PROCEEDINGS, TOUGH Symposium 2003
Lawrence Berkeley National Laboratory, Berkeley, California, May 12–14, 2003

TOUGH2_MP: A PARALLEL VERSION OF TOUGH2

Keni Zhang, Yu-Shu Wu, Chris Ding, and Karsten Pruess

Earth Sciences Division
Lawrence Berkeley National Laboratory
Berkeley, CA, 94720, USA
e-mail: kzhang@lbl.gov

ABSTRACT

TOUGH2_MP is a massively parallel version of TOUGH2. It was developed for running on distributed-memory parallel computers to simulate large simulation problems that may not be solved by the standard, single-CPU TOUGH2 code. The new code implements an efficient massively parallel scheme, while preserving the full capacity and flexibility of the original TOUGH2 code. The new software uses the METIS software package for grid partitioning and AZTEC software package for linear-equation solving. The standard message-passing interface is adopted for communication among processors. Numerical performance of the current version code has been tested on CRAY-T3E and IBM RS/6000 SP platforms. In addition, the parallel code has been successfully applied to real field problems of multi-million-cell simulations for three-dimensional multiphase and multicomponent fluid and heat flow, as well as solute transport. In this paper, we will review the development of the TOUGH2_MP, and discuss the basic features, modules, and their applications.

INTRODUCTION

TOUGH2 (Pruess, 1991) is a general-purpose numerical simulation program for modeling multi-dimensional, multiphase, multicomponent heat and fluid flows in porous and fractured media. The code is written in standard ANSI FORTRAN 77 and in a modular structure. TOUGH2_MP was developed based on TOUGH2 version 1.4 (Wu et al., 1999).

The parallel prototype scheme for the TOUGH2 code was first developed by Elmroth et al. (2001). Zhang et al. (2001) implemented more modules, and further optimized the code for solving extremely large simulation problems by improving memory use and computation efficiency. The current version includes four modules: EOS2, EOS3, EOS9, and T2R3D. It has been successfully applied to real-field large-scale simulation problems (Wu et al., 2002).

To parallelize the TOUGH2 code, a new massively parallel computing method for conducting large-scale multiphase and multicomponent reservoir simulation was developed. The method uses the METIS software package (Karypsis and Kumar, 1998) for partitioning the unstructured computational domain, the AZTEC software package (Tuminaro et al., 1999) for solving linear equations, and the standard message-passing interface (MPI, Message Passing Interface Forum, 1994) for communication among processors. The primary objective of the new development is to present a machine- or platform-independent parallel algorithm that can be easily implemented into a mainframe supercomputer, a multiprocessor workstation, or a cluster of PCs or workstations. Secondly, this work is intended to overcome numerical problems with the efficiency and robustness characteristic of the developed parallel-computing technology for handling severely nonlinear problems. These goals are achieved by integrating and optimizing the following procedures: (1) efficient domain partitioning; (2) parallel Jacobian matrix calculations; (3) parallel-solving linearized equation systems; (4) fast communication and data exchange between processors; and (5) efficient memory sharing among processors.

This paper reviews this parallel version of the TOUGH2 family of codes and demonstrates its applications to large-scale, real-world simulation problems.

DEVELOPMENT OF THE TOUGH2_MP

The numerical scheme of the TOUGH2 code is based on the integral finite difference (IFD) method (Narasimhan and Witherspoon, 1976). Conservation equations involving air, water, and chemical components as well as thermal energy are discretized in space using the IFD method. Time is discretized fully implicitly using a first-order backward finite-difference scheme. The resulting discretized finite-difference equations for mass and energy balances are nonlinear and are solved simultaneously using the Newton/Raphson iterative method.

