

A multilevel parallelization scheme based on MPI communicators for solving groundwater inverse problems

Mohamed Sayeed and Kumar G. Mahinthakumar¹

Campus Box 7908

Dept. of Civil Engineering

North Carolina State University

Raleigh, NC 27695

Phone: 919-515-7696

Fax: 919-515-7908

gmkumar@eos.ncsu.edu

Keywords: Inverse problems, groundwater transport, MPI

Abstract

Inverse problems that are constrained by large-scale partial differential equation (PDE) systems demand very large computational resources. Solutions to these problems generally require the solution of a large number of complex PDE systems. Three dimensional groundwater inverse problems fall under this category. We describe the application of a Genetic Algorithm (GA) based approach for identifying groundwater source zones using parallel computing. The GA optimizer is employed to drive a finite-element (FEM) groundwater forward transport simulator. Parallelism is exploited within the transport simulator (data parallelism) as well as the GA optimizer (task parallelism) through the exclusive use of the MPI (Message Passing Interface) communication library. Data parallelism within the transport simulator is achieved through a domain decomposition strategy. Task parallelism for performing the large number of transport simulations within each GA generation is achieved through a self-scheduling algorithm. The self-scheduling algorithm employs dynamic task scheduling to improve load balancing thus making the application attractive for a heterogeneous computational grid network. Performance results for this application are presented for GA convergence, load balancing, and overall timing on an IBM SP supercomputer.

Introduction

The knowledge of biological activity zones (BAZ) in the subsurface can be used for evaluating feasibility and remediation efficacy of aquifers and other underground water bodies. Biostimulants such as methane, dissolved oxygen, nitrates, and other nutrients are generally injected into the subsurface to enhance the

growth of bacteria that are active in degrading contaminants. Biostimulants can also be used to identify zones of distinct properties such as BAZ, Dense Non Aqueous Phase Liquids (DNAPL) or contaminant sources. In these cases, the biostimulants act as tracer signals and by measuring the biostimulant concentrations at specified locations the zones of distinct properties that perturbed these signals can be back calculated. This is an inverse problem. Regardless of the method used, however, solution of inverse problems can be several orders of magnitude more *computationally challenging* than solution of the corresponding forward problem (i.e., prediction) since a large number of solutions of the forward model are generally required. In our case, the forward model is a three-dimensional coupled partial differential equation that describes multi-component groundwater transport.

As stated in the abstract, a client-server (or manager-worker) approach is used, where the client (a single processor of a supercomputer or a workstation) performs the GA computations and the servers (multiple processors of a supercomputer) do the forward transport computations. The communication between the client and server modules is conveniently implemented using the MPI library (Gropp et al. 1999). This paper focuses on computational aspects relating to GA performance and parallel algorithm performance. GA convergence performance is examined for different encoding and crossover strategies, and effect of population size. The parallel performance is examined for an IBM SP supercomputer with regard to overall performance, communication efficiency, and load balancing.

Overview of Genetic Algorithms

GAs optimize using a search process that emulates natural evolution. In a GA, a potential solution

¹ Corresponding author.

to a problem is represented as a vector (chromosome string). The values in this vector are analogous to the genes in an individual's DNA. The GA starts with a set of potential solutions, or population. The performance of each solution is characterized by a fitness value. In the context of inverse problems, fitness is calculated using a forward solve and is inversely proportional to the difference between computed and observed values. During the GA search process, the population is iteratively subjected to several probabilistic operators that are analogous to natural selection, mating (including genetic recombination), and mutation. Each iteration through these steps constitutes a generation. Because fitter solutions are more likely to be selected for mating, the incidence of good traits of a solution tends to increase from one generation to the next. Crossover serves to sample these traits in many different combinations. Typically, the quality of solutions improves quickly at the onset of the algorithm. As the population converges, improvements diminish. Often, a stopping criterion is used to determine when improvements are sufficiently small that additional computations are not warranted. These concepts are well described in many texts, including Goldberg (1989), Davis (1991), and Michalewicz (1996).

One of the drawbacks of GA is that it is computationally intensive if the fitness evaluation (or objective function) is expensive. In this application, the objective function to be minimized is the root square error (RSE) between the observed and the computed output concentration signals at a few selected points in the domain. In order to compute the output signals for each individual in a GA operation, a forward transport simulation is performed. Because the time intensive fitness calculation for each individual in a generation can proceed independently, GAs are amenable for use in a parallel or distributed computing environment.

