

TASK PARALLEL AND DATA PARALLEL COMPUTING FOR SUBSURFACE INVERSE CHARACTERIZATION PROBLEMS[†]

G. Mahinthakumar and J. P. Gwo

Center for Computational Sciences, Oak Ridge National Laboratory.

Oak Ridge, TN 37831

phone: 423-241-5628, fax: 423-241-2850, e-mail: kumar@ornl.gov.

KEYWORDS

Parameter Identification, Parallel Computing, Genetic Algorithms, Hydrology

ABSTRACT

We describe two subsurface inverse characterization problems where both task parallel and data parallel computing are used on massively parallel and workstation cluster platforms. Both applications use genetic search algorithms for finding the solution given input and output tracer signals. The first application involves both task parallel and data parallel computing on a massively parallel platform and is for subsurface source zone (biological activity zone) identification. The second application involves task parallel computing on workstation clusters and is for subsurface characterization of fracture networks. Both applications are based on the manager-worker model. Because of current limitations in different message passing interfaces, the first application required communication and I/O in MPI, PVM, and NX. We expect most of the features that required the use of PVM and NX to be available in the Globus (<http://www.globus.org>) enabled version of MPI so that MPI could be exclusively used. In the second application all communication is performed with PVM. This application uses dynamic task scheduling to improve load balancing within a heterogeneous workstation cluster. Here we present some performance results related to speedup and scalability for both these applications.

1. INTRODUCTION

The problem of deducing information about the subsurface given experimental tracer signal measurements can be classified as a subsurface inverse characterization problem. The solution to such problems can be extremely computationally demanding if the information required is spatially distributed. The commonly used inverse methods in subsurface science fall under two broad categories: stochastic, and deterministic. Here we use genetic search algorithms which fall under the deterministic category. The de-

terministic methods are based on minimizing the difference between the observed and computed breakthrough curves at the observation points by adjusting the values of the parameters required (e.g. curve fitting).

Genetic algorithms (GAs) are search procedures based on the mechanics of natural selection and natural genetics. The GA was developed by John H. Holland in the 1960s to allow computers to evolve solutions to difficult problems, such as function optimization and artificial intelligence. The basic operation of a GA is conceptually simple: (1) maintain a population of solutions to a problem (2) select the better solutions for recombination with each other, (3) use their offspring to replace poorer solutions. The combination of selection pressure and innovation (through crossover and mutation) generally leads to improved solutions, often the best found to date by any method (Goldberg 1989).

GA is being used in a wide range of applications including pattern recognition and matching, inverse modeling, and optimization. One of the drawbacks of GA is that it can be computationally demanding if the objective function evaluation is expensive. For example, in the simulations performed here the objective function is the root mean square error between the observed output signals and the computed output signals. In order to compute the output signals (or breakthrough curve) for each individual a forward transport simulation need to be performed. Each GA generation can consist of a population of hundreds of individuals implying hundreds of forward transport simulations for each generation. Fortunately, today's massively parallel computers or workstation clusters can be used to our advantage in these situations. In a massively parallel computing environment the objective function computations for an entire population of individuals (or each generation) can be performed in parallel since they are all independent of each other.

2. SIMPLE GENETIC ALGORITHM

Implementing a simple genetic algorithm (SGA) for a source identification problem is conceptually very simple. However, the performance of SGA for these problems

[†]To appear in *High Performance Computing 1999* (Editor: A. M. Tentner), *Proceedings of the 1999 Simulation Multiconference, San Diego, CA, Apr 11-15, 1999.*

depend on various strategies such as problem encoding, selection, and crossover. In addition, performance and ease of computing the objective function including adequate computer resources are also key to its success. In the simulations performed in this study the following major steps are involved: (1) Encode possible biological activity locations or fracture networks as binary strings, (2) Generate an initial random ensemble of these strings equal to the number of individuals in a population or population size, (3) Perform transport simulation for each individual by decoding the strings into source locations or fracture networks, (4) Compute the value of objective functions for each individual by computing the difference between the observed (stored in a file) and computed output signals, (5) Select the individuals that perform best (those giving smaller objective function values) using an appropriate selection strategy and mate the binary strings randomly (using an appropriate crossover strategy) to produce the next generation of individuals, (6) Repeat steps 3–5 until convergence or up to a prescribed maximal number of generations. Convergence criteria can be defined in terms of the best performing individual (minimum root square error) or the average performance (average root square error) of the entire population, and (7) If convergence is not achieved within the prescribed maximal number of generations then the source locations for the best performing individual of all the generations can be chosen as the optimal solution.