The TOUGH2_MP is implemented with dynamic memory management, modules, array operations, matrix manipulation, and other FORTRAN 90 features. Some important modifications to the original TOUGH2 code are also made in the time-step looping subroutine. This subroutine now

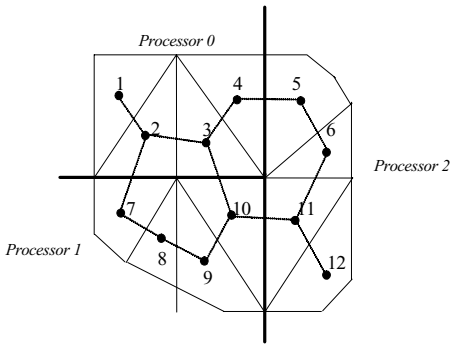
provides general control of problem initialization, grid partitioning, data distribution, and memory requirement balancing among all processors (in addition to time stepping and output options). In the following sections, we will discuss the most important implementations in the parallel code, including domain partitioning, distribution of memory requirements, parallel assembly of Jacobian matrix, parallel solving of linear equations, and message passing.

Domain Partitioning and Gridblock Reordering

Developing an efficient and effective method for partitioning unstructured grid domains is a first and critical step for a successful parallel scheme. To obtain optimal performance, the partitioning algorithm should ideally take the following five issues into account: (1) Evening the computational load balance; (2) Minimizing the average (or total) communication volume; (3) Evening the load balance in communication volume; (4) Minimizing the average (or total) number of neighboring processors; (5) Evening the load balance in number of neighboring processors. To find an optimal trade-off between these five issues, computer system characteristics, such as floating-point performance and bandwidth and latency of the communication subsystem, must be taken into account. Because this is a very complex problem, commonly used algorithms and software for partitioning large grids do not generally take all these five issues into account. The current practice typically finds a trade-off between computation load balance and low total communication volume, even though they may not be theoretically optimal. More discussion of these issues is given in Elmroth (2000).

In a TOUGH2 simulation, a model domain is represented by a set of three-dimensional gridblocks (or elements) and the interfaces between any two gridblocks are represented by connections. The entire connection system of gridblocks is treated as an unstructured grid. From the connection information of a gridblock, an adjacency matrix can be constructed. The adjacency or connection structure of a model grid is stored in a compressed storage format (CSR). Figure 1a shows the connection of a 12-element domain; Figure 1b illustrates its corresponding CSR-format arrays.

We utilize three partitioning algorithms of the METIS package (version 4.0) (Karypsis and Kumar, 1998) for partitioning a grid domain. The three algorithms are here denoted as the *K-way*, the *VK-way*, and the *Recursive* partitioning algorithms. *K-way* is used for partitioning a global mesh (graph) into a large number of partitions (more than 8). The objective of this algorithm is to minimize the number of edges that straddle different partitions. If a small number of partitions are desired, the *Recursive* partitioning method, a recursive bisection algorithm, should be used. *VK-way* is a modification of *K-way*, and its objective is to minimize the total communication volume. Both *K-way* and *VK-way* are multilevel partitioning algorithms.



(a) A 12-elements domain partitioning on 3 processors

Elements		1	2	3	4	5	6	7	8	9	10	11	12
xadj	1	2	5	8	10	12	14	16	18	20	23	26	27
adj		2	1,3,7	2,4,10	3,5	4,6	5,11	2,8	7,9	8,10	3,9,11	6,10,12	11

(b) CSR format

Figure 1. An example of domain partitioning and CSR format for storing connections.

Figure 1a shows a scheme for partitioning a sample domain into three parts. Gridblocks are assigned to particular processors through partitioning methods and reordered by each processor to a local index ordering. Elements corresponding to these blocks are explicitly stored in the processor and are defined by a set of indices referred to as the processor's *update* set. The *update* set is further divided into two subsets: *internal* and *border*. Elements of the *internal* set are updated using only the information on the current processor. The *border* set consists of blocks with at least one edge to a block assigned to another processor. The *border* set includes blocks that would require values from other processors to be updated. The set of blocks not in the current processor, but needed to update components in the *border* set, is referred to as an *external* set. Table 1 shows the partitioning results for the sample problem presented in Figure 1a.