Genetic Algorithm Implementation

A slightly enhanced version of the Simple Genetic Algorithm (SGA) presented in Goldberg, 1989 is used in this application. The primary modifications are elitism (always retain the best solution in the new population), additional selection procedures (tournament selection in addition to the original roulette wheel selection), additional crossover strategies (uniform and multi-point crossovers in addition to the original simple crossover), and support for both binary and integer encodings. An adaptive mutation operator is implemented so that the mutation probability can be progressively reduced once one of the individuals finds the correct solution (we note here that this can only be performed if

the correct solution is known a priori as in our case!). A *restart* option is implemented so that the simulation can be restarted from the last completed generation. In the simulations performed the following steps are involved:

1. Encode the unknown zone locations as binary or integer strings.
2. Generate an initial random ensemble of strings (with a user-defined bias) equal to the number of individuals in a population or population size.
3. Perform transport simulation for each individual by decoding the strings into zone locations.
4. Compute RSE 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 a smaller RSE) using an appropriate selection strategy and mate the strings randomly (using an appropriate crossover strategy – single point, uniform, multiple point) to produce the next generation for individuals.
6. Repeat steps 3 – 5 until convergence or up to prescribed maximal number of generations.
7. If convergence is not achieved within the prescribed number of generations, then either the zone locations for the best-performing individual of all the generations or the probability distribution of the entire population at the end of simulation can be chosen as the optimal solution.

Simple and uniform crossover warrants additional explanation since they are compared in a later section. In simple crossover, two chromosome strings selected for mating are spliced at a randomly selected single point and a segment (usually the tail segment) is exchanged. In uniform crossover, randomly selected locations in the strings are swapped.

Description of the FEM transport simulator

The parallel transport simulator employed in the GA function evaluations solves the multi-component groundwater transport problem. The general system of equations describing transport of nc dissolved components undergoing reactions in saturated media is defined by

$$\frac{\partial C_i}{\partial t} = \nabla \cdot (\mathbf{D} \cdot \nabla C_i) - \nabla \cdot (C_i \mathbf{v}) + \frac{q}{\theta} (C_i - C_{0i}) + R_i$$

$$i = 1, 2, 3, \dots, nc$$

where \mathbf{v} is the 3x1 velocity field vector, \mathbf{D} is the 3x3 dispersion tensor dependent on \mathbf{v} , and C_i is the dissolved concentration of component i . The term $q(C_i - C_{0i})/\theta$

represents the source term with volumetric flux q , medium porosity θ , and injected concentration C_{0i} (e.g. from injection wells). R_i is the rate of mass loss of component i due to sorption and bioremediation reactions and is the main coupling term for the system of equations. The term, R_i , may contain many terms and can be nonlinear. For example, if only bioremediation reactions are present then R_i is given by

$$R_i = \mu_{\max} F_i X \prod_{\substack{j=1 \\ f_j \neq 0}}^{j=nc} f_{ji} \left(\frac{C_j}{K_j + C_j} \right) \quad i = 1, 2, 3, \dots, nc$$

Where F_i is the stoichiometric ratio, X is the biomass concentration, μ_{\max} is the maximum utilization rate, and f_{ji} is a factor controlling component j 's contribution to component i 's biodegradation process. If $f_{ji} = 0$ then component j does not participate in component i 's biodegradation process.

The system of transport and reaction equations is discretized using the Galerkin finite element method (FEM) with 8-node linear hexahedral elements. A logically rectangular grid structure is assumed but irregular geometries are supported using distorted elements. A Crank-Nicolson approximation (central finite-difference) is used for the time derivative terms. A lumped mass formulation (Huyakorn and Pinder 1983) is used for all time-derivative and non-derivative (zeroth spatial derivative) terms. The coupled non-linear system is solved using a modified form of the Sequential Iterative Algorithm (SIA). Several Krylov subspace iterative solvers are implemented in the code for the matrix solution (Mahinthakumar et al. 1997). Simulations in this paper are conducted using the BiCGSTAB solver, which performs reasonably well for most problems.

