PROCEEDINGS, Twenty-Fourth Workshop on Geothermal Reservoir Engineering
Stanford University, Stanford, California, January 25-27, 1999
SGP-TR-162

AUTOMATIC CALIBRATION OF GEOTHERMAL RESERVOIR MODELS THROUGH PARALLEL COMPUTING ON A WORKSTATION CLUSTER

Stefan Finsterle and Karsten Pruess

Lawrence Berkeley National Laboratory
Earth Sciences Division
University of California
Berkeley, CA 94720

ABSTRACT

ITOUGH2 is an optimization code that allows estimation of any input parameter of the nonisothermal, multiphase flow simulator TOUGH2. ITOUGH2 inversions are computationally intensive because the so-called forward problem, i.e., the simulation of fluid and heat flow through the geologic formation, must be solved many times for different parameter combinations to evaluate the misfit criterion or to numerically calculate sensitivity coefficients. Most of these forward runs are independent from each other and can therefore be performed in parallel. Message passing based on the Parallel Virtual Machine (PVM) system has been implemented into ITOUGH2 to enable parallel processing of forward simulations on a heterogeneous network of Unix workstations or networked PCs that run under the Linux operating system. This paper describes the PVM system and its implementation into ITOUGH2. Examples are discussed, demonstrating the use, efficiency, and limitations of ITOUGH2-PVM.

INTRODUCTION

Inverse modeling is a technique to estimate hydrogeologic and thermal properties of a geothermal reservoir by automatically matching the output of a numerical model to data collected in a laboratory experiment, field test, or during production. The method requires the development of an appropriate conceptual and numerical forward model, which in principle must be able to reproduce the observed data. In geothermal applications, the prediction of the system behavior usually calls for a rather sophisticated numerical simulator, capable of modeling coupled multiphase flow and transport processes in heterogeneous, fractured-porous media. We use the TOUGH2 simulator (Pruess, 1991) for this task.

The ITOUGH2 code (Finsterle, 1999abc) provides inverse modeling capabilities for TOUGH2. ITOUGH2 inversions are computationally intensive because the forward problem—the prediction of state variables using the TOUGH2 simulator—must be solved many times to evaluate the misfit between the model predictions and the observed data, and to numerically calculate sensitivity coefficients. ITOUGH2 has been successfully applied to the analysis of laboratory experiments (see, for example, Finsterle and Persoff, 1997). Preliminary attempts have been undertaken to calibrate a geothermal reservoir model based on production data (White, 1995; Finsterle *et al.*, 1997; Bullivant and O'Sullivan, 1998). These studies concluded that the respective forward models must be refined in order to reduce the estimation bias as a result of an oversimplified description of the geothermal system. Such a refinement comes at the expense of increased computational burden for each forward simulation, making inversions using standard ITOUGH2 prohibitive.

Most of the forward runs required as part of an ITOUGH2 inversion are independent from each other and can therefore be performed in parallel. Since obtaining the forward solutions consumes the bulk of the CPU time used in an ITOUGH2 inversion, with only a few percent of the time spent in the optimization routines, processing individual TOUGH2 runs in parallel has the potential to significantly reduce the turn-around time of an ITOUGH2 inversion. Message passing based on the Parallel Virtual Machine system (PVM, Geist *et al.*, 1994) has been implemented into ITOUGH2 to enable parallelization of TOUGH2 forward simulations on a heterogeneous network of Unix workstations. We will discuss the efficiency and limitations of ITOUGH2-PVM to

solve inverse problems for the characterization of geothermal reservoirs.

PARALLELIZATION CONCEPT

The main task of ITOUGH2 is to initiate multiple TOUGH2 simulations with different parameter sets, and to analyze each of the corresponding model outputs at selected calibration points. A new, improved parameter set is then proposed following a certain strategy to minimize the objective function. The objective function is a measure of misfit between the model predictions and the measured data. The degree to which an ITOUGH2 job can be parallelized and the maximum attainable efficiency depends on the minimization algorithm chosen, the number of parameters to be estimated, and the number of processors available for parallelization. In order to select the number of processors one would reasonably want to engage, m_{procs}, and to estimate the potential increase in efficiency, it is important to know which tasks are parallelized in a given ITOUGH2-PVM application. Table 1 lists the various analyses performed by ITOUGH2 and the minimization algorithms available, and indicates which forward simulations can be run in parallel, and which ones must be executed sequentially.