The original version of the SGA code (Goldberg 1989) is written by Ulrich Hermes of University of Dortmund, Germany. This code can be freely downloaded from the University of Dortmund anonymous ftp site under GNU licence agreements. The SGA code was then loosely coupled to the groundwater transport codes which compute the objective function. The objective function to be minimized here is the root square error between the computed and observed output signals. Below we describe the specifics related to each of the two applications that we consider here.

3. BIOLOGICAL ACTIVITY ZONE IDENTIFICATION PROBLEM

In many natural bioremediation feasibility studies the location of indigenous microbes is a critical factor in achieving good results. In most cases biostimulants are injected into the subsurface to stimulate bacterial growth to desirable levels so that they can eventually degrade or sequester the contaminants. Here we use genetic algorithms to determine possible biological activity zones from the results of a biostimulation experiment. In these experiments indige-

nous methanotropic bacteria are stimulated by continuous periodic injection of dissolved oxygen and methane (see Figure 1). The observed breakthroughs of methane are used to deduce possible biological activity zones in the subsurface. The same method could be extended to a number of other subsurface detection problems such as detection of DNAPL's using partitioning tracers and elucidation of geologic subsurface heterogeneity using non-reactive tracers.

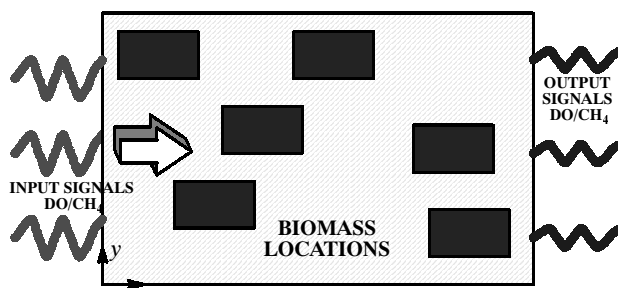


Figure 1. Problem setup for the biological activity zone identification problem

3.1. Parallel Implementation

For this application we use a parallel groundwater transport simulator (Mahinthakumar and Saied, 1999) to compute the objective function. This code is parallelized using a two-dimensional domain decomposition (in the x and y directions) using explicit message passing to exchange information between these domains (Saied and Mahinthakumar 1998; Mahinthakumar et al. 1997). The MPI (message passing interface) (e.g. Gropp et al. 1994) communication library is used for the message passing. This code has been tested on a variety of parallel architectures (Mahinthakumar and Saied 1998), but in this study we will exclusively use the Intel Paragon supercomputers at ORNL for our simulations. By nature this code is a data parallel code and in order to do the large number of GA simulations we had to build in simple task parallelism into this code. To minimize extensive modifications to the code we adopted a master-slave (or manager-worker) approach to perform the multi-tasking operations.

An example of the parallel computing layout is shown in Figure 2. For simplicity we have shown a scenario which uses a very small population size ($= 3$ or a small multiple of 3) per generation using 12 compute nodes of the Intel Paragon. Typically we have population sizes of 128 individuals using 512 compute nodes of the Intel Paragon with 4 compute nodes per transport simulation. The computationally trivial SGA (simple genetic algorithm) code is executed on one of the service nodes of the Paragon and the compute