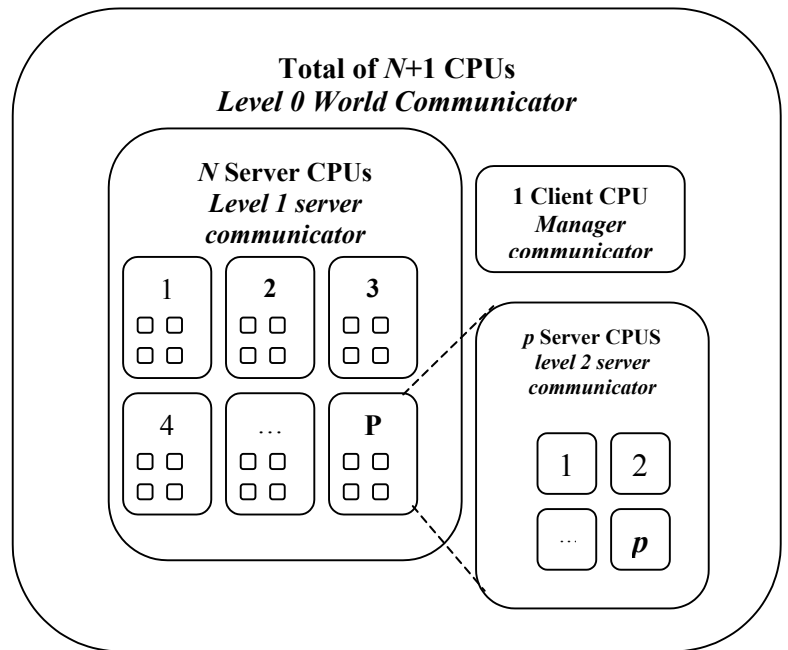
This transport simulator is parallelized using a two-dimensional domain decomposition (in the x and y directions) using explicit message passing (MPI library) to exchange information between these domains. The simulator has been tested extensively for scalability and performance on a variety of parallel architectures. Details can be found elsewhere (Mahinthakumar and Saied 1999, Mahinthakumar and Saied 2002).

GA-FEM coupled implementation

In this implementation, the FEM transport and GA modules are combined in to a single module. This is in contrast to a previous implementation (Mahinthakumar and Gwo 1999), which used the PVM library (Parallel Virtual Machine, Geist et al. 1994) for spawning tasks to perform forward transport simulation. An important

feature of the present implementation is the exclusive use of the MPI library. Use of MPI provides improved portability to a wide range of parallel architectures.

The use of “communicators” in MPI provides a convenient way to couple the GA and FEM modules through multi-level parallelism. In MPI, communicators can be assigned to any group of processes. Communicator serves as a handle to that group. Within each group, each processor has its own local process id (also called “rank”). Ranks range from 0 to n-1 within each group, where n is the total number of processes in the group. By default, all processes are assigned to the “world group” and the group handle is the “MPI world communicator”. Any number of subgroups can be created from the world group, and more groups can be created from each subgroup. Once a subgroup is created, each processor will have a local process id and all local communication within that group can be handled using the “group communicator”. By hierarchically creating subgroups we can elegantly manage multi-level communication. More details on the use of MPI groups and communicators can be found in any MPI book (e.g., Gropp et al. 1999).



$N+1$ = Total number of processes (or CPUs)
 P = Number of server subgroups
 p = number of server CPUs per subgroup (N/P)

Figure 1. Multilevel communication using MPI.

The computationally trivial GA operations are conducted on a single processor that serves as the manager or client processor. Note that in our discussion the words “processes” and “processors” are interchangeable since we always associate 1 process with 1 processor (or CPU). The objective function evaluation (forward transport simulation) for each individual of a population can be assigned to a single process or to multiple processes of a server subgroup. When a single process is used then each subgroup has just one process and the number of server subgroups equals one less than the total number of processors. When multiple processes are used, the number of server subgroups equals the total number of server processes divided by the number of processes per group. The processes and groups corresponding to our implementation are schematically shown in figure 1.

The communication is carried out at three levels between the client process and server processes. Let's assume we have a total of $N+1$ processors for our coupled GA-FEM simulation. At first, all the $N+1$ processes are first assigned to the world group with the default communicator `MPI_COMM_WORLD` (level-0). The processes are then divided into the client (1 process) and the server (N processes). The N processes form the level-1 server group. The server processes are then further divided into P groups with each group having p processes ($P = N/p$). Each of these p groups are assigned a server subgroup communicator (level-2). Each server subgroup (p processes) will perform one transport simulation at a time. In each level, local process id numbers (or ranks) will be assigned to each process. Since the basic input file is the same for all the server subgroups performing the transport simulations, only one process (in our case, the process with rank 0) at level-1 will need to read the input file. Once read, the input data can be broadcast to all the other server processes using the level-1 group communicator. This mechanism avoids the need for each server process to read the input file independently and thus saving on costly I/O time. All local communication within each transport simulation is handled using the level-2 communicator.