Table 1. Summary of Tasks Parallelized

Method	Parallel	Serial	m_{procs}
Levenberg-Marquardt	Jacobian λ-steps	First run	n
Gauss-Newton	Jacobian	First run	n
Simplex algorithm	Initial simplex *n*-contraction Final Jacobian	First run *1D*-contraction Reflection Expansion	n
Grid search	All runs	-	n_{runs}
Sensitivity	Jacobian	First run	n
FOSM	Jacobian	First run	n
Monte Carlo	All but first	First run	n_{MC}

n Number of parameters to be estimated.
n_{runs} Number of parameter sets to be evaluated.
n_{MC} Number of Monte Carlo simulations.
FOSM First-order-second-moment uncertainty analysis.

It is obvious that the many forward runs performed during a Monte Carlo study can be executed in parallel with a maximum increase in efficiency due to complete independence of the individual simulations. Such an analysis is supported by ITOUGH2-PVM; for an example, see Finsterle (1998).

We focus here on the Levenberg-Marquardt minimization algorithm as the prime method used in ITOUGH2 to solve the nonlinear least-squares problem. Levenberg-Marquardt is a gradient-based method that requires evaluating sensitivity coefficients of the calculated system response at the calibration points with respect to each parameter to be estimated. In ITOUGH2, the sensitivity coefficients are calculated based on the perturbation method using the following forward finite difference quotient:

$$J_{ij} = \frac{\partial z_i}{\partial p_j} \approx \frac{z_i(\mathbf{p}; p_j + \delta p_j) - z_i(\mathbf{p})}{\delta p_j} \qquad (1)$$

Here, z_i is the calculated system response (e.g., flow rate, temperature, tracer concentration) at calibration point i, $i = 1,\ldots,m$, and $\mathbf{p}$ is the parameter vector of length n. The evaluation of the Jacobian matrix $\mathbf{J}$ thus requires $n+1$ TOUGH2 simulations: one run is used to obtain the elements $z_i(\mathbf{p})$, followed by n additional runs, each providing one column of the Jacobian matrix. In each run, one of the parameters is perturbed by a small amount δp_j. These n runs with the perturbed parameter sets are independent and are thus parallelized in ITOUGH2-PVM. The maximum number of processors to participate in this parallelized calculation is therefore n. The initial forward run is not performed in parallel. Once the Jacobian is evaluated, the Levenberg-Marquardt algorithm proposes an update vector $\Delta\mathbf{p}$, which depends on the Levenberg parameter λ as follows:

$$\Delta\mathbf{p} = \left(\mathbf{J}^T\mathbf{C}_{zz}^{-1}\mathbf{J} + \lambda\mathbf{D}\right)^{-1}\mathbf{J}^T\mathbf{C}_{zz}^{-1}\mathbf{r} \qquad (2)$$

Here, $\mathbf{C}_{zz}$ is the observation covariance matrix, $\mathbf{D}$ is an $n \times n$ diagonal matrix with elements $D_{ii} = \left(\mathbf{J}^T\mathbf{C}_{zz}^{-1}\mathbf{J}\right)_{ii}$, and $\mathbf{r}$ is the residual vector holding the differences between the observed and predicted system response. If the Levenberg parameter λ is large, $\Delta\mathbf{p}$ becomes a robust step parallel to the steepest-descent direction with a step length approaching zero; if λ is zero, $\Delta\mathbf{p}$ is identical to a Gauss-Newton step with its quadratic convergence rate.

The following procedure is used in the original ITOUGH2 implementation: if step $\Delta\mathbf{p}$ is successful (i.e., the objective function $S = \mathbf{r}^T\mathbf{C}_{zz}^{-1}\mathbf{r}$ becomes smaller, that is, $S(\mathbf{p}+\Delta\mathbf{p}) < S(\mathbf{p})$), λ is reduced by

the Marquardt parameter ν, and a new Jacobian matrix is evaluated using Eq. (1); if the step is not successful (i.e., leads to an increase in the objective function), λ is increased by ν, and a new parameter vector $\mathbf{p}^{(k+1)} = \mathbf{p}^{(k)} + \Delta\mathbf{p}$ is calculated at iteration k using Eq. (2), until a successful step is obtained.