[†]To appear in *High Performance Computing 1999* (Editor: A. M. Tentner), *Proceedings of the 1999 Simulation Multiconference, San Diego, CA, Apr 11–15, 1999.*

nodes are used for the computationally demanding 3D dual-component transport simulations. Since the current version of MPI does not support the spawn command where a code executable can be spawned on multiple nodes of a parallel machine we used the PVM (Parallel Virtual Machine) library (e.g. Giest et al. 1994) for spawning the transport code on the compute partition. The bit strings for each individual are also transmitted via the PVM send command to the entire compute partition. Once the entire compute partition is spawned and the bit strings for all the individuals are received then MPI communication library (much more efficient on massively parallel architectures than PVM) takes over. All subsequent communications are performed using MPI group operations. Transport simulation for each individual is then assigned to a group of nodes (typically 4) by MPI. The data parallel communications within each group is facilitated by the MPI group communicators. Once all the groups complete their transport simulations the computed objective function (root mean square error) is transmitted back to the master SGA code using PVM.

The average computation time taken for each transport simulation in most of our test cases is about 500 seconds on 4 processors of the Intel Paragon. Using the parallel environment described above we can perform the transport simulations for an entire generation requiring 128 individuals (128 transport simulations) using 512 processors in one sweep in about 650 seconds. Most of the additional 150 seconds are PVM spawn and clear operation overheads. The SGA code operations take less than 10 seconds per generation for most of our simulations. A 50 generation simulation takes about 9 hours on 512 processors of the Intel Paragon. All our runs are performed using the overnight and weekend batch queues on the 512 processor XPS/35 and 1024 processor XPS/150 Intel Paragons at ORNL. We have a restart option in the 'master' code so that the simulation can be restarted from the last completed generation.

Most of the PVM overhead is for spawning and clearing the tasks. Once the results are received at the master via PVM, the slaves had to be probed via the 'pvm_pstat' command to make sure they have been freed so that the next generation or sweep could be spawned. We replaced the 'pvm_spawn' call in the pvm library from 'nx_loadve' to 'nx_spawn' to improve the performance of the PVM spawn process on the Intel Paragon (more than 3 times improvement). In addition to the regular ascii I/O where one processor reads the data (a small number of common parameters)

and broadcasts the results to the other processors, some transport simulations (those involving heterogeneous flow fields) also required parallel binary I/O to read the velocity and flux fields. In order to improve this performance (by more than 5 times) a new set of MPI groups had to be defined so that only one group accesses the entire file and then transmits the data to the appropriate processors in the other groups. All parallel I/O is performed using the Intel Paragon NX calls.

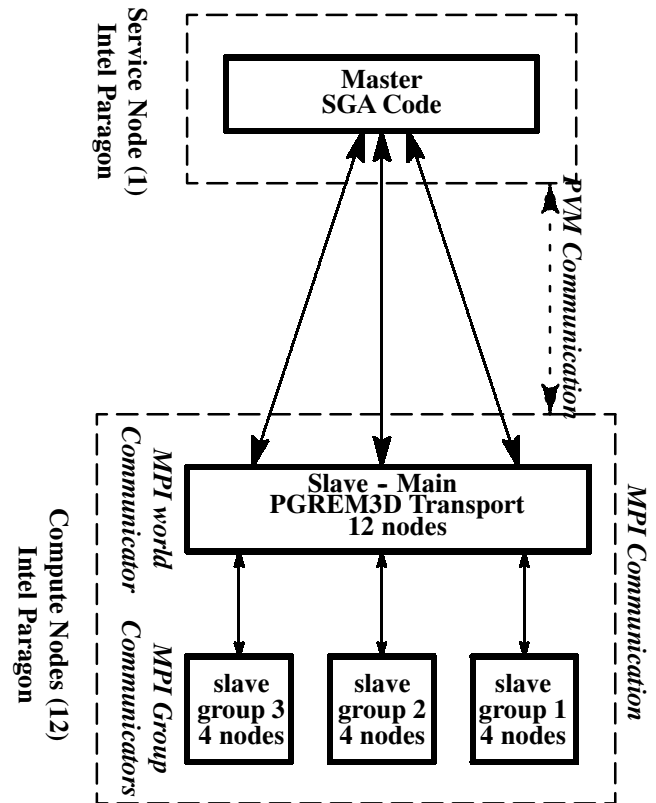


Figure 2. Layout of the Parallel Environment. The example shown here uses 1 service node and 12 compute nodes of the Intel Paragon. The master SGA code is executed on the service node. Each transport simulation requires 4 compute nodes.