The manager process first sends the binary or integer string representing the unknown BAZ locations (or chromosome string) to the server processes. The server groups complete the transport simulation and return the RSE value. In each group, only the processor with rank 0 or “group leader” communicates with the manager processor. When multiple processes are used in the server subgroups to do the forward transport run, the process with local server subgroup rank 0 will receive the individual (chromosome string) from the manager and

does a broadcast within its subgroup. The subgroup processes will do the transport run and the process with local rank 0 in the subgroup will communicate back the RSE value to the manager using the level 0 global communicator.

Initially, all the server processes in the server subgroups with local rank zero will receive an individual (chromosome string) from the client, after which the individuals are assigned dynamically to whichever server subgroup returns the RSE value first to the client. The client process keeps sending the individuals in a population to server subgroup processes until all individuals in a population of a generation is completed. However, the client process needs to synchronize the processors at the end of each generation. The dynamic dispatching of the individuals by the client process will help keep all the processes busy in a homogeneous or heterogeneous grid-computing environment, where the processors with different speeds and architectures are used. The dynamic task scheduler helps achieve load balancing to a large extent if we have small number of processors and a large population size. Typically we use a population size of 128 or 256 and 65 or 129 processors on IBM SP, with one process (client) dedicated to do the GA computations. The load balance results obtained are presented in a later section.

Zone identification test Problems

Identifying the location of biological activity zones (BAZ) is critical to the efficacy of bioremediation measures. Bio-stimulants (methane, dissolved oxygen) are commonly injected into the subsurface to stimulate the growth of bacteria so that they can eventually degrade the contaminant to desired levels (Semprini and McCarty 1991). The GA-FEM framework developed above is used

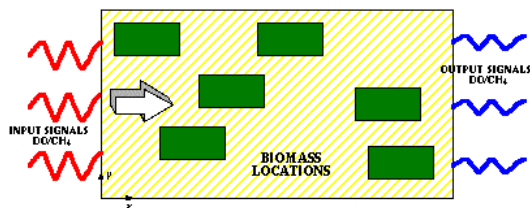


Figure 3. Problem setup for the biological activity zone identification problem

to solve the inverse problem of determining possible biological activity zones (BAZ) from the results of a biostimulation experiment. In these experiments indigenous methanotropic bacteria are stimulated by continuous periodic injection of dissolved oxygen and

methane (see Figure 3). The observed breakthroughs of methane are used to deduce possible biological activity zones in the subsurface.

We chose a moderate size problem with a grid resolution of 51 x 31 x 11 (18,904 finite element nodes). The grid spacing is fixed at 0.2 m in each direction leading to a problem domain of 10m x 6m x 2m. Stimulated methane concentrations are observed at 9 downstream locations for a time period of 40 days (200 time steps). The observed methane concentrations are pre-calculated using an assumed zonal distribution. GA will attempt to find this distribution by minimizing the error between these pre-calculated values (observed or reference signals) and the computed values for each trial solution. Two types of zone identification problems are examined: three zone identification and ten zone identification. These are described later.

GA Encoding scheme

The 10 x 6 x 2 m domain is divided into 36 rectangular zones (4 x 3 x 3 decomposition) numbered in the two different encoding schemes as shown in Figure 2. The length of binary/integer bit string for GA encoding is 36, one for each of the 36 zones. For the binary strings, the bits are either 0 or 1, and for integer strings, the bits are 0, 1, 2 or 3. If the bit is 0 for some zone location, then that zone is inactive and if it is not 0, then the zone is