In ITOUGH2-PVM, the approach taken is to initiate n_{procs} forward runs simultaneously with various λ values, where $n_{procs} \le m_{procs}$ is the number of actually available child processes. The simulation that yields the lowest objective function is identified. If this run constitutes a successful step, optimization continues; if it is an unsuccessful step, another n_{procs} runs are performed with $\lambda_i = \lambda_0 \cdot \nu^i$, $i = 1, \ldots, n_{procs}$, where λ_0 is the Levenberg parameter that yielded the lowest value of the objective function in the previous set of runs. The procedure is repeated until a successful step can be taken or one of the convergence criteria is met.

This approach is equivalent to performing a limited search for the minimum along the line of possible Levenberg-Marquardt steps with various values for λ at each iteration. This may further accelerate the Levenberg-Marquardt algorithm, as will be shown below. However, testing multiple parameter steps in parallel and picking the one with the lowest objective function changes the solution path compared with that taken by standard ITOUGH2. The user has therefore the choice to restrict parallelization to the evaluation of the Jacobian matrix.

The parallel evaluation of the Jacobian has to be completed first, before a new parameter vector can be calculated using Eq. (2). This means that some processors may be idle until all columns of the Jacobian are evaluated. The number of child processes n_{procs} should be selected such that all processors are busy. For example, if m_{procs}=n=8 and 7 processors of equivalent speed and work load are available, it is reasonable to select only n_{procs}=4 to avoid 6 processors being idle for 50% of the time during the calculation of the Jacobian. Similar restrictions apply to most algorithms listed in Table 1 (Finsterle, 1998), with the notable exception of grid search and Monte Carlo simulations, in which no waiting times exist, i.e., all available processors can be used simultaneously regardless of their relative speed. Note that idle time may not pose an inefficiency in itself because the corresponding processor is available at that time for other calculations.

PVM IMPLEMENTATION

The parallelization concepts outlined in the previous section are implemented into ITOUGH2 using the Parallel Virtual Machine (PVM) system (Geist *et al.*, 1994). PVM is a freely available software package that permits a heterogeneous collection of Unix workstations networked together to be viewed as a single parallel computer. PVM must be installed on all hosts in the cluster, following the instructions in Geist *et al.* (1994).

ITOUGH2-PVM is programmed following the "node-only" model where multiple instances of the same code are executed on all computers in the network. One process—the parent process—takes over the non-computational responsibilities such as spawning child processes, initialization, distribution and collection of data, and synchronization. In addition, the parent process contributes to the computation itself (see tasks listed in the third column of Table 1).

Figure 1 shows a simplified flow chart of ITOUGH2-PVM. The source codes for the parent and child processes are identical. ITOUGH2-PVM first enrolls itself into PVM, obtains its task identifier (TID), and determines whether it is a parent or a child process. The parent process reads the standard TOUGH2 and ITOUGH2 input files. Hosts are added to the virtual machine and child processes are spawned. Next, the parent process sends the name of the working directory with the input files to the child processes.

As soon as the directory name is received by a child process, it starts reading the TOUGH2 and ITOUGH2 input files, which were automatically copied to the host by a Unix shell script. After input reading is completed, the child process waits for the arrival of data or parameter sets sent by the parent process. In the meantime, the parent process performs the initial forward run with the base-case parameter set (except for grid search).

The results from the initial run are broadcast to all hosts if performing either sensitivity analyses, first-order-second-moment (FOSM) uncertainty propagation analyses, or Monte Carlo simulations. No such step is required when performing optimization using the Levenberg-Marquardt, Gauss-Newton, Downhill Simplex, or Grid Search method. After the initial run, the parameter set is updated according to the selected algorithm. The updated parameter set is then sent to one of the child processes. This procedure is repeated until all child processors are busy.

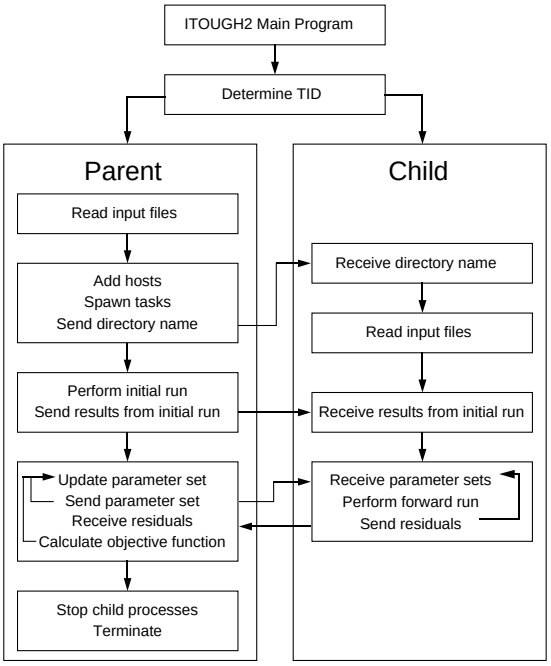

Fig. 1. Simplified ITOUGH2-PVM flow chart.

