PROCEEDINGS, TOUGH Symposium 2006
Lawrence Berkeley National Laboratory, Berkeley, California, May 15–17, 2006

ENHANCING SCALABILITY AND EFFICIENCY OF THE TOUGH2_MP FOR LINUX CLUSTERS

Keni Zhang and Yu-Shu Wu

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

ABSTRACT

TOUGH2_MP, the parallel version TOUGH2 code, has been enhanced by implementing more efficient communication schemes. This enhancement is achieved through reducing the amount of small-size messages and the volume of large messages. The message exchange speed is further improved by using non-blocking communications for both linear and nonlinear iterations. In addition, we have modified the AZTEC parallel linear-equation solver to non-blocking communication. Through the improvement of code structuring and bug fixing, the new version code is now more stable, while demonstrating similar or even better nonlinear iteration converging speed than the original TOUGH2 code. As a result, the new version of TOUGH2_MP is improved significantly in its efficiency. In this paper, the scalability and efficiency of the parallel code are demonstrated by solving two large-scale problems. The testing results indicate that speedup of the code may depend on both problem size and complexity. In general, the code has excellent scalability in memory requirement as well as computing time.

INTRODUCTION

The TOUGH2_MP code (Zhang et al. 2001, Wu et al. 2002) was originally developed on the CRAY T3E and IBM SP supercomputers in 2000. The code has been applied successfully to solve multi-million gridblock problems on super-computers (Zhang et al. 2003). However, because of the requirements for extensive communications during Newton-Raphson iterations and linear equation solutions, there are still problems with the parallel-code scalability and efficiency when run on Linux/PC clusters, which in general have relatively lower bandwidth or higher latency.

The TOUGH2_MP code uses the MPI (Message Passing Forum, 1994) for parallel implementation. The code partitions a simulation domain, defined by an unstructured grid, using partitioning algorithm from the METIS software package (Karypsis and Kumar, 1998). In parallel simulation, each processor is in charge of one part of the simulation domain for updating thermophysical properties, assembling mass and energy balance equations, solving liner equation systems, and performing other local computations. The local linear-equation systems are solved in parallel by multiple processors with the Aztec linear solver package (Tuminaro et al., 1999). Data communication between processors is an essential component of the parallel code. Although each processor solves the linearized equations of local- or subdomains independently, communication between neighboring processors is necessary to update and solve the entire equation system during solving the linear equation systems. Detailed discussion of the prototype of data-exchange scheme can be found in Elmroth et al. (2001).

The new version TOUGH2_MP code has been implemented with more efficient communication schemes. The enhancements are achieved through reducing the numbers of small-size messages and volume of large messages. At each Newton iteration, the subdomain boundary information exchanges are limited to primary variables only. All other or secondary variables are updated using primary variables locally. The message exchange speed is further improved by using non-blocking communications for both linear and nonlinear iterations. We have also modified the AZTEC parallel liner-equation solver to non-blocking communication. As a result, the new version of TOUGH2_MP is improved significantly in its computational efficiency. With the improvement in code structuring and bug fixing, the new version code is more stable, faster nonlinear iteration converging speed, and better scalability, when compared to the previous version. Testing examples show that linear or super linear speedup can be obtained on both clusters and supercomputers for most problems.

MATHEMATICAL MODEL AND PROBLEMS IN PARALLELIZATION

The TOUGH2_MP solves the same equation systems as those solved by the original TOUGH2 code. The basic mass- and energy-balance equations solved by TOUGH2 can be written in the general integrated form (Pruess et al. 1999):

$$\frac{d}{dt}\int_{V_n} \boldsymbol{M}^{\kappa} dV_n = \int_{\Gamma_n} \boldsymbol{F}^{\kappa} \bullet \boldsymbol{n} d\Gamma_n + \int_{V_n} q^{\kappa} dV_n \qquad (1)$$

The integration is over an arbitrary subdomain V_n of the flow system under study, which is bounded by the closed surface Γ_n. The quantity M appearing in the accumulation term (left hand side) represents mass or energy per volume, with $\kappa = 1, ..., NK$ labeling the mass components (water, air, CO_2, solutes, ...), and $\kappa = NK + 1$ the heat component. **F** denotes mass or heat flux (see below), and q denotes sinks and sources, and **n** is a normal vector on surface element $d\Gamma_n$, pointing inward into Vn.