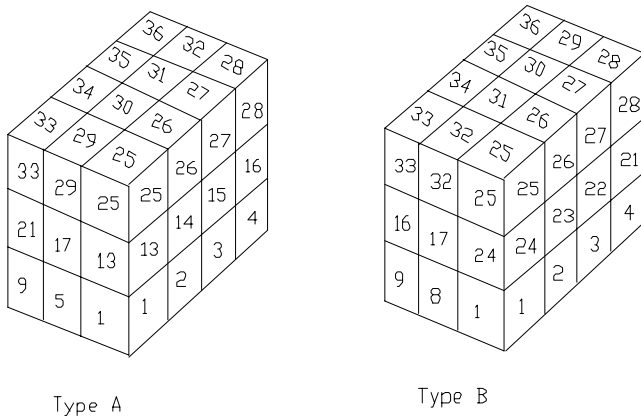


Figure 2. Two types of zone encoding

active. For integer encoding, the values denote an activity level of the zone; i.e., 0 – inactive, 1 – low, 2 – medium, and 3- high. These values are decoded into appropriate BAZ concentrations in the transport code. The locations of the bits in the chromosome string are encoded to correspond to the actual zone locations. As noted earlier, for integer encoding, each string not only encodes the zone locations but also the corresponding biological activity level. We have studied two different encoding

schemes as shown in figure 2. In encoding type A, a zone at $x = 1, y = 1, z = 2$ would correspond to a zone number 13, whereas in encoding type B a zone at $x = 1, y = 1, z = 2$ would correspond to zone number 24. One would expect encoding Type B to perform better with single point crossover. This is because, in encoding Type B, adjacent locations in the string correspond to adjacent locations in the real domain.

GA performance results

The performance of GA for several different zone identification problems are presented here. GA performance is measured in terms of convergence to the exact solution. For both three zone and ten zone cases, the initial population is generated using a probability of 0.5 whether a location is active or not. Note that in real life, we may not actually know how many zones are active. If we had prior information regarding the number of zones that are active then we would have used a probability of 0.1 (3/36) for the three zone case and a probability of 0.3 (10/36) for the ten zone case. Thus an unbiased initial population generation is more realistic. Several random seeds were tried out for generating the initial population. Convergence rate varied slightly for different random seeds and the results corresponding to the median performing seed are reported below. GA performance results presented below for both problems using both encoding types (A and B) and crossover strategies (simple and uniform). In all cases, a crossover probability of 0.4 and an initial mutation probability of 0.01 are used.

Three zone problem

For the reference case three arbitrary zones were chosen with numbers, 17, 20, and 34. For encoding type A, these numbers would correspond to (x,y,z) coordinates of (1,2,2), (4,2,2), and (2,3,3) respectively and for encoding type B, these numbers would correspond to (2,2,2), (4,2,2), and (2,3,3). A fixed population size of 256 and a fixed random seed is used for all cases. The GA convergence plots are shown in figures 4 and 5 for both encoding schemes using uniform and simple crossover respectively.

The figures show the average RSE values for the population after each generation. The results show that for both simple and uniform crossover, encoding type B performed better. Encoding B's advantage, however, is more pronounced for simple crossover. This result is expected as discussed earlier.

