RECENT EXPERIENCES WITH OVERCOMING TOUGH2 MEMORY AND SPEED LIMITS

Angus Yeh, Adrian Croucher and Michael J. O'Sullivan

Department of Engineering Science

University of Auckland, Private Bag 92019, Auckland 1142, New Zealand

cyeh015@aucklanduni.ac.nz

Keywords: *geothermal, simulation, modelling, memory limit, speed, performance, parallel computing, TOUGH2, AUTOUGH2, TOUGH2-MP, Linux, Windows, 32-bit, 64-bit.*

ABSTRACT

Modellers face two issues when they are increasing the complexity and size of models: memory limit and computational speed.

TOUGH2/AUTOUGH2 users on 32-bit computer systems are limited to 2 GB of memory, the amount required for a 3-D model with approximately 80,000 blocks. We discuss here how this limit can be avoided by using a 64-bit computer architecture. We also discuss modifications to the AUTOUGH2 code to make memory use more efficient by making the memory allocation dynamic.

The use of new compilers and new hardware helps to improve the speed of TOUGH2 simulations. One can achieve even more improvement by using the parallelised version, TOUGH2-MP. We have tested the speedup of TOUGH2-MP on a variety of machines, and results are good. With the advancement of multi-core personal computers, great speed-up of TOUGH2 simulations by using a parallel simulator is now very practical and affordable.

1. INTRODUCTION

Larger, more complex models are often desired by modellers to obtain more accurate long-term predictions, higher spatial resolution and detailed local effects. Often these improvements of the models require greater computational power, in terms of larger memory requirements and extended simulation time.

Advancement of computer technology can help to tackle these issues. However, it is not always easy to fully utilise new computer hardware and software systems. This paper describes our recent experiences in speeding up the performance of TOUGH2. In particular discussions are made on memory limits of 32-bit versus 64-bit versions of the simulator and computer operating systems and speeding up simulations with parallel code.

The simulators described here are TOUGH2 [11] and two additional derived versions of it: AUTOUGH2 [7] and TOUGH2-MP [14]. TOUGH2 is a simulator that is widely used in the geothermal industry [10]. It is an integral finite difference or finite-volume code designed for multi-phase multi-component fluid flows in porous media. AUTOUGH2 (the University of Auckland version of TOUGH2) is a locally modified version with some additional capabilities such as faster thermodynamic calculations, a combined-EOS single executable, and a number of additional generator types useful for more complex boundary conditions and future scenario modelling. TOUGH2-MP (Massively Parallel Version of TOUGH2) is a code that parallelised TOUGH2 to allow a single simulation to be run on multiple CPUs (central processing units) or multiple nodes in a cluster environment, hence shortening the simulation time by distributing the computational load.

2. MEMORY LIMIT

Even though the 64-bit computer architecture was developed a long time ago, the 32-bit architecture has still been dominant until recently. It is difficult to obtain reliable statistics, but there are still a large number of 32-bit operating systems and applications in use, even on 64-bit capable hardware.

One of the benefits of the newer 64-bit architecture over older 32-bit is the greatly increased memory limit. For many applications, the 32-bit limit is still beyond the normal requirements, but this is, however, no longer the case for geothermal modellers. The size limit of TOUGH2 models under the 32-bit platform will be discussed and it will be shown that the implied limit on model size is no longer beyond routine requirements. In the geothermal modelling group at the University of Auckland many current models of various geothermal fields are approaching that limit or exceed it.

Model	Year	Blocks	Connections
Wairakei-Tauhara	2010	9,011	26,179
Ohaaki	2011	22,817	66,999
Wayang Windu	2010	33,092	84,235
Taupo Reporoa Basin	2011	40,237	119,022
Wairakei-Tauhara	2011	41,461	117,142
Palinpinon	2011	64,754	192,209

2.1 Memory explained

Before looking at the limits of various hardware, OS (operating systems) and applications, it is important to understand terms used such as physical memory, total virtual memory and process address space. Different hardware and operating systems use different terminology, and how they work can be slightly different. What will be described here is based on mainstream PCs (personal computers), with Microsoft Windows or Linux OS.