The child processes perform one TOUGH2 forward calculation with the parameter set they have received from the parent process. After completion of the run, they send the resulting residuals to the parent process. They then wait for the next parameter set to arrive. The parent process checks for incoming residual vectors, and processes them according to the selected method. If convergence is achieved or one of the child processes signals that it was stopped, the parent process stops all child processes before it continues with the error analysis and terminates.

In ITOUGH2-PVM, the amount of data transferred between the parent and child processes is relatively small. The parent process sends the parameter vector of length n, and the child process returns the residual vector of length m. Both n and m are of order ten to a few thousand, making efficient message passing a minor issue even on a relatively slow network.

In addition to the parameter and residual vector, some flags and iteration statistics are exchanged between the parent and child processes. For example, the parent process sends an integer indicating which column of the Jacobian matrix is to be evaluated with the corresponding perturbed parameter set. Upon completion of the forward simulation, the child process returns this flag, so that the Jacobian matrix can be properly assembled by the parent process.

The parent process can be suspended for a short time (typically 1 second) each time it checks for incoming data from the child processes. This prevents the parent process from spending time in a nonproductive loop, freeing its CPU and allowing the user to assign an additional child process on the computer running the parent process. Some time is lost—on average half a second per parallelized forward run—because the parent process waits despite having received new data that would be ready to be processed. For large inverse problems, this loss is negligible.

PERFORMANCE COMPARISON

General remarks and conclusions regarding the performance of ITOUGH2-PVM in comparison with standard ITOUGH2 are difficult to make, mainly because the attainable efficiency improvement strongly depends on the number, relative speed, and workload of the individual computers in the workstation cluster. As briefly mentioned above, the choice of the minimization algorithm and the solution path taken during the optimization also affect the performance. An important factor is the number of available hosts, n_{procs}, as related to m_{procs}, the maximum number of tasks that can potentially be parallelized in a given application (see last column in Table 1). Furthermore, the decision whether the parent process be assigned to the slowest or the fastest machine in the cluster also depends on the application. Finally, if one computer is significantly slower than all the others, it may be better not to use it at all under certain circumstances; in other cases, however, even a very slow machine can make a contribution to the overall performance of an inversion.

Some of the factors affecting the performance of ITOUGH2-PVM, i.e., m_{procs} (Table 1), the general characteristics of the minimization algorithm, and the number of hosts available, are known at the time of the run. Others, especially the workload on the computers in the cluster, are difficult to predict. The user has to decide (1) how many child processes to initiate, and (2) whether to start the parent process on a fast or a slow machine. These decisions are relatively easy to make despite the complicated interaction of factors affecting the overall performance. The following performance tests show the speed improvements attainable in a typical ITOUGH2-PVM application. Additional examples, illustrating both optimal and pathological cases, can be found in Finsterle (1998).

Table 2 contains a list of Unix workstations used for running the sample problems. Their relative speeds as indicated in the last column were measured by running a typical ITOUGH2 application on a single processor, and normalizing the speed to the slowest machine in the cluster. Recall that it is not the CPU time but the turnaround time of a TOUGH2 forward run that determines the effectiveness of a specific workstation in the cluster.

Table 2. Computers in the Workstation Cluster

Host	Architecture, Operating System	Relative Speed
1	DEC Alpha, DEC OSF-1	4.3
2	Silicon Graphics, IRIX	4.1
3	Sun SPARC, 14 CPUs, Solaris	2.9
4	CRAY Y-MP, 16 CPUs, UNICOS	2.4
5	Sun 4, SPARCstation, Solaris	1.8
6	IBM/RS6000, AIX 3.2	1.0

SAMPLE PROBLEMS

The first example consists of running in parallel a geothermal inverse problem previously described in Finsterle *et al.* (1997). Six parameters representing hydrological, thermal, and geometric properties of a synthetic geothermal reservoir are estimated based on pressure, temperature, liquid and vapor flow rate data collected in the production, observation, and reinjection wells. The inverse modeling results are discussed in detail in Finsterle *et al.* (1997) and Finsterle (1999c).