Local numbering of gridblocks is carried out to facilitate the communication between processors. The numbering sequence is *internal* blocks, followed by *border* blocks, and finally by the *external* set. In addition, all *external* blocks from the same processor are in consecutive order (see Table 1).

Only nonzero entries of a submatrix for a partitioned mesh domain are stored in the processor. Each processor stores only the rows that correspond to its

update set. These rows form a submatrix whose entries correspond to variables of both the *update* set and the *external* set defined on this processor.

Table 1. Example of domain partitioning and local numbering.

		Update		External
		Internal	Border	
PE0	Gridblocks	1	2 3 4	5 7 10
	Local Numbering	1	2 3 4	5 6 7
PE1	Gridblocks	8 9	7 10	2 3 11
	Local Numbering	1 2	3 4	5 6 7
PE2	Gridblocks	6 12	5 11	4 10
	Local Numbering	1 2	3 4	5 6

Organization of Input and Output Data

The input data include hydrogeologic parameters and constitutive relations of domain media, such as absolute and relative permeability, porosity, capillary pressure, thermophysical properties of fluids and rock, and initial and boundary conditions of the system. Other processing requirements include specification of space-discretized geometric information (grid) and various program options (such as computational and time-stepping parameters). In general, a large-scale, three-dimensional simulation requires at least several gigabytes of memory, and the memory requirements should be distributed to all processors at input.

To make efficient use of the memory of each processor (considering that each processor has limited memory available), the input data files for the TOUGH2 simulation are organized in sequential format. There are two large groups of data blocks within a TOUGH2 mesh file, one with dimensions equal to the number of gridblocks and the other with dimensions equal to the number of connections (interfaces). Large data blocks are read one by one through a temporary full-sized array and then distributed to PEs (processing elements or processors). This method avoids storing all input data in a single PE (which may be too small). Other small-volume data, such as simulation control parameters, are duplicated on all PEs.

Initialization of the parallel code requires full-connection information, for such tasks as domain partitioning and local-connection index searching. These parts can exhaust the memory requirement when solving a large problem. Since the full-connection information is used only once at the beginning of a simulation, it may be better to handle these tasks in a preprocessing procedure.

Parallel Assembly of Jacobian Matrix

In the TOUGH2 formulation, the discretization of mass and energy conservation equations in space and time using the IFD leads to a set of strongly coupled nonlinear algebraic equations, which are solved by the Newton method. The resulting system of linear equations is then solved using an iterative linear solver with different preconditioning procedures.

The Jacobian matrix needs to be recalculated at each Newton iteration, and thus computational efforts may be extensive for a large simulation. In the parallel code, the assembly of the linear equation system is shared by all the processors, and each processor is responsible for computing the rows of the Jacobian matrix that correspond specifically to blocks in the processor's own *update* set. Computation of the elements in the Jacobian matrix is performed in two parts. The first part consists of computations relating to individual blocks (source/sink terms). Such calculations are carried out using the information stored on the current processor; no communications with other processors are needed. The second part includes all computations relating to the connections or flow terms. Calculation of flow terms at elements in the *border* set requires information from the *external* set, which in turn requires communication with neighboring processors. Before performing these computations, an exchange of relevant primary and secondary variables is necessary. For elements corresponding to *border* set blocks, one processor sends these elements to other related processors, which receive these elements as *external* blocks.

The Jacobian matrix for local gridblocks in each processor is stored in the distributed variable block row (DVBR) format, a generalization of the VBR format (Carney et al., 1993). All matrix blocks are stored row-wise, with the diagonal blocks stored first in each block row. Scalar elements of gridblocks are stored in column major order. The data structure consists of a real vector and five integer vectors, forming the Jacobian matrix. Detailed explanation of the DVBR data format can be found in Tuminaro et al. (1999).