Physical memory is the easiest to understand. It is simply the hardware RAM (random-access memory) that is installed on a given system. On modern computers there are different amounts of maximum accessible RAM that different OS (either 32- or 64-bit) can utilise. More RAM should result in overall better system performance. However in contrast to most people's instincts, this may be largely irrelevant to the size of TOUGH2 model that can be run. The key parameter is process address space.

Total virtual memory is the amount of memory that an OS can utilise, which is nearly always larger than the RAM installed. In addition to the RAM installed, part of the hard drive (swap file in Linux and page file in Windows) is added to the total virtual memory. Since computers typically now have relatively cheap and large hard drives, the size of total virtual memory is usually not a limiting factor. This total virtual memory is what is actually available to be shared by all the applications running on an OS.

Process address space is a portion of the total virtual memory that can be 'seen' by a single process/application. The OS dictates which parts of the total virtual memory a running application can access. The limit on process address space directly affects how much memory a TOUGH2 simulation can use, and hence limiting the model size. It is important to understand that even if an application uses more process address space memory than the actual RAM installed, it can still be run, as long as there is enough total virtual memory left. However, there is a penalty on performance when the address space for a process is significantly larger than RAM, because some of the memory allocated to the process has to physically sit in the hard drive, which is very slow compared to RAM.

2.2 Process Address Space Limits

In general, 64-bit hardware can support either a 32- or 64-bit OS. A 64-bit OS allows both 64- and 32-bit applications to run. But 32-bit hardware cannot support a 64-bit OS. Likewise, a 32-bit OS cannot run 64-bit applications. Thus it is wise to choose 64-bit hardware and a 64-bit OS for flexibility.

For running an application, the process address space is an array of memory that is accessible by memory pointers. A 32-bit application uses a 32-bit long pointer. This allows a maximum of 2^32 bits, or 4 GB (gigabytes), of memory to be accessed.

Theoretically 64-bit architecture allows a maximum of 2^64 bits of process address space. However, hardware available today currently limits it to 2^48 bits, or 256 TB (terabytes), by only using 48 bits memory pointers. Some current OS even restrict this number further in their implementation. Nevertheless, these numbers are still significantly larger than the 32-bit limit.

There is a further twist on these limits. On 32-bit Windows and Linux, the 4 GB addressable space is further separated into user mode and kernel mode. The kernel mode is directly controlled by the OS to provide system level services. User mode is the actual memory space that an application, such as TOUGH2, can use freely. The user mode limit is usually 2 GB. With some applications and OS tweaks, the limit can be lifted to approximately 3 GB. In the case when a 32-bit application is running on a 64-bit OS, the OS kernel will run in 64-bit mode, making the whole 32-bit process address space of 4GB available to the application.

	32-bit OS		64-bit OS	
	Windows	Linux	Windows	Linux
32-bit App	2 GB[1]	2 GB[1]	4 GB	4 GB
64-bit App	-	-	8 TB	128 TB[2]

[1] Expandable to 3 GB with special system configuration

[2] Varies with different Linux builds

The table above summarises the limits. Running 64-bit applications on a 64-bit OS is obviously the better choice. It is also valuable to recognise the benefit of running 32-bit applications on 64-bit OS, if one is unable to recompile simulation codes into 64-bit executables.

2.3 Static versus dynamic data

There is a further memory issue inherent with the original code for TOUGH2, or AUTOUGH2. All the variables used to store data such as blocks, connections, generators, and linear equation data are all declared with COMMON statements in the FORTRAN language. These are stored as static data in the process address space. Dynamic data has space allocated at run-time, but the static data of an application is fixed at compilation. Due to the way process address space is managed, the maximum allowable static data is less than that available to dynamic data.

On Linux, there are compiler options that can change the way the system arranges the process memory and therefore removes the 2 GB limit. This is arguably less efficient but we have found no real performance impact. On Windows, there is a fixed maximum of 2 GB of process memory, even if the code is compiled as a 64-bit executable. In contrast, there is no such limit on dynamic data, which is simply bounded by the available memory in process address space at the time of allocation.

In addition to the limits imposed on the static data memory, using static data has several other drawbacks. The TOUGH2/AUTOUGH2 executable needs to be re-compiled each time a bigger model is required. Static data is less efficient than dynamic data, which can allocate exactly the required amount at run-time.

