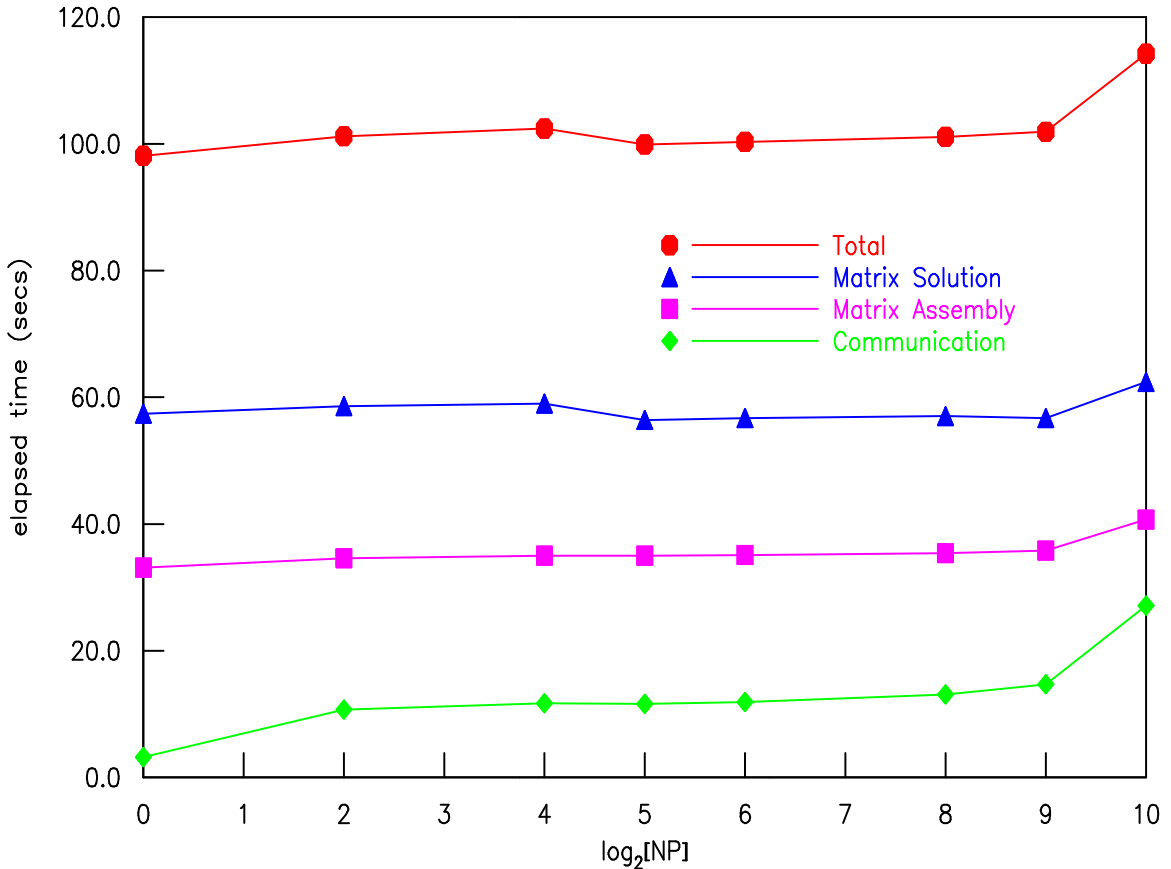


Figure 2: Paragon scalability up to 1024 processors. NP = number of processors. Local problem size is fixed at 41x41x11. All parameters same as Figure 1.

The largest problem we have solved on the 1024 node Paragon is a 1401x1401x11 problem (21.5 million nodes) with 6 transporting components (130 million unknowns). In the number of unknowns we only include moving components which require full matrix solves. The other components such as biomass and sorbed phase concentrations are not included. The total time for 100 time steps of this simulation is 733 secs. This averages to about 7.33 secs per time step which includes several matrix solves for nonlinear iterations. As we increase the number of time steps to 1000 the average time reduces to 5.4 seconds. Such large problems and rapid solution times are only possible with the memory and computational saving features we have implemented in this code. The overall perfor-

mance of the code is about 10 Gflops and the matrix solution portion achieves about 12.5 Glops (20% of peak).

Cache effects on the Origin 2000

Of all the architectures we have tested only Origin 2000 had any significant secondary cache (4 MB) to examine the cache effects. Since the matrix solution portion dominates the computation for most practical simulations we restrict this analysis to the BiCGSTAB solver which has been used in these tests. The most dominating component (over 80%) of the solver is the sparse matrix multiplication operation. Since cache effects primarily affect the single node performance our tests are done on a single CPU of the Origin 2000 (peak performance of 500 Mflops).