The parameter estimation problem is solved using the Levenberg-Marquardt algorithm, limiting the number of child processes to $m_{procs}=n=6$ (see Table 1). Five iterations are performed for this benchmark example, requiring a total of 36 forward simulations, namely an initial run, 5 evaluations of the Jacobian matrix at 6 runs each, plus 5 runs to test whether the Levenberg-Marquardt-step was successful (no unsuccessful steps are performed in the first 5 iterations). According to Table 1, 6 of the 36 runs must be executed in sequence, and 30 can be performed in parallel. The parallel runs are expected to require about the same time as spent by $30/n_{procs}$ forward runs. The theoretical minimum time needed to solve the problem using ITOUGH2-PVM is therefore $(6+(30/6))/36=0.3$ of the time required by standard ITOUGH2, assuming that all processors are equally fast, that workload is balanced, and that all forward calculations take the same amount of time regardless of the parameter set being tested. Furthermore, PVM overhead is assumed to be negligible. In this example, the maximum attainable speed improvement is modest because of the relatively large fraction of runs that must be performed in sequence. If 24 instead of 6 parameters were to be estimated, the maximum achievable reduction with 6 child processes would be $(6+(120/6))/126=0.2$, i.e., close to $1/n_{procs}$.

We will look at different cluster configurations to analyze the performance of ITOUGH2-PVM. Table 3 shows the number of child processes and hosts selected for parallel execution, and the time reduction as compared to a serial run on the fastest machine in the respective cluster.

Table 3. Cluster Configuration and Time Reduction

Case	n_{procs}	Hosts (see Table 2)	Parent Host	Time Reduction
A	2	2, 3	2	0.64
B	2	2, 5	2	0.87
C	2	2, 5	5	1.08
D	3	1, 2, 3	1	0.51
E	6	1, 2, 3, 3, 3, 3	1	0.41
F	6	1, 2, 3, 3, 5, 6	1	0.82
G	6	4, 4, 4, 4, 4, 4	4	0.31

The first configuration (Case A) consists of only two hosts. Even though Host #2 is almost 50% faster than Host #3, each host takes over half of the runs during the evaluation of the Jacobian, i.e., the parent process and Host #2 remain idle for some time, waiting for Host #3 to finish its task. A greater speed difference (see Case B) or a smaller ratio n_{procs}/m_{procs} is needed to allow the faster process to take over some of the load of the slower process. Nevertheless, the time reduction factor of 0.64 due to parallelization with only two child processes can be considered satisfactory.

In Case B, the second machine in the cluster is significantly slower than the parent host. For each Jacobian evaluation, Host #2 performs 4 forward runs, whereas Host #5 completes 2 simulations, each of which takes about 4 times as long as a single run on the faster machine. As a result, a time saving of only 13% is achieved.

Since the six serial runs performed by the parent process will consume a significant portion of the total time, it is reasonable to select the fastest machine in the cluster as the parent host, as was done in Cases A and B. In Case C, the parent process was assigned to the slow machine, which in fact makes the turnaround time of the inversion longer as compared to a serial run performed on the faster

machine. The negative impact of selecting the wrong parent host decreases as the number of serial runs becomes small relative to the total number of forward simulations.

Cases D and E show the performance with 3 and 6 child processes, reducing the time to 51 and 41%, respectively. This performance is reasonably close to the maximum attainable efficiency with a time requirement of 30% of the serial run. Recall, that the time reduction is measured against the fastest computer in the cluster, and that waiting time is included as the 6 runs performed in parallel at each iteration exhibit different turnaround times. While Case E realizes the shortest time in this example, the overall performance gain is poor given the fact that resources from six computers are linked together. This is a direct result of the test case characteristics with its relatively few parameters, the significant fraction of serial runs, and a large ratio n_{procs}/m_{procs}, which makes the slowest machine become the limiting factor. This last point is illustrated in Case F, which performs almost as poorly as the pathologic Case B despite the larger number of hosts. The two-host configuration of Case A is certainly preferable, and may be considered the most efficient solution for this example.