We have updated the AUTOUGH2 code to use dynamic memory allocation. The exact memory size needed is only determined after processing the input file. We had some concerns about the loss in performance that might be incurred by changing from static to dynamic memory allocation, but the effect is found to be insignificant. A few seconds of additional time are required to determine the model size and allocate memory but there is no effect on the actual computation time at all.

Lawrence Berkeley National Laboratory also used dynamic memory allocation when they parallelised the original TOUGH2 code into TOUGH2-MP. The users of the original TOUGH2 are less fortunate, as the problem of the 2 GB static data limit cannot be solved with a method that works across different platforms and compilers.

2.4 Overcoming memory limits with parallelisation

It is important to emphasise again that the 2 GB limit of 32-bit systems only applies to a single application process. With parallelisation, it is possible to gain access to more memory by using multiple processes, each having its 2 GB limit.

The parallel simulation code TOUGH2-MP achieved exactly this. TOUGH2-MP divides a model up into sub-domains, each running on a separate process. Multiple processes together can deal with models that are multiple times larger. Other benefits of TOUGH-MP, particularly the speed improvement, will be discussed in later sections.

2.5 Generalised model size limit

So what does all this mean to a TOUGH2/AUTOUGH2 modeller? The simple solution is to compile dynamic code as a 64-bit application, and run it on a 64-bit OS. It is interesting, however, to understand the approximate limit of 32-bit TOUGH2/AUTOUGH2 in terms of actual model size, the number of blocks for example.

A generalised case of a simple three-dimensional model can be used to demonstrate the approximate relationship between number of blocks in a model and the amount of memory required. Here we assume the 3-D (three-dimensional) model has a quadrilateral mesh with N blocks. The maximum number of connections, or element interfaces, is N x 3 theoretically. We assume that the models may require complex boundary conditions, implemented through generators that include rainfall, surface discharge, basement heat and mass inflow, side boundary recharge. Additional generators that have a time dependent values must also be included, e.g. time-varying rainfall, historic production data, and future makeup wells. We assume that there are N x 20 generator table entries for each model. Note that it is difficult to generalise the number of generator table entries, but fortunately the memory requirement is significantly less dependent on the number of generators than on the number of blocks and connections.

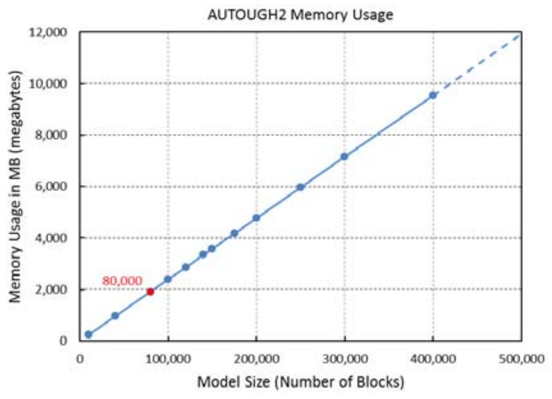

Figure 1: Memory usage of AUTOUGH2 as a function of model size

Figure 1 shows measured memory usage of AUTOUGH2 as a function of model size. These numbers were obtained using AUTOUGH2, which is slightly different from the original TOUGH2, but it should give a good estimate of the memory requirement for TOUGH2. It is clear to see the memory requirement is linearly dependent on the model size. With a 2 GB restriction, 32-bit AUTOUGH2 can only support models with approximately 80,000 blocks.

2.6 Discussion

The linear growth of memory requirement may seem reasonable. However, refining the mesh of a three-dimensional model usually means a four to eight fold increase in the number of blocks. Thus it does not take many refinements to give a very large memory requirement. Furthermore, TOUGH2 models with MINC (multiple interacting continua) processing obviously requires far more memory as it is common to divide each of the original blocks into several smaller fracture and matrix blocks.

Obviously, users of TOUGH2/AUTOUGH2 should be encouraged to move on to 64-bit systems if they wish to run larger models. Upgrading the codes to use dynamic memory allocation will also bring benefits.

3. SIMULATION SPEED AND PARALLELISATION

Improving the speed of TOUGH2/AUTOUGH2 simulations can be most simply achieved by upgrading to new and improved hardware. However, some of the advancement in new hardware can only be utilised by using appropriately updated compilers. This is because a lot of the recent improvements in CPU technology come in the form of new instruction sets. These instruction sets are designed in the hardware to complete certain operations with less CPU clock cycles than was previously possible. Only with updated compilers that understand the new instruction sets, is it possible to fully utilise the capability of the new hardware. This is especially true for applications like TOUGH2 which use a lot of floating point operations, which many of these new instruction sets are designed to carry out quickly.