3.2. GA Performance

The performance of GA for identifying the locations of possible biological activity zones in the subsurface is presented here. In the example shown in Figure 3 we are interested in identifying the centroids of 3 zones. The small spheres denote positions of GA individuals. The rectangular gridded regions are the actual locations of the biologically active zones.

[†]To appear in *High Performance Computing 1999* (Editor: A. M. Tentner), *Proceedings of the 1999 Simulation Multiconference, San Diego, CA, Apr 11–15, 1999.*

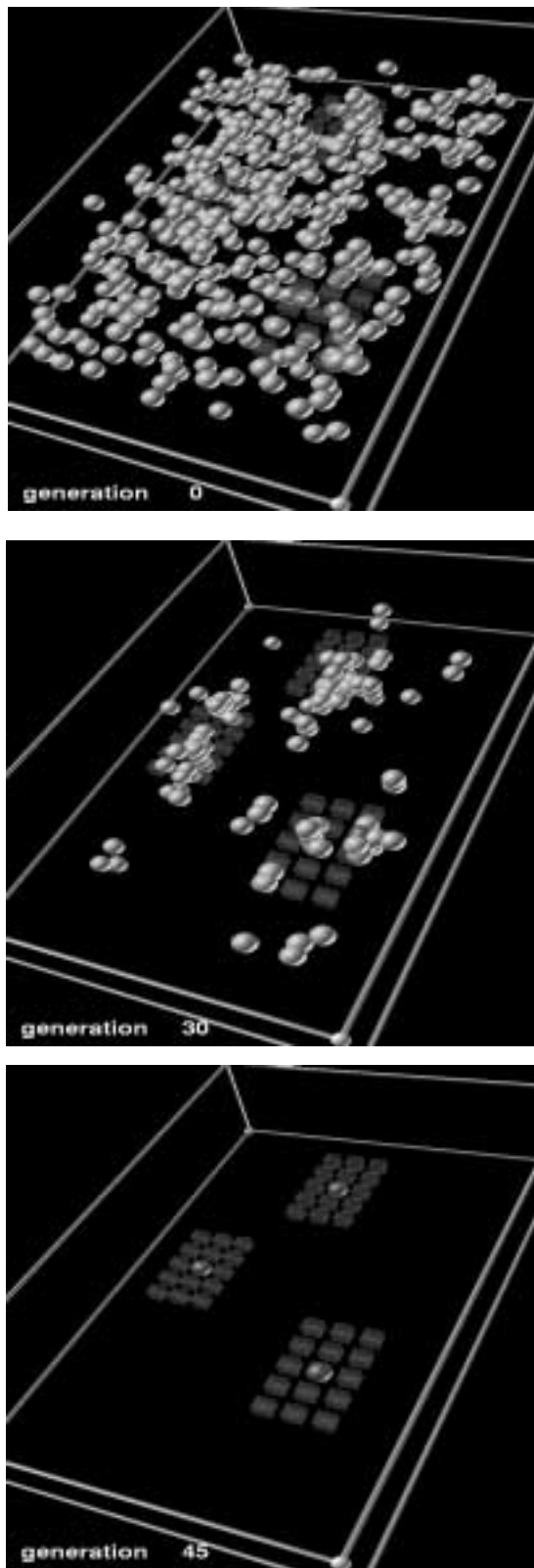


Figure 3. Performance of GA for the biological activity zone identification problem

The reference observed signals are precomputed based on the actual locations of these 3 zones. Here we simply verify that GA can find these locations given only the input and output signals corresponding to these locations. From Figure 3 we see that GA converged to the exact solution in 45 generations.

We looked at several more scenarios for this problem including multiple activity zone problems in heterogeneous flow fields. For all problems tested we were able to pinpoint to the exact locations in less than 100 GA generations. More details are available in Mahinthakumar et al. (1998).