Parallel Solving of Linear Equations

The linearized equation system arising at each Newton step is solved using an iterative linear solver from the Aztec package (Tuminaro et al., 1999). We can select different solvers and preconditioners from the package. The available solvers include conjugate gradient, restarted generalized minimal residual, conjugate gradient squared, transposed-free quasi-minimal residual, and bi-conjugate gradient with stabilization methods. The work of solving the global linearized equation is shared by all processors, with

each processor responsible for solving its own portion of the partitioned domain equations.

During a simulation, time steps are automatically adjusted (increased or reduced) depending on the convergence rate of iterations. In the parallel version code, the time-step size is calculated at the first processor (master processor, named PE0) after collecting necessary data from all processors. The convergence rates may be different in different processors. Only when all processors reach stopping criteria will the time march to the next time step. At the end of a time step or a simulation, the solutions obtained from all processors are then transferred to the master processor for output.

Communication Among Processors

Communication of data between processors working on neighboring/connected gridblocks, partitioned into different domains, is an essential component of the parallel algorithm. Moreover, global communication is also required to compute norms of vectors, contributed by all processors, for checking the convergence. In addition to the communication taking place inside the Aztec routine to solve the linear equation system, communication between neighboring processors is necessary to update primary and secondary variables (for example, a new Jacobian matrix is calculated for each Newton iteration). A subroutine is used to manage data exchange between processors. When the subroutine is called by a processor, an exchange of vector elements corresponding to the *external* set of the gridblocks is performed. During time stepping or Newton iteration, exchange of external variables is also required for the vectors containing the primary and secondary variables. More discussion on the scheme used for data exchange is given in Elmroth et al. (2001).

All data input and output are carried out through the master processor, while the most time-consuming efforts (assembling the Jacobian matrix, updating thermophysical parameters, and solving the linear equation systems) are distributed to all processors. In addition, the memory requirements are also distributed to all processors. Distributing both computing and memory requirements is essential for solving large-scale field problems.

APPLICATION EXAMPLES

Five examples are presented in the following sections to demonstrate applications of the TOUGH2_MP and the computational performance of the code.

Unsaturated Flow Simulation

The first problem is based on the site-scale model developed for investigations of the unsaturated zone at Yucca Mountain, Nevada (Wu et al., 1999). The problem involves using a much-refined grid to evaluate the numerical performance of the TOUGH_MP. The problem concerns steady-state unsaturated flow through fractured rock under ambient conditions, using a 3-D, unstructured grid. A dual-permeability approach is applied to handle fracture-matrix interactions. Model domain and the numerical grid encompass approximately 40 km^2 of the Yucca Mountain area. There are approximately 9,900 blocks per grid layer and about 60 computational grid layers vertically from land surface to water table. This results in a total of 1,100,000 gridblocks for fractures and matrix, and 4,050,000 connections. A distributed-memory Cray T3E-900 computer was used for this simulation example. This simulation was run as a single-phase flow (water with Richards' equation using the EOS9 module of TOUGH2).

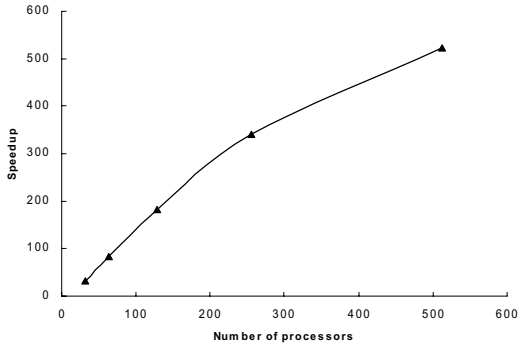


Figure 2. Speedup for the 3-D unsaturated flow simulation

In performance evaluation, the unsaturated flow problem was solved for 200 time steps using 32, 64, 128, 256, and 512 processors, respectively. Because of the automatic time-step adjustment, based on the convergence rate of the iteration process, the cumulative length of simulation times over 200 time-steps with different numbers of processors may be slightly different. However, the computational targets and efforts are similar, and comparing the performance of different numbers of processors with the same number of time steps is reasonable for evaluating parallel-computing efficiency.