Time and space discritization for Equation (1) results in a set of coupled nonlinear equations, which can be written in residual form (Pruess et al., 1999):

$$R_n^\kappa(x^{t+1}) = M_n^\kappa(x^{t+1}) - M_n^\kappa(x^t) - \frac{\Delta t}{V_n}\{\sum_m A_{nm}F_{nm}^\kappa(x^{t+1}) + V_n q_n^{\kappa,t+1}\} = 0 \quad (2)$$

where the vector x^t consists of primary variables at time t, R_n^κ is the residual of component κ for block n, M denotes mass or thermal energy per unit volume for a component, V_n is the volume of the block n, and q denotes sinks and sources of mass or energy, Δt denotes he current time-step size, t+1 denotes the current time, A_{nm} is the interface area between blocks n and m, and F_{nm} is the flow between them. Equation (2) is solved by Newton/Raphson iteration, leading to

$$-\sum_i \frac{\partial R_n^{\kappa,t+1}}{\partial x_i}\bigg|_p (x_{i,p+1} - x_{i,p}) = R_n^{\kappa,t+1}(x_{i,p}) \quad (3)$$

where $x_{i,p}$ represents the value of ith primary variable at the pth iteration step.

The parallel implementation for solving Equation (3) first partitions the simulation domain. After domain decomposition, computations (which include assembling Jacobian matrix, solving linear equations, and updating thermophysical properties) are done at the local subdomain by different processors. All processors involved in solving Equation (3) will solve similar equation systems for different subdomains. This property guarantees as large as possible reuse of sequential TOUGH2 code for computation parts. Each processor solves only part of the full modeling domain; and definitely speeds up the computation. However, the best convergence performance of Newton iteration can only be achieved when the local equation systems are solved simultaneously as partial of the whole system. Doing so therefore requires intensive and expensive communications between neighboring processors.

In addition, while the equation system is solved, communications between processors are needed for updating border thermophysical properties, collecting extreme values, conditioning over the whole simulation domain, and input/output. Equation (2) shows that only the flow term needs information from neighboring gridblocks for mass and energy conservation computation. The mass accumulation terms, and sink/source terms, are domain independent. Computations related to the two terms do not require communication between neighboring processors.

The efficiency of the TOUGH2_MP depends on code efficiency in both computation and communication. Because we use similar computation schemes in the parallel code, the efficiency of the code is mainly determined by communication efficiency. In addition, domain partitioning schemes used in parallel computing may introduce additional non-linearity, as shown in the example problems below, to the global discretized equation for describing a fully coupled physical system, while handling extra cross-bound flux terms between partitioned grid domains. This additional perturbation to the equation system from parallelization may partially override the numerical performance benefit from parallel computing itself, when solving highly nonlinear problems using non-Newtonian iteration or a less than fully implicit scheme.

CODE IMPROVEMENT

The parallel code performance can be improved by improvement in the partitioning algorithm, parallel computation schemes, and communication efficiency. A good partitioning algorithm should be able to balance computation load and communication load, and minimize communication volume and number of messages. We use the METIS (Karypsis and Kumar, 1998) software package for domain partitioning. The package provides three partitioning algorithms, the *K-way*, the *VK-way*, and the *Recursive*, for unstructured grid partition. Elmroth (2000) has provided an in-depth discussion on the performance of the three algorithms through the prototype of this code. He concluded that the *Recursive* outperforms the other two algorithms in terms of computation load balance, and *VK-way* outperforms the other two algorithms in minimizing the maximum and average communication volume in terms of number of external gridblocks required per processor. In general, the three algorithms produce partitions that are beneficial from different aspects. *Recursive* performs the best work in load distribution up to a large number of processors, *VK-way* is best in minimizing the communication volume, and *K-way* is slightly better than the other two in minimizing the number of messages to be sent during computation. To achieve better code performance, different algorithms may be selected, based on the system communication bandwidth, latency, and CPU speed that the code is run at.