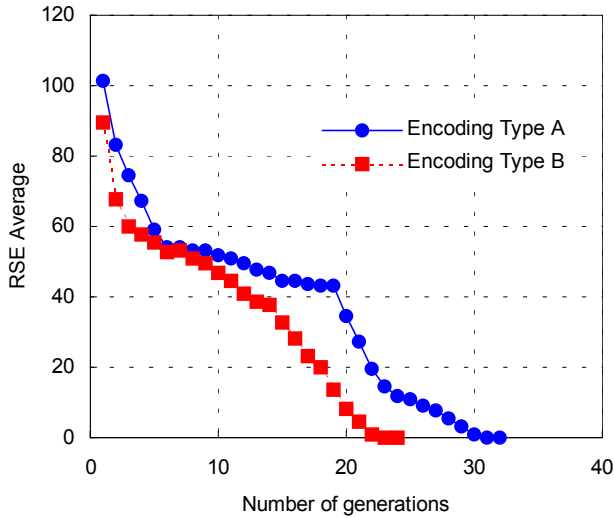


Figure 4. GA convergence history for 3-zone problem using uniform crossover

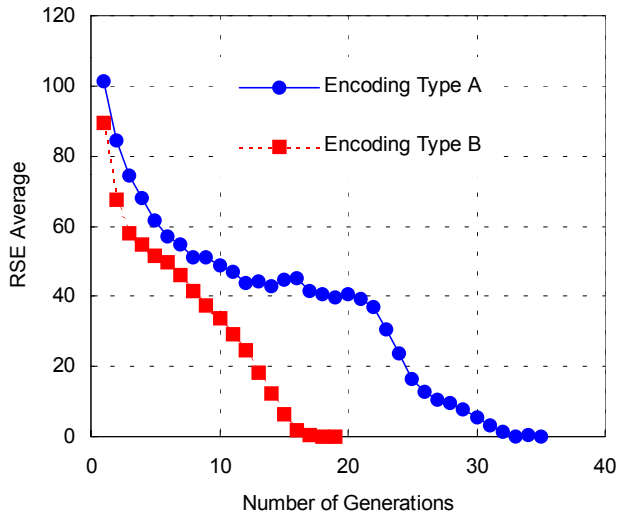


Figure 5. GA convergence history for 3-zone problem using simple crossover

Ten zone problem

This is a more difficult problem than the three zones identification problem for the GA to solve, as it is trying to identify ten active sites out of a possible 36. Doing elementary combinatorial analysis (Spiegel 1975) shows that there are ${}_{36}P_{10,26} = 36!/(10! \cdot 26!) = 2.5 \times 10^8$ possibilities! The arbitrarily chosen BAZ in the reference case are zone numbers 1, 2, 5, 7, 10, 14, 18, 19, 25, and 32. The figures 6 and 7 show the performance of GA for this problem using simple and uniform crossover

respectively. The same GA parameters as in the three zones case are used.

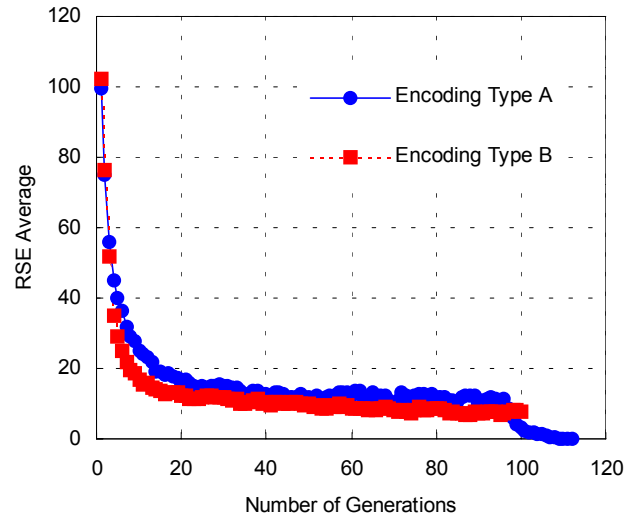


Figure 6. GA convergence history for 10-zone problem using uniform crossover

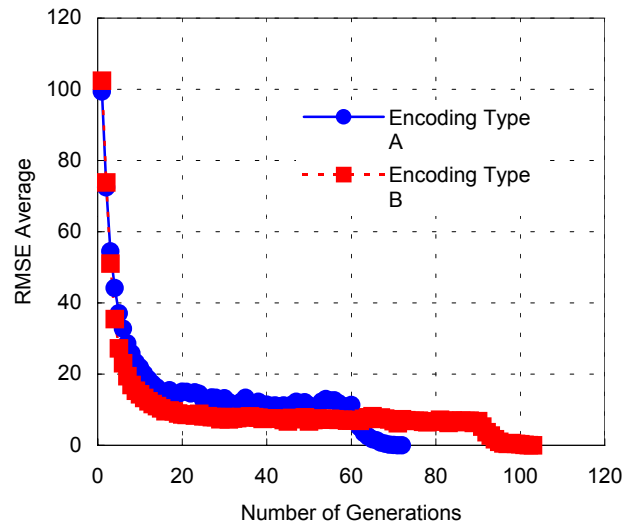


Figure 7. GA convergence history for 10-zone problem using simple crossover

Although it is not entirely clear as to which encoding scheme or crossover strategy is better for this problem, from the initial convergence behavior we see that encoding type B and simple crossover perform better. We note here that the convergence of encoding type A in figure 7 around the 70th generation seems to be a random

behavior (luck!). Obviously, the GA convergence is much better for the 3 zone identification problem than this problem (compare figures 4 and 5 with 6 and 7).