Figure 2 illustrates the speedup of the code versus processor numbers used for a 200-time-step simulation. The speedup is defined as relative to the performance of 32 processors as $32T_{32}/T_p$, where T_p denotes the total execution time using p processors. Speedup factors using 32 to 64, 128, 256, and 512 processors are 2.63, 2.16, 1.87, and 1.54, respectively. Super-linear speedup appears when the processor number doubles from 32 to 64, and to 128 with a speedup of 2.63 and 2.16. The overall speedup

for 512 processors is 523. This behavior was extensively analyzed, including performance results for the different parts of the execution, for smaller problems in Elmroth et al. (2001). The computational performance for this example indicates that the parallel scheme implemented in TOUGH2_MP code is very efficient.

Two-Phase Fluid and Heat Flow

The second example is used to evaluate the numerical performance of the TOUGH2_MP for highly nonlinear, two-phase fluid and heat flow in fractured rock. The model domain and numerical grid for this problem are the same as the previous unsaturated flow case. The difference is that the current problem models flows of water, gas, and heat in a two-active-phase system using the multicomponent EOS3 module. The previous example handles only one active phase, using Richards' equation. Therefore, there are three equations per gridblock, and a total of $3 \times 1{,}075{,}522$ equations need to be solved per Newton iteration for the simulation. A distributed-memory IBM RS/600 SP was used for simulation.

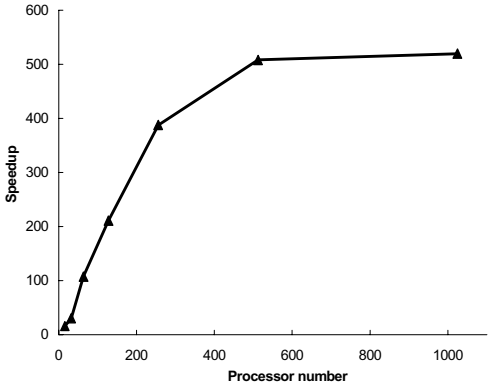


Figure 3. Speedup for the 3-D site-scale model of nonisothermal two-phase fluid and heat flow as a function of number of processors used

Figure 3 presents the speedup of the parallel-code simulation versus processor numbers used in this problem for a 1,000-year simulation. Similarly, the speedup is defined to be relative to the performance with 16 processors (i.e., equal to 16 for using 16 processors). Figure 3 also shows a better than linear speedup until the number of processors reaches 512. However, the speedup curve (as shown in Figure 3) for more than 512 processors becomes flat, which is very different from Figure 2. This deterioration in speedup performance using a larger number of processors results primarily from the increasing communication in the linear solver. Since this performance test is performed for a relatively short simulation time, the one-time expenses for the *input, distribution, and initialization* phases, also contribute to this effect, because this part is relatively large for this problem. For example, the one-time expense takes 35% of the total execution time in the case of using 1,024 processors, compared to 8.5% for 16 processors. Nevertheless, the effect of this one-time expense in the *input, distribution, and initialization* phase will diminish as total simulation times or time steps increase.

Liquid Flow Through Unsaturated Fractures

The third application is for analysis of flow focusing and discrete flow paths within the unsaturated zone of the Yucca Mountain site. In particular, the model is used to assess the frequency and flux distributions of major water-bearing flow paths from the bottom of the Paintbrush nonwelded (PTn) unit, a unit immediately above the Topopah Spring welded (TSw) unit. Flow focusing along the preferential paths or well-connected fracture networks may play an important role in controlling patterns of percolation through highly fractured tuffs. To quantify flow-focusing behavior, we developed a stochastic fracture-continuum model to incorporate fracture data measured from the welded tuffs and to study flow allocation mechanisms and patterns. Fracture permeability of the simulation domain is prescribed stochastically for its spatial distribution, conditioned using measured air-permeability data.