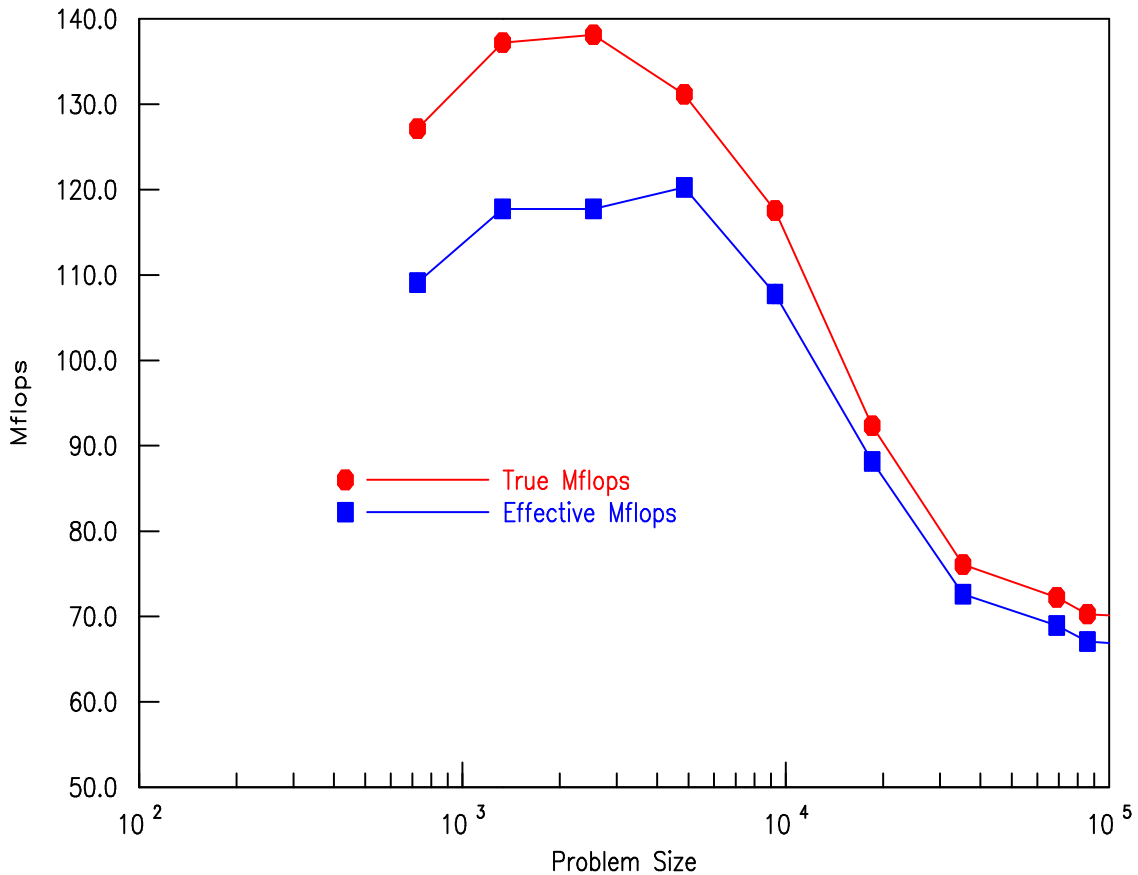


Figure 3: Effect of cache on the single node performance of the BICGSTAB matrix solver on the Origin 2000. See text for explanation of 'true' and 'effective' mflops.

In Figure 3 we show the single node performance of the solver as we increase the problem size. In this figure, "True" Mflops indicate the floating point performance based on the exact number of floating point operations performed and "Effective" Mflops indicate performance based on the useful number of operations. Since we use padded arrays with "ghost" entries to simplify parallel implementation each processor performs dummy operations on these entries to ensure contiguity in memory access and thereby improve performance. The best performance of 130 – 140 Mflops (25% of peak) is achieved for problem sizes around 3000 and as the problem size increases to about 100000 the performance levels off to about 70 Mflops. Since we have about 4 MB of cache the entire matrix and the associated vectors can easily fit in the cache if the problem size is less than 3000. We note here

that we are solving transport of 6 components and if we did not implement the memory saving features mentioned earlier and decided to store the fully coupled matrix of all 6 components we would not have benefitted as much from the available cache.