It is also important to note how CPU technology has grown in power in recent years. Instead of increasing the CPU clock speed (more and more difficult for a number of reasons) to achieve higher performance, it is more common to increase the number of computation operations per clock cycle. Apart from adding new instruction sets as mentioned previously, often performance is enhanced by adding more cores into one CPU, or more CPUs into one computer box. Then further improvements in speed can only be obtained by dividing the computation up and running it in parallel.

Traditionally parallel computing was achieved by using large supercomputers, each containing many CPUs with dedicated interlinks, memory, specialised OS, and code specifically programmed to utilise multiple CPUs. As the technology progressed, clusters began to be used, formed by linking many individual computers, to execute parallel simulations previously only possible on supercomputer. Multiple core/CPU PCs are also becoming more common. Both clusters and multi-core PCs are now very affordable. Standardisation of parallel programming has also played an important role in ensuring the portability of parallel simulation codes.

3.1 TOUGH2-MP

TOUGH2-MP is the result of a major effort by Lawrence Berkeley National Laboratory to parallelise the original TOUGH2 code. TOUGH2-MP divides a given model into sub-domains, each treated like a smaller problem with the computation implemented on a single processor. The integrity of model across the whole domain is kept by synchronising primary variables across the boundaries of sub-domains. Since all of these sub-domain problems can be solved simultaneously by different processors, the total length of the simulation time can be cut down to the time required for each of the smaller sub-domains.

We have tested TOUGH2-MP on a variety of machines: multi-core PCs, an HPC (high performance computer which is closer to supercomputers in the traditional sense), and a cluster. Speed-up tests were carried out to see how much performance improvement is possible with the systems available to us.

3.2 Setting up TOUGH2-MP

Unlike the original TOUGH2, getting TOUGH2-MP to work is not always easy, especially with different machines and OS. Instead of simply compiling the TOUGH2-MP source code directly on the targeted platform, TOUGH2-MP additionally requires users to ensure MPI (the Message Passing Interface) [1] works on the target system and it is compatible with the compilation of TOUGH2-MP source codes.

MPI, an industry standard for parallel computation, is used by TOUGH2-MP to deal with the communications between processors (nodes in a cluster or cores/CPUs in a single computer box) while each processor runs a simulation similar to the original TOUGH2. Among different implementations of MPI, Open MPI [2] works straightforwardly on Linux powered systems, while we have found it easier to use MPICH2 [3] on Windows machines.

After setting up MPI, the compilation of TOUGH2-MP takes several steps. It is relatively easy on Linux-based systems. The code distribution comes with batch-processing scripts and make files which greatly ease the process. On Windows, however, users are provided with less help in compiling the two base libraries: METIS [9] and Aztec [12]. METIS is the domain partitioning software, while Aztec is a parallel linear equations solver. These steps can be difficult to follow if the user is not familiar with compiling complex codes. It is especially tricky to deal with these codes, because they are designed for multiple platforms and written with mixed computer languages.

3.3 Performance gains

Of the computers on which we tested TOUGH2-MP, two were eight-core Dell Precision T5500 PCs running at 2.66 GHz (gigahertz), one running Debian GNU/Linux 6.0 and one running Windows 7. We had to use quite different setups to make each system work. The default Open MPI and GCC compiler were used on Linux, and MPICH2, Intel Fortran, and Microsoft Visual C++ were used on Windows. The performance of these two machines is nearly identical, and so results from only one machine are shown.

The HPC is two IBM Systems x3755 servers linked together, each running 4 dual-core CPUs at 2.6 GHz, totalling 16 cores. The Red Hat Enterprise Linux OS is installed.

BeSTGRID [4] is a cluster, or perhaps more appropriately a computing grid, that is composed of numerous computational resources from many universities and research organisations from New Zealand. BeSTGRID can be accessed via a job management system. Tools such as compilers and MPI are preloaded and users can compile TOUGH2-MP and upload the executable along with models to a job queue.