The model domain of the 3-D flow problem is a 50 m $\times$ 50 m $\times$ 150 m parallelepiped, with the upper boundary at the bottom of the PTn and the lower boundary at the repository horizon. The horizontal dimension was considered sufficient because the correlation length for variability in fracture permeability and spacing is approximately 1 m. The 150 m vertical extent of the model corresponds to an average distance from the interface between the PTn and TSw units to the repository horizon. The model covers five hydrogeological subunits from TSw31 to TSw35. The bottom of the PTn was chosen as the upper boundary because this unit behaves as a porous medium, leading to more uniform percolation flux to the units below.

A refined grid is generated with each gridblock size of 0.5 m $\times$ 0.5 m $\times$ 0.75 m, in the same order in dimension as observed for fracture spacing. Such a refined grid captures flow behavior through individual discrete fractures. This leads to 2×10^6 gridblock elements and 6×10^6 connections with the 3-D grid.

Table 2 presents a summary of execution times for two steady-state simulations of different boundary conditions (infiltration rates on the top boundary are 1 mm/year and 25 mm/year, respectively). The

simulations were run on a Cray T3E-900 machine. Table 2 shows that it took only about two hours to complete a two-million-gridblock simulation, demonstrating the efficiency of the parallel implementation. In addition, Table 2 shows that CPU times used in calculations of secondary parameters and assembly of the Jacobian are longer than those used in solving linear equations for this problem. This is very different from what we observed using a single-processor simulator with the Newton iteration scheme. This is because of the increase in overheads caused by communication between processors during setup of the linear equation system. Note that the longer execution times for the case of 1 mm/yr infiltration (compared with 25 mm/yr in Table 2) result from lower infiltration, leading to drier, unsaturated fractures or a more nonlinear problem. Therefore, a smaller infiltration case takes in general a longer simulation time for the example.

Table 2. Summary of execution times (seconds) used for two simulations of the 3-D flow-focusing problem, using 128 processors.

Simulation scheme	1 mm/yr Infiltration	25 mm/yr infiltration
Input, distribution, and initialization	218	218
Update parameters and setup Jacobian matrix	4865	4655
Solve linear equations	2109	1194
Total execution time	8111	6960

Transport Simulation

This example demonstrates the application and efficiency of the parallel code for modeling 3-D tracer/radionuclide transport within the unsaturated zone at Yucca Mountain, using the steady-state, 3-D flow field of the first example. In this case, the calculation was run to 2,000,000 years using 64 processors on the distributed-memory IBM RS/6000 SP computer.

The tracer is treated as a conservative component. Mechanical dispersion effects through the fracture-matrix system are ignored, and a constant molecular diffusion coefficient of 3.2×10^{-11} (m^2/s) is used for matrix diffusion. The transport simulation is run to 2,000,000 years, with a constant initial concentration source released at the repository from fracture or matrix blocks.

Table 3 presents parallel-code performance in simulating radionuclide transport of the initial radionuclide release from fractures. A comparison of the execution times, spent in different portions during the two-million year simulation (shown in Table 3), indicates that solving the linear equations takes a much smaller percentage (17%) of the total simulation time than that for the flow simulation. This is because the transport problem becomes simply linear with the computational options selected.

Table 3. Summary of execution times and iterations for the transport simulation with fracture release using 64 processors.

Input, distribution, and initialization (s)	139.7
Update thermophysical parameters, Jacobian matrix (s)	333.7
Solve linear equations (s)	124.5
Total execution time (s)	727.7
Total Newton iterations	118
Total time steps	118
Total Aztec iterations/PE of solving linear equations	1,096

Unsaturated Discrete Fracture Flow Simulation

This example pertains to investigating flow-focusing phenomena in a large-scale discrete-fracture network, constructed using field data collected from the unsaturated zone of Yucca Mountain, Nevada. We constructed the two-dimensional, vertical-cross-section fracture network using fracture mapping data, including field-measured fracture density, trace lengths, and orientations. For simplicity, each fracture in the network is randomly distributed. However, generation of the fracture network is governed by statistical information from field measurement data. The statistical properties of the generated fracture network should reflect the corresponding properties of fracture distribution in the study domain. This generated two-dimensional fracture network for an area of 100m×50m contains more than 20,000 fractures.