CONCLUSIONS

Our preliminary results show that the implementation is scalable on architectures which have a good ratio (> 2) of communication bandwidth (MB/sec) to peak performance (Mflops). For example, on the Intel Paragon XPS/150 we observe almost perfect scalability up to 512 processors (less than 5 % loss) and reasonable scalability up to 1024 processors (about 15 % loss). The single node performance is mainly affected by memory bandwidth and secondary cache size because sparse matrix operations dominate the computations. In terms of floating point performance the single node performance is on the order of 10–15 % of peak (typical for sparse matrix computations) on most architectures. On machines such as the Origin 2000 which have a good sized (4 MB) secondary cache we achieve better percentage of the peak performance for small to moderate size problems than machines which have no secondary cache.

The special memory and computational savings that have been implemented in these codes enable us to solve nonlinear coupled partial differential equation systems with more than 100 million degrees of freedom in about 5–10 seconds per time step on machines such as the 1024–processor Intel Paragon XPS/150. To our knowledge, solution to groundwater transport problems of this size has not been reported before in literature. In this implementation we have not included geochemistry reactions which is a much easier problem from a performance point of view since the computations are local in nature (no communication) involving small dense matrices (better cache reuse). We expect the overall floating point performance to increase significantly (25–30 % of peak) when geochemical reactions are included in our multicomponent transport model.

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–AC05–96OR22464. The authors gratefully acknowledge the use of High Performance Computing Facilities at the Center for Computational Sciences of Oak Ridge National Laboratory, Mathematics and Computer Science Division of the Argonne National Laboratory, National Energy Research Supercomputing Center (NERSC), and the National Center for Supercomputing Applications (NCSA). The authors thank Dr. Pat Worely of ORNL for his assistance in the T3E runs at NERSC.

REFERENCES

- Fox, G.C., **Domain decomposition in distributed and shared memory environments 1. A uniform decomposition and performance analysis for the NCUBE and JPL Mark hypercubes.**, *Lect Notes Comput Sci* 297: 1042–1073, 1988.
- Gropp W. D., and D.E. Keyes, **Complexity of parallel implementation of domain decomposition techniques for elliptic partial differential equations.**, *SIAM J. Sci Stat Comp.*, 9(2): 312–326, Mar 1988.
- Huyakorn, P.S., and G.F. Pinder, *Computational Methods in Subsurface Flow*, Academic Press, New York, N.Y., 1983.
- Mackay, D., E. F. D’Azevedo, and G. Mahinthakumar, 1998., **A study of I/O in a parallel finite–element groundwater transport code**, *Int. J. of High Performance Computing Applications*, 12(3), Fall 1998, p. 307–319.

- Mahinthakumar, G., Caroline Tebes–Stevens, Albert J. Valocchi, and Faisal Saied, **Efficient solution of large scale multicomponent transport systems**, *1999 SIAM Meeting on Computational Issues in Geosciences (GS99)*, San Antonio, Texas, March 1999.
- Mahinthakumar, G., 1998. **PGREM3D: Massively Parallel Codes for Groundwater Transport and Remediation**, *Oak Ridge National Laboratory Technical Report* (under review), ORNL/TM–13435.
- Mahinthakumar, G., and S. R. H. Hoole, **A Parallelized Element–by–Element Jacobi Conjugate Gradients Algorithm for Field Problems and a Comparison with Other Schemes**, *Applied Electromagnetics in Materials*, Vol. 1, No. 1, 15–28, Jul 1990.
- Mahinthakumar, G., F. Saied, and A. J. Valocchi, **Comparison of some parallel krylov solvers for large scale contaminant transport simulations**, *High Performance Computing 1997* (Editor: A. M. Tenner), Proceedings of the 1997 Simulation Multiconference, p. 134–139, Atlanta, GA, Apr 6–10, 1997.
- Tezduyar, T.E., and Liou J., **Grouped element–by–element iteration schemes for incompressible flow computations**, *Comp. Phys. Commun.* 53: (1–3) 441–453 May 1989.
- Tezduyar, T., Aliabadi S., Behr M., Johnson A., and Mittal S., **Parallel Finite element computations of 3D flows**, *Computer* 26: (10) 27–36, October 1993.
- Saied, F., and G. Mahinthakumar, **Efficient Parallel Multigrid Based Solvers for Large Scale Groundwater Flow Simulations** *Computers Math. Applic.*, Vol. 35, No. 7, p. 45–54, 1998.