Case G demonstrates that if identical processors are used, for example, by running ITOUGH2-PVM on a multiprocessor machine such as the CRAY Y-MP C90, the performance approaches the theoretical maximum efficiency.

In the final example, two parallelized inversions are performed, both using the Levenberg-Marquardt minimization algorithm. In the first run, only the evaluation of the Jacobian matrix is parallelized. In the second run, testing of new parameter sets is also performed in parallel using n_{procs} different values of the Levenberg parameter λ (see discussion of Eq. 2). As mentioned above, this amounts to determining the minimum of the objective function calculated at n_{procs} points along the curve given by all possible Levenberg-Marquardt steps at each iteration.

Figure 2 shows how the objective function is reduced by the Levenberg-Marquardt algorithm. Both runs start at the same initial value. After two iterations, the objective function is reduced to 56% of the initial value if using standard ITOUGH2 or a version in which only the Jacobian is evaluated in parallel. Parallel testing of n_{procs}=4 steps with different Levenberg parameters leads to a much faster decline of the objective function to 8%, and the minimum is nearly identified after 3 iterations, whereas it takes 6 iterations for standard ITOUGH2 to reach a similarly low level. In this example, both approaches converge to the same best-estimate parameter set despite the fact that different solution paths are taken.

Figure 3 shows how the Levenberg parameter is updated between iterations for the two methods. A value of 10 was chosen for the Marquardt parameter, the factor by which the Levenberg parameter is multiplied (in case of an unsuccessful step) or divided (in case of a successful step).

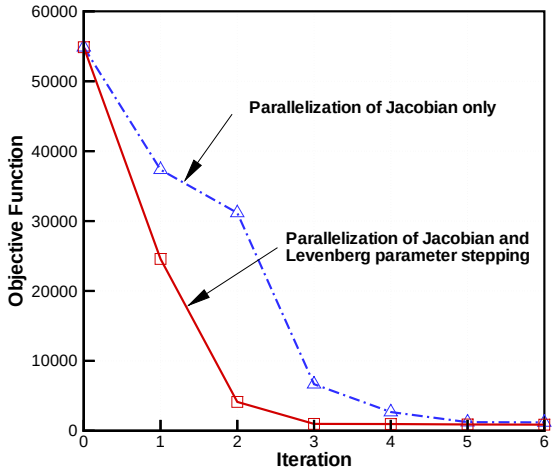

Fig. 2. Reduction of objective function with and without parallelization of Levenberg stepping.

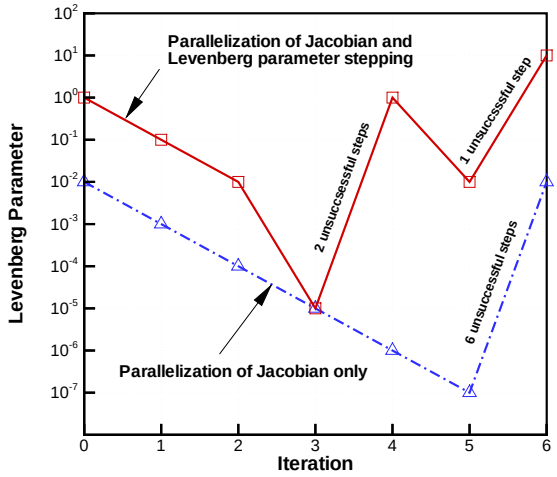

Fig. 3. Value of Levenberg parameter at each iteration with and without parallelization of Levenberg stepping.

In standard ITOUGH2, the Levenberg parameter can only be reduced by one order of magnitude between iterations. If an unsuccessful step is encountered, new parameter sets calculated with increasing Levenberg parameters are tested sequentially. For example, 6 trial runs were performed between iterations 5 and 6, before a parameter set was created that successfully reduced the objective function. In the parallelized version, the Levenberg parameter can be reduced by more than one order of magnitude between successful iterations (see, for example, between iterations 2 and 3), and testing of smaller, more robust steps with increased λ's is more efficient (see, for example, between iterations 3 and 4).

Note that simultaneous testing of n_{procs} parameter sets is likely to take more time than testing only one set as in standard ITOUGH2 because the slowest machine in the cluster is the limiting factor. Moreover, different parameter sets require different execution times due to different time-stepping and convergence characteristics of the Newton-Raphson iterations and the iterative linear equation solver. As a result, each ITOUGH2 iteration may take more time in the parallel version as long as successful steps are taken. However, unsuccessful steps are less frequent, and if multiple unsuccessful steps are encountered, parallelization has again an advantage over testing of parameter sets with sequentially increased Levenberg parameters.