Efficiency of the parallel code can be improved through enhancement of the code communication performance. Communication volume and message number are the most important factors that influence total communication time. In TOUGH2_MP program, there are three most important type communications: boundary message exchange at the end of each Newton iteration, message exchange for parallel solving of linear equations, conditioning, gathering, and searching for extreme values.

```
At the beginning of each Newton iteration
    1.  Start communication for primary variables
        (in background)
    2.  Do 1 to LNEL (LNEL: total gridblocks for
        current processor)
    3.  updating secondary variables(EOS)
    4.  End Do
    5.  Make sure communication finished
    6.  Do LNEL+1 to LNEL+extarnal_blocks
    7.  updating secondary variables(EOS)
    8.  End Do
    9.  assembling Jacobian matrix, no
        communication needed(MULTI)
```

Figure 1. Outline of the communication procedure for boundary message exchange at the end of each Newton iteration

It is possible to minimize the communication volume for boundary message exchange at the end of each Newtown iteration, by introducing some computation overlapping. Note that TOUGH2 uses two type variables: the primary thermodynamic variables for all gridblocks, and all other (secondary) thermophysical parameters needed to assemble the governing flow and transport equations. At each iteration step, primary variables are solved directly from the equation system, and secondary variables are then updated based on the new primary variables. To reduce communication volume, boundary message exchange is limited to the primary variables only. Entries of the Jacobian matrix related to the gridblocks of the *internal* set are updated using only the information on the current processor (discussion of the different gridblock sets can be found at Wu et al [2002]). The *border* set, consisting of blocks with at least one edge to a block assigned to another processor, requires 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 of current processors, is referred to as an *external* set. If only the primary variables are communicated, secondary variables for the *external* set must be updated at the current processor, using the most updated primary variables of the *external* set. This approach may introduce an additional computation burden with respect to updating secondary variables for the *external* set. However, for most Linux clusters, trading off between computation and communication may gain obvious efficiency for the code. A scheme with non-blocking communication is implemented in the enhanced version of the code. Figure 1 outlines of the communication procedure.

Non-blocking communication can improve code performance on many systems by overlapping communication and computation. The AZTEC parallel linear solver has been modified to non-blocking communication for message exchanges during solving equations. The most time-consuming computation for solving linear equations is matrix-vector multiplication. The multiplication at the current processor needs the most updated information for its external blocks from neighboring processors. The non-blocking communication was implemented in the code as shown on Figure 2.

```
At the beginning of matrix-vector multiplication
  1.  Start communication (in background)
  2.  Do 1 to  Internal_Blocks
  3.  M.V
  4.  End Do
  5.  Make sure communication finished
  6.  Do Internal_Blocks+1   to   LNEL
  7.  M.V
  8.  End Do
```

Figure 2. Non-blocking communication for the parallel linear equation solver

Small-size messages are needed to coordinate the processors involved in parallel solution of a problem. Reducing the total small-message numbers is an important factor in improving code performance, especially on high-latency computer systems. The most common small messages may include messages for finding and collecting information. The code finds global extreme values, performs global conditioning, and calculates global sums through reducing and gathering operations. The number of small messages may be reduced through combination of several messages into one message. For example, at the end of each Newton iteration step, the code needs to check convergence of the iteration. This requires collecting the maximum residual from all processors. If the maximum residual is less than the convergence criterion, the program will proceed to the next time step. The code also needs to gather information about the origin of maximum residual (i.e. from which gridblock and which component). These different types of messages (real, characters, integer) may be combined to one small array for communication. Another common message is for collecting values of a variable from all processors. Then, based on the variable values, the code will perform different actions. If the action is to exit program execution, communication may not be necessary; the program can directly examine conditions for the variable values at local processors. If the condition is valid, the processor will inform all other processors to stop execution.