The test problem we used is a 3-D model of the Palinpinon geothermal field (using EOS1 for pure water) with 64754 blocks, 192209 connections, and a total well table length of 4070. Timing and speedup of the simulations is presented in Figure 2 and Figure 3. All speedup factors are calculated by using the simulation time of the two processor case as the base case (TOUGH2-MP will not run on fewer than two processors). The 'ideal' linear performance gain of doubling speed by doubling the number of processors is shown in the figure for comparison.

3.4 Discussion

Figure 3 shows that for this test problem, the speedup factor decreases as the number of processors increases, and is noticeably less than the linear ideal for all three platforms tested- that is, doubling the number of processors results in less than a doubling of speed. Of these three, the HPC showed the best speedup behaviour (even though it was the slowest in terms of computer time) and the 8-core PC the worst. These differences are, however, not great, and are likely due to differences in computer architecture and the way the processors are linked together, with the HPC and BeSTGRID being better optimized for parallel computation.

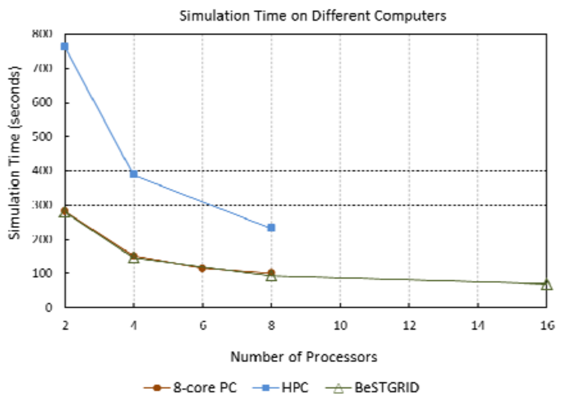

Figure 2: Simulation time on different computers.

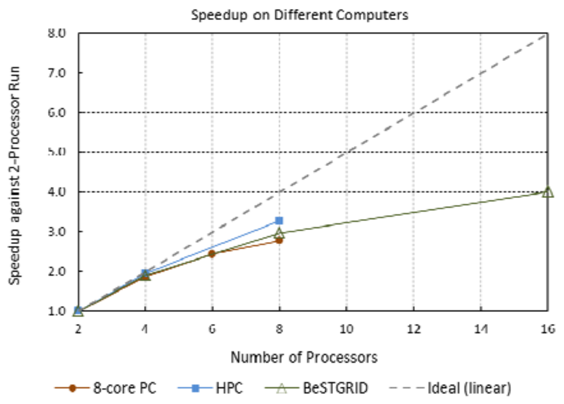

Figure 3: Speedup on different computers.

Previously published results for TOUGH2-MP [8, 13] have shown results closer to ideal linear speedup for other problems, but typically for very large models with over one million blocks. Speedup results are very problem-dependent, with the benefits of parallelization being greater for larger and more complex problems. However, for the test problem here, which is of a size more typical of current geothermal models, a useful degree of speedup is seen even for a relatively modest numbers of processors. Figure 3 indicates that using larger numbers of processors on BeSTGRID would result in some further speedup for this problem, although there would likely be little gain in going beyond 32 or 64 processors.

In practice, the performance measure of most interest to modellers is not speedup so much as absolute simulation time. Figure 2 shows that machines that give the best speedup do not necessarily give the shortest simulation time: for this problem, the 8-core PCs and BeSTGRID give similar simulation times for up to eight processors, while the HPC lags significantly behind both. Here the HPC's slightly better speedup behaviour is overshadowed simply by the relative slowness of its individual processors. The HPC was built in 2006, whereas the 8-core PCs were purchased in 2010 and have processors with only marginally higher clock speed but significantly updated instruction sets and other improvements.

Larger parallel machines like BeSTGRID, with access to hundreds or thousands of processors, will likely become increasingly useful as model sizes continue to increase. However, there is no guarantee that simply using ever-larger numbers of processors will give any benefit, as the speedup results will still depend on the problem being solved.

4. FUTURE WORK

A GPU (graphics processing unit) is a specialised circuit originally designed for assisting the output of graphics in a computer system. In a modern PC, it is a crucial component for 3-D graphics rendering, video decoding, and many other floating point operations related to graphics output. GPUs are becoming more powerful and highly optimised in recent years. In many cases, a GPU is actually more suitable for floating point calculations than a CPU. Current GPUs may contain hundreds of computational cores, at a relatively low total cost.