3.3. Parallel Performance

Here we present the computational performance results related to biological activity zone identification problem. In Figure 4, we present timings for 2 GA generations as the number of processors computing the objective function increases from 1 to 8 (i.e., size of MPI groups). The number of individuals in each generation is 128 and each generation is completed with one PVM spawn sweep. PVM overhead includes time for spawning and clearing the tasks. The total time decreases up to 512 processors but increases slightly to 1024 processors. This is mainly due to the increased PVM overhead for spawning and clearing 1024 processors.

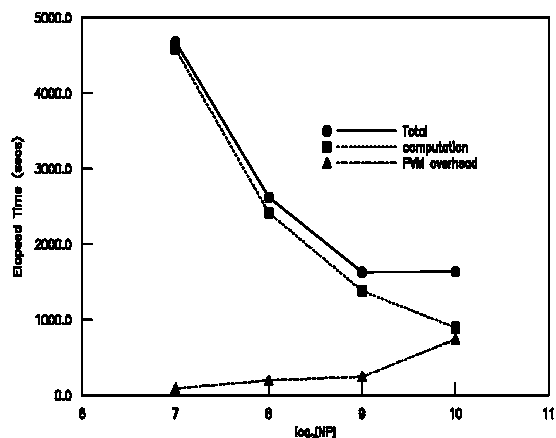


Figure 4. Increasing data parallelism. NP = number of processors. NP increases from 128 to 1024 processors. Timings are shown for 2 GA generations. Each generation has 128 individuals. The number of processors per individual (data parallelism) increases from 1 to 8. The number of tasks per PVM sweep is fixed at 128.

The best overall timing is for the case with 512 processors where data parallel groups of 4 processors are used to compute the objective function. For this case, the average time for each transport simulation is about 500 seconds on 4 processors of the Intel Paragon. Using the parallel environment described earlier we can perform the transport simula-

tions for an entire generation requiring 128 individuals (128 transport simulations) using 512 processors in one sweep in about 650 seconds. Most of the additional 150 seconds are PVM spawn and clear operation overheads. The SGA code operations take less than 10 seconds per generation for most of our simulations.

In Figure 5, we show the effect of increasing task parallelism. For the case with 64 processors 8 PVM sweeps are required to complete a generation, while for the case with 512 processors 1 PVM sweep is required to complete a generation. Thus the PVM overhead decreases considerably for the 512 processor case. Therefore we should try to chose the number of processors so that an entire generation is completed in a minimum number of PVM sweeps (best case is 1).

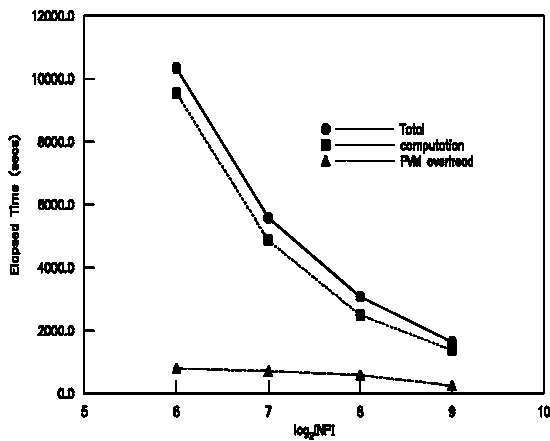


Figure 5. Increasing task parallelism. NP = number of processors. NP increases from 64 to 512 processors. Timings are shown for 2 GA generations. Each generation has 128 individuals. The number of processors per individual is fixed at 4. The number of tasks per PVM sweep increases from 8 to 128.

4. FRACTURE NETWORK CHARACTERIZATION PROBLEM

Accurate characterization of fracture networks of a subsurface environment is essential for predicting contaminant movement, and for designing remediation strategies in fractured porous media. In this application we use genetic algorithms to deduce fracture networks from observed tracer data.

This application is somewhat different from the previous application in that the objective function evaluation is less computationally demanding (a 2D single component transport problem) hence enabling us to use single workstations for each simulation. However, here we deal with a heterogeneous computing environment that can have load balancing problems. Our approach here is to dole out

new tasks as workstations become available until each generation is completed.