EXAMPLES FOR SCALABILITY AND EFFICIENCY INVESTIGATION

The original version of the TOUGH2_MP code had shown the ability to efficiently use a large number of processors (as many as 1024 processors) on supercomputers (Zhang et al. 2003, Wu et al., 2002). The new version, improved in communication scheme, is expected to be more efficient on relatively low bandwidth and high latency systems. Here, we investigate the scalability and efficiency of the new-version code through two examples, running on a 32-processor cluster and a super computer. The two examples solve unsaturated flow and two phase flow problems, respectively.

The first example demonstrates simulation of mountain-scale unsaturated flow for the Yucca Mountain site. The 3-D model domain was discretized into 60 computational grid layers and 9,900 blocks per layer. A dual porosity approach was used to represent the fractured media. The model consists of 1,075,522, gridblocks and 4,047,209 connections. Detailed discussion of the problem, including geological layers, model grid, and boundary conditions, is presented in Zhang et al. (2003).

Table 1 shows the reduction in execution time as the number of processors increase. Simulation was run for 100 time steps, representing a similar computation load. Because of the automatic time-step adjustment, based on the convergence rate of the iteration process, the cumulative length of simulation time over 100 time steps with a different number of processors may be slightly different. These simulations were run under the same conditions, except for using a different number of processors.

The time reduction is significant with the increase in numbers of processors. In general, performance of the code for the two parts, (1) updating thermophysical parameters and assembling Jacobian matrix, and (2) solving linear equations; and total execution shows a linear and super linear speedup. Note that except for the time for the two-part computations, total execution time also includes time for input/output, initialization and others. The table shows that the parallel performance in solving linear-equation systems is much better than linear speedup. Elmroth et al. (2000) provided a theoretical analysis for this phenomenon and indicated that the super linear behavior is mainly caused by the performance of the preconditioner. The time of 64-processor simulation is that running on a IBM SP super-computer; it may not be comparable with other simulation runs because of the platform difference. However, by comparing to the 32-processor simulation run, we find that the time reduction for solving linear equations is more significant than for updating thermophysical parameters and assembling Jacobian matrix. This is because updating thermophysical parameters and assembling Jacobian matrix is computation dominant; and solving linear equations require intensive communication. The cluster has better computation performance and lower communication speed compared to the super computer used for the simulation.

Table 1 also shows reduction of the memory requirement with the increase in processor number. Doubling the processor number reduces the memory requirement for each processor by slightly less than a half. From the point of view of memory requirement, the 32-processor cluster can perform simulations for models with several-tens of million gridblocks.

A similar comparison, running the same problem on

Table 1. Comparison of execution time and memory use for 100 time steps using different numbers of processors on Linux clusters and a supercomputer--cluster cpu: Athlon, 1.7GHz; super-computer cpu: power 3, 375MHz

Number of processors	2	4	8	16	32	64*
Time for updating thermophysical parameters and assembling Jacobian matrix (s)	1458	776	359	205	101	135
Time for solving linear equations (s)	20100	7588	1921	504	201	87
Total execution time (s)	21804	8583	2486	919	550	554
Memory used by each processor	27.5%	15%	7.8%	4.2%	2.2%	

** running on IBM RS/6000 super-computer*

IBM RS/6000 super-computer using the previous version code with many more processors, was reported by Wu et al. (2002). By comparing current results with the previous data, we can see that the efficiency with the new version has been improved significantly.

The second example demonstrates a three-multidimensional, mountain-scale, thermal-hydrologic (TH) numerical model for investigating unsaturated flow behavior in response to decay heat from the radioactive waste repository in the Yucca Mountain unsaturated zone (UZ). The model evaluates the coupled TH processes related to mountain-scale UZ flow. It also simulates the impact of radioactive waste heat release on the natural hydrogeological system, including heat-driven processes occurring near and far away from the emplacement tunnels or drifts. Model simulations predict thermally perturbed liquid saturation, gas- and liquid-phase fluxes, and water and rock temperature elevations, as well as the changes in water flux driven by evaporation/condensation processes and drainage between drifts.