The fracture network is discretized into 126,432 linear elements that have a maximum length of 1 m. The elements have a total connection of more than 300,000. The intersection point of any two fractures is treated as an element. The volume of these intersection elements is possibly extremely small. In fact, the volume of the fracture elements may vary over several orders of magnitude.

Flow simulations are carried out by introducing water infiltration at the top of the fracture network. Because of the strong heterogeneities and complex fracture connections present in fractures along the top boundary, it is impossible to simply distribute desired infiltration water uniformly over all fractures crossing the top boundary. In response to this limitation, we accomplish infiltration by attaching an

additional gridblock to the top of the fracture network. The entire top boundary of the network is connected to this single block, and a prescribed infiltration rate is applied to it. As water is injected into this block, water eventually flows into the fractures beneath the block. This outflow will in general partition non-uniformly among the fractures. After a rapid initial transient, this infiltration block will reach a steady flow condition. In this way, a constant infiltration rate is applied to the top boundary of the fracture network. Side boundaries are considered impermeable, and a free-drainage condition is imposed at the bottom boundary.

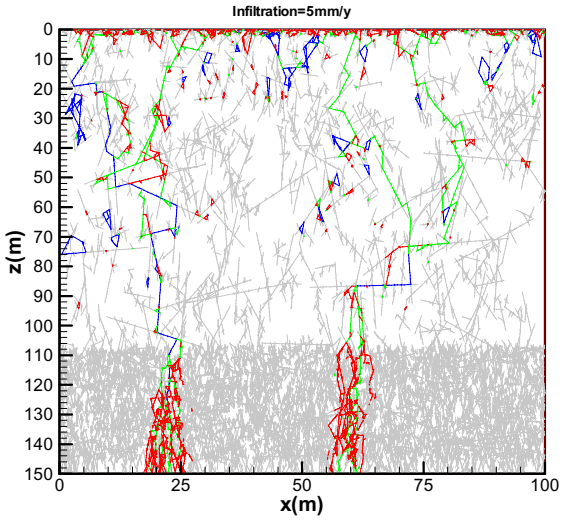


Figure 4. Steady-state flux distribution in the fractures. Each color represents a one-order-of-magnitude difference in flux (from high to low: blue, green, red, and grey).

Steady-state unsaturated water flow through the fracture network is simulated using the TOUGH2_MP. Because randomly distributed fractures can lead to an extremely complex element connection system, and elements have large differences in volume, numerical difficulties can arise, requiring intensive computational work. All computations are conducted using 64 processors on an IBM SP RS/6000 supercomputer. This simulation is very time consuming. Without using the parallel code, we would have difficulty obtaining a steady-state solution for this simulation. Figure 4 shows the simulation result of steady-state flux distribution in the fracture network.

SUMMARY AND CONCLUSIONS

This paper reviews the development of TOUGH2_MP code and presents application examples for the code to large-scale simulations. The code was written in Fortran 77 and 90, using MPI for interprocessor communication. It uses the METIS software package for partitioning the unstructured computational domain and the AZTEC software package for solving linear equations. In TOUGH2_MP, both computing time and memory requirements are distributed among and shared by all processors of a multi-CPU computer or cluster. The best numerical performance has been achieved by integrating and optimizing the following procedures: (1) efficient domain partitioning; (2) parallel assembly of the Jacobian matrix; (3) parallel preconditioned iterative linear solving; (4) fast communication and data exchange between processors; and (5) efficient utilization of the processors' aggregate memory.