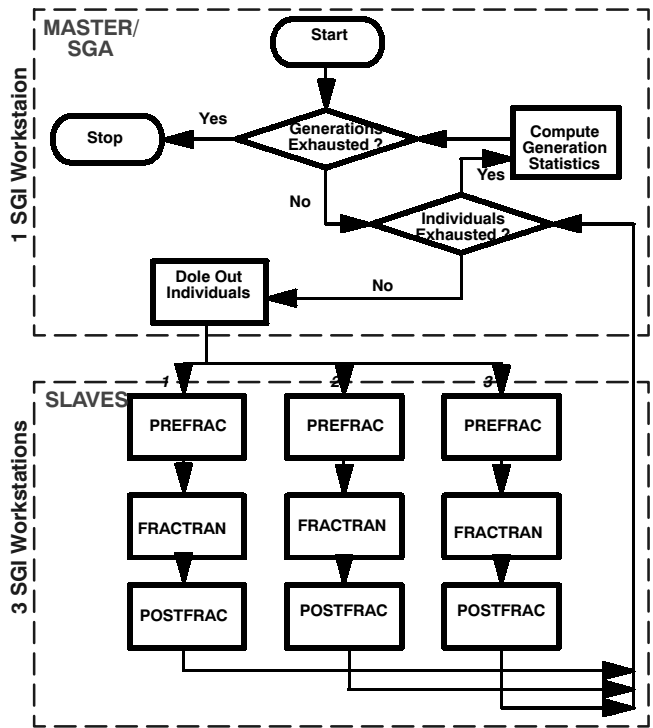


Figure 6. Flow chart of the virtual machine for characterizing fracture network. In the example shown the virtual machine consists of 3 workstations

One single vertical fracture at the center of a soil column is first used to generate the exact solution of solute concentration as a function of time. A population of 128 individuals is initialized in the SGA code. Each individual is simulated using the FRACTRAN (Sudicky and McLaren, 1992) code. A preprocessor (PREFRAC) is available and is interfaced with the SGA code to prepare input for FRACTRAN (Figure 6) or the main process in the virtual machine that generates the objective function. Solutions by FRACTRAN are post-processed by POSTFRAC to extract the evolution of outflow solute concentration, or the objective function. Root mean square error of the solution, as compared with the exact solution, is calculated by the post-processor and used as the fitness of the individual. PREFRAC, FRACTRAN, and POSTFRAC are the slave processes, in comparison with the master/SGA process. Execution of slave processes is on the first-available-first-serve principle, which is more likely to accommodate the dynamics of individual machine loads. Among the slave processes, PREFRAC acts as a secondary master that controls the execution of FRACTRAN and POSTFRAC, because convergence of the FRACTRAN simulation needs to be ob-

[†]To appear in *High Performance Computing 1999* (Editor: A. M. Tentner), *Proceedings of the 1999 Simulation Multiconference, San Diego, CA, Apr 11–15, 1999.*

tained iteratively. All of these processes are currently implemented using PVM on a cluster of SGI workstations in the Center for Computational Sciences at ORNL. It is therefore highly portable to other platforms and architectures.

4.1. GA Performance

The GA performance is presented for this problem in Figure 7 for a hypothetical case involving a single fracture. The GA does not know where the fracture is located except for the corresponding input and output signals. For the initial generation of individuals a random distribution of fractures is assumed along grid lines. We see that GA finds the exact solution for this problem in about 65 generations.