The model covers approximately 20 km2 of the area and uses a refined mesh in the vicinity of the emplacement drifts. The model grid explicitly incorporates every repository drift by taking into account their orientations, lengths, elevations, and spacing. In the model grid, faults are explicitly represented by vertical or inclined zones of finite width, and properties for gridblocks within the fault zones are adjusted to represent specific fault configurations. The TH model mesh consists of 86,440 gridblocks, and 343,520 connections between the gridblocks. Detailed discussion of the model development can be found at Wu et al. (2006). Since the model involves very complex thermal hydrodynamics processes, the simulation requires intensive computation effort. Previous simulation (Wu et al., 2006) was done using the single CPU version TOUGH2 for 5000 years, which took several months to finish. The model was run again with the same input by TOUGH2 V1.4 and TOUGH2_MP for a 100-year simulation. Table 2 shows the performance of the parallel code by comparing to performance of original TOUGH2 code.

Table 2 shows that the total execution time for the 100-year thermal loading simulation decreases from 188.0 hours with the single CPU version TOUGH2 to 5.77 hours with TOUGH2_MP using 32 processors, and corresponding memory requirement decreases from 27% to 0.9%. The computing time reduction is very significant with the increase in numbers of processors. Note that the total number of time steps needed for the simulation is quite different with different numbers of processors. Moreover, with the increase in numbers of processors, the average iterations for solving the linear equation system increase. This is because the domain decomposition based process is expected to become less efficient as the number of processors increases. Further discussion of this behavior is provided by Elmroth et al.(2000). Despite the overall increasing number of iterations in both Newton iteration and linear solver for increasing numbers of processors, performance of the parallel code is still very impressive (near linear speedup).

Table 2. Comparison of performance for 100-year thermal loading simulation using different numbers of processors

Number of processors	1*	4	8	16	32
Time for updating thermophysical parameters and assembling Jacobian matrix (hr)		6.39	3.20	1.78	1.03
Time for solving linear equations(hr)		38.89	16.51	8.73	4.74
Total execution time (hr)	188.0	45.28	19.71	10.51	5.77
Memory used by each processor	27.5%	4.2%	2.2%	1.3%	0.9%
Total Newton iterations		9544	9033	9352	9886
Total time steps	5417	3364	3131	3464	3901
Average iterations for solving linear equations		25.3	28.8	33.6	35.1

** run by TOUGH2 V1.4 with exactly the same input files and same computer*

CONCLUSIONS

A new version of TOUGH2_MP has been developed, adding enhancements in communication through reducing the number of small-size messages and volume of large messages. The message-exchange speed is further improved by using non-blocking communications for both linear and nonlinear iterations. With the improvement in code structuring and bug fixing, the new version code is also more stable. The code demonstrates similar or even better nonlinear iteration converging speed than the original TOUGH2 code. As a result, the new version of TOUGH2_MP has significantly improved efficiency. The improvement in scalability and efficiency of the new-version parallel code is demonstrated through two examples, showing good efficiency and scalability in both memory requirement and computing time. It is expected that this code will find more applications in solving multiple million gridblock problems, on clusters or supercomputers with several tens to hundreds of processors.

ACKNOWLEDGEMENTS

The authors would like to thank Lehua Pan and Yinqgi Zhang for their review of this paper. Thanks are also due to Greg Kurtzer for the Linux cluster technical support. This work was supported by the U.S. Department of Energy. The support is provided to Berkeley Lab through the U.S. Department of Energy Contract No. DE-AC03-76SF00098.

REFERENCES

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), pp. 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.

Pruess, K, C. Oldenburg, and G. Moridis, *TOUGH2 User's Guide, V2.0.* Lawrence Berkeley National Laboratory Report LBNL-43134, Berkeley, CA, 1999.

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

Wu, Y.S., S. Mukhopadhyay, K. Zhang, and G. S. Bodvarsson, A mountain-scale thermal-hydrologic model for simulating fluid flow and heat transfer in unsaturated fractured rock, accepted by *Journal of Contaminant Hydrology*, 2006.

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.

Zhang, K., Y.S. Wu, and G.S. Bodvarsson, Parallel computing simulation of fluid flow in the unsaturated zone of Yucca Mountain, Nevada, *Journal of Contaminant Hydrology*, 62-63, 381-399, 2003.