SUMMARY AND CONCLUSIONS

A version of the inverse code ITOUGH2 has been developed that allows for parallel execution of independent TOUGH2 forward simulations on a heterogeneous cluster of Unix workstations or networked PCs running under the Linux operating system. The parallelization strategy depends on the minimization algorithm used. The performance of ITOUGH2-PVM as compared with standard, sequential execution of multiple forward runs strongly depends on the selected optimization method, the number of processors available, the ratio of the number of processors available to the maximum possible number of parallel processes, the relative speed of the machines in the cluster, the selection of the parent process, the work load on each machine, and the efficiency of the network.

Some of the advantages of choosing PVM over other, potentially more efficient message passing interfaces is that PVM is freely available, and its use is not restricted to high-performance computers, which often have limited access. However, multiprocessor machines can be included in the cluster and viewed as multiple hosts. ITOUGH2-PVM is easy to install and to use, and it is highly portable across platforms. Because parallelization in ITOUGH2-PVM occurs on a high level, modifications made to any of the TOUGH2 modules do not usually require any adaptation to ITOUGH2-PVM. The PVM code is fully integrated into ITOUGH2, i.e., the same source code can be used for sequential and parallel applications.

Parallelizing the time-consuming forward runs in ITOUGH2 provides the needed efficiency to deal with challenging inverse problems. It enables one to use a more sophisticated and more accurate forward model in an inversion, reducing the impact of systematic modeling errors, which strongly affect the outcome of an inversion. ITOUGH2-PVM is a significant step towards making automatic calibration of large-scale reservoir models part of standard geothermal engineering practice.

For more information about ITOUGH2, visit the Web Site at http://www-esd.lbl.gov/ITOUGH2.

ACKNOWLEDGMENT

We would like to thank C. Oldenburg and T. Xu for a careful review of the manuscript. This work was supported, in part, by the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Geothermal Technologies, of the U.S. Department of Energy, under Contract No. DE-AC03-76SF00098.

REFERENCES

Bullivant, D. P. and M. J. O'Sullivan (1998), "Inverse Modelling of the Wairakei Geothermal Field," *Proceedings*, TOUGH Workshop '98, Report-41995, Lawrence Berkeley National Laboratory, Berkeley, Calif., 53–58.

Finsterle, S. (1998), *Parallelization of ITOUGH2 Using PVM*, Report LBNL-42261, Lawrence Berkeley National Laboratory, Berkeley, Calif.

Finsterle, S. (1999a), *ITOUGH2 User's Guide*, Report LBNL-40040, Lawrence Berkeley National Laboratory, Berkeley, Calif.

Finsterle, S. (1999b), *ITOUGH2 Command Reference*, Report LBNL-40041 (updated reprint), Lawrence Berkeley National Laboratory, Berkeley, Calif.

Finsterle, S. (1999c), *ITOUGH2 Sample Problems*, Report LBNL-40042 (updated reprint), Lawrence Berkeley National Laboratory, Berkeley, Calif.

Finsterle, S. and P. Persoff (1997), "Determining Permeability of Tight Rock Samples Using Inverse Modeling," *Water Resour. Res.*, (*33*)8, 1803–1811.

Finsterle, S., K. Pruess, D. P. Bullivant, and M. J. O'Sullivan (1997), "Application of Inverse Modeling to Geothermal Reservoir Simulation", *Proceedings*, Twenty-Second Workshop on Geothermal Reservoir Engineering, Stanford, Calif., January 27–29.

Geist, A., A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam (1994), *PVM: Parallel Virtual Machine—A User's Guide and Tutorial for Networked Parallel Computing*, MIT Press, Cambridge, MA.

Pruess, K. (1991), *TOUGH2—A General Purpose Numerical Simulator for Multiphase Fluid and Heat Flow*, Report LBL-29400, Lawrence Berkeley National Laboratory, Berkeley, Calif.

White, S. P. (1995), "Inverse Modeling of the Kawerau Geothermal Reservoir, NZ", *Proceedings*, 17th New Zealand Geothermal Workshop, Auckland, New Zealand, 211–216.