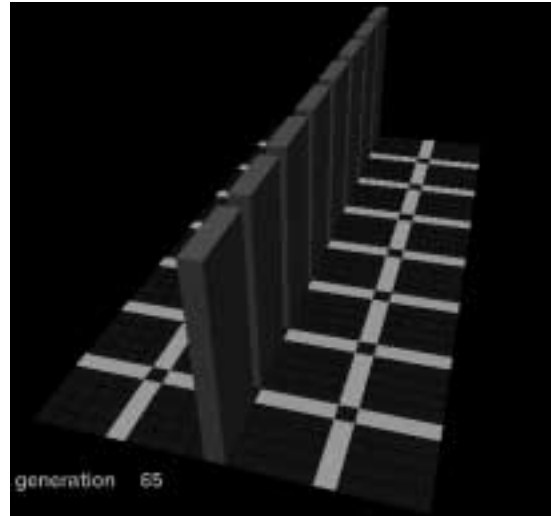
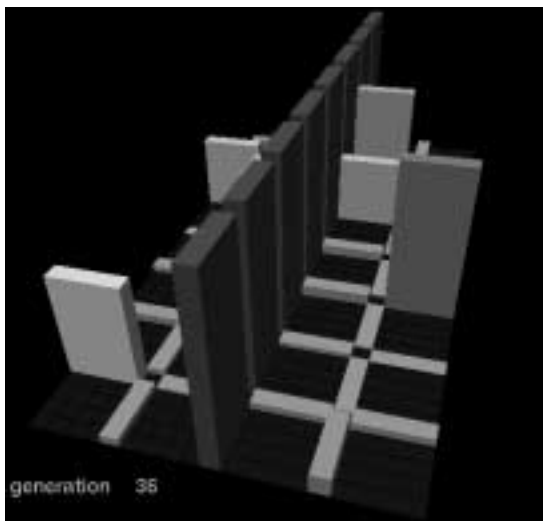
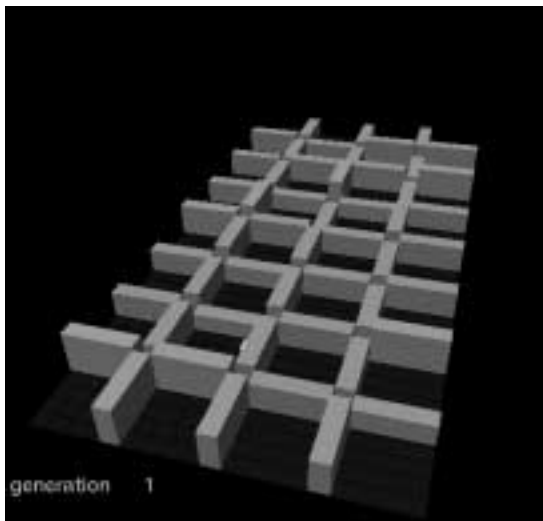


Figure 7. Performance of GA for the fracture network identification problem.

4.2. Parallel Performance

Here we present some performance results related to the second GA application where we use a workstation cluster to do the computations. The times for the various processes on the virtual machine are shown in Figure 8, for simulations using 64 individuals, 3 generations, and up to 6 SGI 100 MHz R4600 processors.

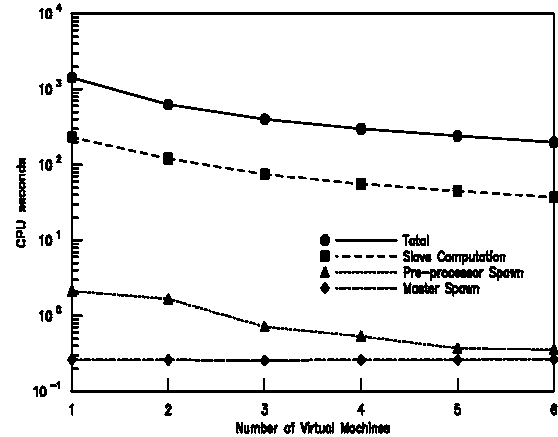


Figure 8. Virtual machine CPU times for 3 generations of GA fracture application. Each generation contains 64 individuals.