Some manufactures have started to allow GPUs to be used in general programming and to assist certain floating point operations that were done by CPUs before [5, 6]. This has moved into the field of scientific calculations. We have started investigating this development. At the moment, the GPU industry has not settled on widely accepted standards which are important to ensure the future portability of code. There are many issues that need to be looked at, but we think there is potential here to further improve the speed of the TOUGH2 code family by sharing some of the workload between CPUs and GPUs.

5. CONCLUSIONS

With the older 32-bit systems, TOUGH2/AUTOUGH2 users will find the model size restricted by the 2 GB process memory limit. This translates to a general three-dimensional model of about 80,000 blocks. To overcome the limit, it is advisable to move the simulations on to 64-bit platforms, as 64-bit applications on 64-bit operating systems have a much greater memory limit. We have also updated AUTOUGH2 codes to allocate memory dynamically which allows more efficient use of memory.

To speed up the simulations, users should pay attention to updating compilers as well as upgrading computers. To cut down the simulation time further, parallel simulation is needed. TOUGH2-MP, a parallel version of the original TOUGH2, is able to bring a beneficial speed-up to many different platforms. In our tests, TOUGH2-MP’s speed-up on PCs is not as good as on an HPC or cluster. The affordability of multi-core PCs these days, combined with the ease of regularly upgrading hardware, makes running TOUGH2-MP on PCs very attractive.

ACKNOWLEDGEMENTS

Thanks go to everyone in the Geothermal Research Group, at the Department of Engineering Science who use AUTOUGH2 every day. Your struggles with memory limits and long simulation times have been the driver for this work.

REFERENCES

[1] Message Passing Interface Forum. (2011, 13th October). *The Message Passing Interface (MPI) standard*. Available: http://www.mcs.anl.gov/research/projects/mpi/

[2] The Open MPI Project. (2011, 13th October). *Open MPI: Open Source High Performance Computing.* Available: http://www.open-mpi.org/

[3] Argonne National Laboratory. (2011, 13th October). *MPICH2 : High-performance and Widely Portable MPI*. Available: http://www.mcs.anl.gov/research/projects/mpich2/

[4] Ministry of Research, Science & Technology eResearch programme. (2011, 13th October). *BeSTGRID: Broadband enabled Science and Technology GRID*. Available: https://www.bestgrid.org/

[5] NVIDIA Corporation. (2011, 13th October). *CUDA*. http://developer.nvidia.com/category/zone/cuda-zone

[6] Khronos Group. (2011, 13th October). *OpenCL - The open standard for parallel programming of heterogeneous systems*. http://www.khronos.org/opencl/

[7] Bullivant, D. P., O'Sullivan, M. J., and Zyvoloski, G. A., "Enhancements of the MULKOM geothermal simulator," in *Proceedings of 13th New Zealand Geothermal Workshop*, University of Auckland, 1991, pp. 175-182.

[8] Elmroth, E., Ding, C., and Wu, Y.-S., "High Performance Computations for Large Scale Simulations of Subsurface Multiphase Fluid and Heat Flow," *J. Supercomput.*, vol. 18, pp. 235-258, 2001.

[9] Karypis, G. and Kumar, V., "METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing FillReducing Orderings of Sparse Matrices Version 4.0," Minneapolis: University of Minnesota, Department of Computer Science, 1998.

[10] O'Sullivan, M. J., Pruess, K., and Lippmann, M. J., "State of the art of geothermal reservoir simulation," *Geothermics*, vol. 30, pp. 395-429, 2001.

[11] Pruess, K., Oldenburg, C., and Moridis, G., "TOUGH2 User's Guide Version 2.0," Berkeley, California: University of California, 1999.

[12] Tuminaro, R. S., Heroux, M., Hutchinson, S. A., and Shadid, J. N., "Official Aztec User's Guide Version 2.1," Albuquerque: Sandia National Laboratories, 1999.

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

[14] Zhang, K., Wu, Y. S., and Pruess, K., "User's Guide for TOUGH2-MP - A Massively Parallel Version of the TOUGH2 Code," Berkeley, California, USA: Earth Sciences Division, Lawrence Berkeley National Laboratory, 2008.