Integer encoding problem

As noted earlier, in this problem we are trying to determine the locations as well as the activity level of each zone. The convergence results are shown in Figure 8 for both the 3 zone and 10 zone problems. Obviously, this is a much more difficult problem (especially the 10 zone case) than the previous test cases, and thus we see that the 10 zone case does not find the exact solution. Further examination of the solution found by GA indicated that it had correctly identified 9 out of 10 zones in this case. Given that GA is a global search technique, this is very good! We can always employ a local search technique further fine-tune this solution to move towards the exact solution.

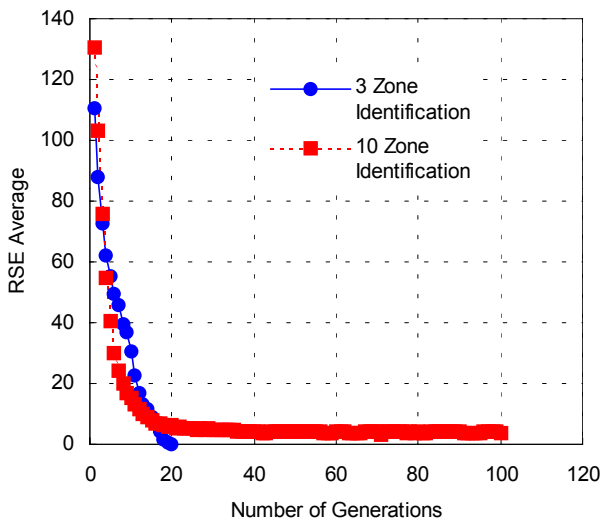


Figure 8. GA convergence history for the integer encoding problem. Encoding type B and simple crossover are used.

Parallel performance

Parallel performance of our GA-FEM implementation is examined in terms of load balancing and overall timing. The IBM SP supercomputers at Oak Ridge National Laboratory (ORNL) and the North Carolina Supercomputing Center (NCSC) were employed. Both supercomputers are identical each with 184 375 MHz Winterhawk II nodes. Results reported below correspond to the NCSC machine although simulations were performed at both sites.

Load balance

The load balance study is artificially simulated by varying the number of time steps of the transport simulation for each group of processes, and keeping a count on the number of individuals evaluated by each group. For example, the groups performing 200 time steps can be thought of as using processors that are 4 times slower than the groups performing 50 time steps. The results are shown in figure 9. The results show a fairly good load balancing achieved by dynamic scheduling of the individuals of the population.

Timing

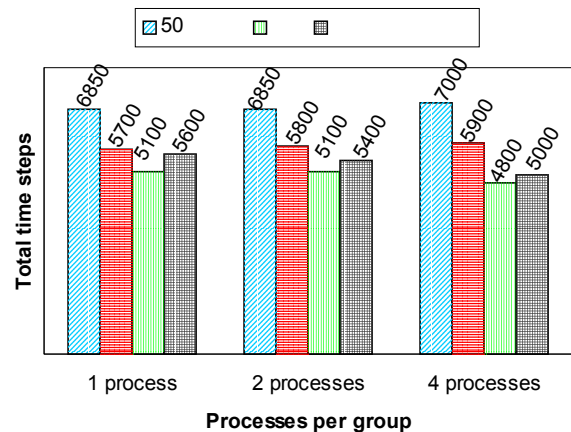


Figure 9. Load balancing efficiency of the self-scheduling algorithm. Processor speed is artificially simulated by varying the number of time steps.

It takes about 130 seconds to evaluate one individual on IBM SP using one processor with a time step size of 0.2 and 200 time steps. Using 129 processors we can perform 128 transport simulations for an entire generation in one sweep in about 150 seconds. The transport code takes about 120 seconds and the remaining time is spent in GA code operations, which takes less than 10 seconds for most of the simulations. A 50 generations simulation with population size of 128 takes about 2 1/2 hours on 65 processors of IBM SP.