The virtual machine essentially achieves super-scalability. For example, adding one additional processor to the existing one-processor virtual machine results in a speed up of 2.29; adding two additional processors (a total of 4) results in a speed up of 4.75. Because the total number of SGA individuals are the same regardless of the number of processors in the virtual machine, the total spawning times in the master process are the same for all cases. However, the spawning times in PREFRAC decrease with the number of

[†]To appear in *High Performance Computing 1999* (Editor: A. M. Tentner), *Proceedings of the 1999 Simulation Multiconference, San Diego, CA, Apr 11–15, 1999.*

processors, caused by the decreasing number of FRAC-TRAN simulations on each processor. Because the same number of individuals are distributed to an increasing number of processors, the per-processor CPU times for the individual processes, e.g., FRACTRAN, also decrease. The per-processor communication time between FRACTRAN and PREFRAC also decreases similarly. Also noticeable are the times the master code spent on probing and communicating data with the slave processes. Percentage-wise they decrease slightly from 86% to 79%, indicating that the entire virtual machine operates in an almost perfectly parallel mode. This statistics corroborates the observations on speed-up and scalability earlier.

5. CONCLUSION

We have shown that genetic algorithms can be efficiently implemented in a parallel computing environment for large search space inverse problems such as subsurface source zone identification problems or fracture network characterization problems. Depending on the complexity of objective function evaluation, a purely task parallel implementation on a work station cluster or a combination of task and data parallel implementation on a massively parallel architecture can be adapted.

6. REFERENCES

- Giest A., Beguelin A., Dongarra J., Jiang W., Manchek B., and Sundaram, V. (1994). *PVM: Parallel Virtual Machine - A Users Guide and Tutorial for Network Parallel Computing*, The MIT Press, Cambridge, MA.
- Goldberg, 1989. "Genetic Algorithms in Search, Optimization, and Machine Learning", *Addison-Wesley*, 412 pp.
- Gropp, W., Lusk W., and Skjellum A., (1994). *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, The MIT Press, Cambridge, MA.
- Mahinthakumar G., Saied F., and Valocchi A.J., (1997). "Comparison of some parallel krylov solvers for large scale contaminant transport simulations", *High Performance Computing 1997* (Editor: A. M. Tentner), Proceedings of the 1997 SCS Simulation Multiconference, p. 134-139, Atlanta, GA, Apr 6-10, 1997.
- Mahinthakumar G., and Saied F. (1999). "Implementation and Performance Analysis of a Parallel Multicompon-

ent Groundwater Transport Code", *Proceedings of the 1999 SIAM Parallel Processing Meeting*, San Antonio, TX.

- Mahinthakumar G., Gwo J.P., and Moline G.R.. (1999). "Subsurface biological activity zone detection using genetic algorithms", accepted for publication the *ASCE Journal of Environmental Engineering*.
- Saied F., and Mahinthakumar G. (1998). "Efficient Parallel Multigrid Based Solvers for Large Scale Groundwater Flow Simulations", *Computers Math. Applic.*, Vol. 35, No. 7, p. 45-54, 1998.
- Sudicky E. A., and McLaren R.G., (1992). "The Laplace transform Galerkin technique for large-scale simulation of mass transport in discretely fractured porous formations", *Water Resour. Res.*, 28, 499-512, 1992.

ACKNOWLEDGEMENTS

This work was sponsored by the Center for Computational Sciences of the Oak Ridge National Laboratory managed by Lockheed Martin Energy Research Corporation for the U.S. Department of Energy under contract number DE-AC05-96OR22464. The authors gratefully acknowledge the use of High Performance Computing Facilities at the Center for Computational Sciences of Oak Ridge National Laboratory.

BIOGRAPHIES

G. (Kumar) Mahinthakumar is a staff research scientist at the Center for Computational Sciences of the Oak Ridge National Laboratory. He received his Ph.D. in Civil Engineering from the University of Illinois at Urbana-Champaign in 1995. His research interests are parallel and distributed computing, subsurface flow and transport modeling, and sparse matrix solvers.

J. (Jack) P. Gwo is a staff research scientist at the Center for Computational Sciences of the Oak Ridge National Laboratory. He received his Ph.D. in Civil Engineering from Pennsylvania State University in 1992. His research interests are subsurface flow and transport modeling, parallel computing, and risk based decision making frameworks.

[†]To appear in *High Performance Computing 1999* (Editor: A. M. Tentner), *Proceedings of the 1999 Simulation Multiconference*, San Diego, CA, Apr 11-15, 1999.