Five application examples of the TOUGH2_MP code have been presented. These examples are run on a Cray T3E or IBM RS/600 SP system. Code performances are evaluated through modeling flow/transport in the unsaturated zone at Yucca Mountain, using different numbers of processors. Test results indicate that the overall performance of the parallel code shows significant improvement in both efficiency and ability in solving large-scale reservoir simulation problems. The major benefits of the code are that it (1) allows for accurate representation of reservoirs with sufficient resolution in space, using refined grids, (2) allows for adequate description of reservoir heterogeneities, and (3) overcomes the limits on problem size and computational requirements experienced in large-scale modeling studies that use conventional single-CPU simulators.

ACKNOWLEDGMENT

The authors would like to Guoxian Zhang and Dan Hawkes for their review of this paper. This work was supported by the Laboratory Directed Research and Development (LDRD) program of the Ernest Orlando Lawrence Berkeley National Laboratory. The support is provided to Berkeley Lab through the U. S. Department of Energy Contract No. DE-AC03-76SF00098.

REFERENCES

Carney S., M. Heroux, and G. Li, *A Proposal for a Sparse BLAS Toolkit*, Technical Report, Cray Research Inc., Eagen, MN, 1993.

Elmroth E., On Grid Partitioning for a High Performance Groundwater Simulation Software, Engquist et al. (eds), *Simulation and Visualization on the Grid*, Springer-Verlag, Berlin, Lecture Notes in Computational Science and Engineering, No. 13, pp. 221-233, 2000.

Elmroth, E., C. Ding, and Y. S. Wu, High Performance computations for Large-Scale Simulations of Subsurface Multiphase Fluid and Heat Flow, *The Journal of Supercomputing*, 18(3), 233-256, 2001.

Karypsis, G. and V. Kumar, *METIS. A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices*, V4.0. Technical Report, Department of Computer Science, University of Minnesota, 1998.

Message Passing Interface Forum. MPI: A Message-Passing Interface Standard, *International Journal of Supercomputing Applications and High performance Computing***,** 8(3-4), 1994.

Narasimhan, T. N. and P. A. Witherspoon, An Integrated Finite Difference Method for Analyzing Fluid Flow in Porous Media, *Water Resources Research*, 12(1), pp. 57-64, 1976.

Pruess K., *TOUGH2 – A general-purpose numerical simulator for multiphase fluid and heart flow*, Lawrence Berkeley Laboratory Report LBNL-29400, Berkeley, CA, 1991.

Tuminaro R.S., M. Heroux, S.A. Hutchinson, and J.N. Shadid, *Official Aztec user's guide*, Ver 2.1, Massively Parallel Computing Research Laboratory, Sandia National Laboratories, Albuquerque, NM, 1999.

Wu Y. S., C. Haukwa, and G.S. Bodvarsson, A Site-Scale Model for Fluid and Heat Flow in the Unsaturated Zone of Yucca Mountain, Nevada, *Journal of Contaminant Hydrology*, 38(1-3), 185-217, 1999.

Wu Y. S., C. Haukwa, and S. Mukhopadhay, TOUGH2 V1.4 and T2R3D V1.4 Verification and Validation Report, Rev. 00, *Earth Sciences Division, Lawrence Berkeley National laboratory*, 1999.

Wu Y. S., K. Zhang, C. Ding, K. Pruess, E. Elmroth, and G. S. Bodvarsson, An efficient parallel-computing method for modeling nonisothermal multiphase flow and multicomponent transport in porous and fractured media, *Advances In Water Resources*, 25,243-261, 2002

Zhang K., Y. S. Wu, C. Ding, K. Pruess, and E. Elmroth, Parallel Computing Techniques for Large-Scale Reservoir Simulation of Multi-Component and Multiphase Fluid Flow, Paper SPE 66343, *Proceedings of the 2001 SPE Reservoir Simulation Symposium*, Houston, Texas, 2001.