Figure 10 shows the time taken to evaluate 128 individuals on 129 processors of IBM SP using 1, 2, 4 and 8 processors per individual for ten generations. Interestingly, the best case scenario corresponds to the case where we use two processors per transport simulation. One would expect the 1 processor case to outperform because the communication operations within the transport code are more intensive. This is probably because in the two-processor case, cache benefits (data

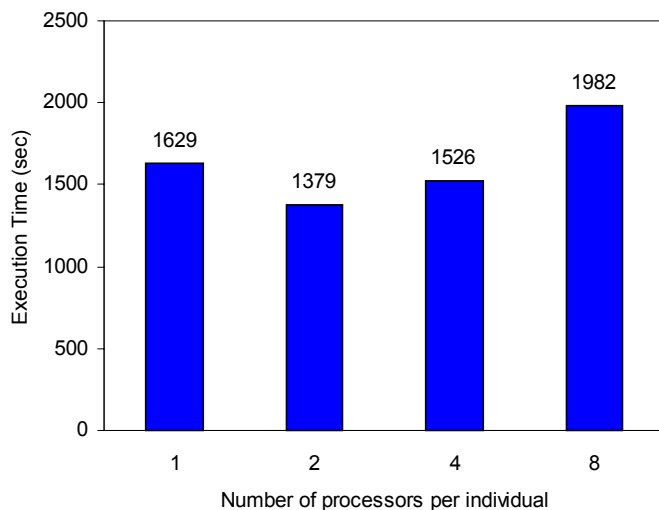


Figure 10. Total time taken by 129 processors to complete 10 generations using 1, 2, 4, and 8 processors per individual on the IBM SP. Population size is 128.

fits in cache if we solve a smaller problem per processor) outweigh the communication cost. As we increase the processors per transport simulation this benefit diminishes due increasing communication cost. This result underscores advantage of parallelizing the transport module especially on modern cache based architectures.

Conclusions

We have shown a three level parallelization strategy for coupling GA with a parallel FEM simulator for solving the ground water zone identification inverse problems. With the use of MPI group communicators and a self-scheduling algorithm, a portable and efficient parallel implementation has been achieved. With regard to GA performance, simple crossover with encoding type B generally produced better results than uniform crossover and encoding type A. In most cases, GA was able to find the exact solution. In the only case that it could not find the solution, considerable reduction in the error is achieved. A local search technique such as Nelder-Meade simplex or Powell's conjugate gradient method (Press et al. 1996) can always be applied to further fine-tune the GA solutions. We also note here that the computations performed in this work would not have been possible without parallel computing. Well over hundred thousand time-dependent FEM transport simulations were performed in these computations and serial computation on even a high-powered PC would have required months of simulation time.

Acknowledgements

The authors wish to acknowledge North Carolina Supercomputing Center and Oak Ridge National

Laboratory for providing the supercomputer resources necessary for this work.

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.
- Gropp, W., Lusk W., and Skjellum A., (1999). **Using MPI: Portable Parallel Programming with the Message-Passing Interface**, 2nd edition, The MIT Press, Cambridge, MA.
- Mahinthakumar G., and Gwo, J.P., (1999), **Task parallel and data parallel computing for subsurface inverse characterization problems**, *High Performance Computing 1999*, (Editor: A. M. Tentner), *Society for computer simulation international*, p. 217-223.
- Mahinthakumar G., and Saied F. (1999). **Implementation and Performance Analysis of a Parallel Multicomponent Groundwater Transport Code**, *CD-ROM Proceedings of the 1999 SIAM Parallel Processing Meeting, San Antonio, TX, March 1999*.
- Mahinthakumar G., Gwo J.P., Moline G.R., Webb O. F., (1999). **Subsurface biological activity zone detection using genetic search algorithms**, *ASCE Journal of Environmental Engineering*, vol. 125, no. 12, p 1103-1112, December 1999.
- Mahinthakumar, G., and F. Saied (2002). **A hybrid MPI-OpenMP implementation of an implicit finite element code on parallel architectures**, *In Press: Int. J. of High Performance Computing with Applications*.
- Mahinthakumar, G., F. Saied, and A. J. Valocchi, **Comparison of some parallel krylov solvers for large scale contaminant transport simulations**, *High Performance Computing 1997* (Editor: A. M. Tentner), *Society for Computer Simulation International*, p. 134-139, 1997.
- Marco, N., Lanteri, S. (2000). **A two-level parallelization strategy for Genetic Algorithms applied to optimum shape design.** *Parallel Computing*, 26 (2000) 377-397.
- Press, W.H., Teukolsky, S.A., Vetterling, W.T., and Flannery, B.P. **Numerical recipes in C, Second edition.** Cambridge University Press, New York, NY, 1996.
- Semprini L., and P.L. McCarty, 1991. **Comparison between model simulations and field results for in-situ bioremediation of chlorinated aliphatics: Part 1. Biostimulation of methanotropic bacteria.** *Ground Water*, 29(3), 365-374.