ELSEVIER
Available online at www.sciencedirect.com
ScienceDirect
Procedia Computer Science 1 (2012) 7–16
Procedia Computer Science
www.elsevier.com/locate/procedia

International Conference on Computational Science, ICCS 2010

Evaluation of a distributed numerical simulation optimization approach applied to aquifer remediation

Patrícia A.P. Costa[*], Eduardo L.M. Garcia, Bruno Schulze, Helio J.C. Barbosa

National Laboratory for Scientific Computing (LNCC), Av. Getúlio Vargas, 333, Quitandinha, Petrópolis, RJ, Brazil

Abstract

In this paper we evaluate a distributed approach which uses numerical simulation and optimization techniques to automatically find remediation solutions to a hypothetical contaminated aquifer. The repeated execution of the numerical simulation model of the aquifer through the optimization cycles tends to be computationally expensive. To overcome this drawback, the numerical simulations are executed in parallel using a network of heterogeneous workstations. Performance metrics for heterogeneous environments are not trivial; a new way of calculating speedup and efficiency for Bag-of-Tasks (BoT) applications is proposed. The performance of the parallel approach is evaluated.

Keywords: simulation optimization, performance evaluation, distributed computing, aquifer remediation

1. Introduction

Recently, there has been a significant increase in the capacity as well as in the availability of computing processors. Their processing power is made available through clusters, grids, and networks of workstations (NOWs), which have the potential of becoming powerful platforms for the simulation of problems that demand high computational power. Some classes of applications are more appropriate to run in specific types of architectures, but there are others that can benefit from all types of architectures. Examples of the latter include parameter sweep or any other kind of Bag-of-Tasks (BoT) applications. However, even in these most favorable situations, the utilization of the resources is not trivial and it becomes necessary to develop software components to make efficient use of the available resources.

In order to study a case of remediation of a contaminated aquifer through the use of simulation optimization techniques, we developed an approach that uses a network of heterogeneous workstations to perform in parallel the numerical simulations of the aquifer model. In this paper, this approach is presented and its performance evaluated.

The remediation strategy considered to cleanup the aquifer is pump-and-treat (PAT), one of the most commonly applied methods for groundwater remediation [1]; it requires the removal of contaminated groundwater using extraction wells [2]. The purpose of optimal design is usually to determine how many wells to install, where the wells should be located and what pumping rate is required from each one, while minimizing a cost function [3]. A simulation model of a hypothetical aquifer system is used to predict its response to a proposed pumping strategy, and an optimization technique automatically simulates a series of alternative pumping scenarios, selecting the best one [4].

[*]Corresponding author
Email address: pcosta@lncc.br (Patrícia A.P. Costa)

The Genetic Algorithm (GA) [5] is used as the optimization technique. The popularity of GAs lies in their ease of implementation and their ability to locate a global optimum. However, GAs are criticized for the large number of calls to the numerical simulator that are often required to locate a near optimal solution. Fortunately, GAs are embarrassingly parallel and the candidate solutions can be evaluated concurrently, in different computational nodes.

The parallel approach uses a master/worker pattern [6], where the master performs the optimization algorithm and assigns the simulations to the workers, distributed in the available nodes. Dynamic scheduling of the simulations, which has been shown to be essential for the efficiency of the system, is also a concern. Special attention is given to the performance evaluation of the system. Performance metrics should allow the application developer to identify factors affecting performance and to verify where adjustments can be made to improve it. These metrics are well established for homogeneous, dedicated environments. However, for heterogeneous environments, they are controversial and distinct ways of considering the different capacity of the processing elements composing the system can be found in the literature [7, 8, 9, 10]. Since NOWs are often non-dedicated systems, we propose the use of performance metrics to evaluate BoT applications that consider instead of the potential capacity of the system, its available capacity at the time of the application execution.

The rest of this work is structured as follows. Section 2 shows the aquifer model. In Section 3 the optimization technique and the characteristics of its implementation are described; experiments are conducted to demonstrate the use of this methodology. Section 4 is dedicated to the distributed system and to performance metrics. In Section 5, experiments that allow the analysis of the system performance are presented. Conclusions are drawn in Section 6.

2. The Aquifer Model

In order to design remediation strategies, the ability to predict future behavior of the groundwater system is required. Here, this is accomplished with a two-dimensional simulation model of groundwater flow and contaminant transport based on a mathematical representation of the physical system.

The contaminated aquifer is modeled by a system of partial differential equations that determine the velocity field and then the contaminant transport, using a miscible flow model [11]. The simulator solves this system using the Finite Element Method (FEM) and determines the quantity of contaminant removed by the extraction wells. As input data, the model requires the parameters that define the geometry of the aquifer, the porous medium and the fluid's physical properties, the initial and boundary conditions of the mathematical model, the finite element mesh used to discretize the problem, the number of installed pumping wells, their location and pumping rates. The output data consists of a list of contaminant concentration values in each well as a function of time, which allows one to determine the volume of removed contaminant and with that, the quality of the proposed solution.

3. Simulation Optimization

Simulation optimization is the process of finding the best values for some decision variables in a system where the performance is evaluated based on the output of a simulation model of this system. Over the last decades, groundwater quality control and remediation have been the focus of the optimization efforts in the subsurface literature [12, 13], in particular, the design of PAT systems, the most frequent technique considered [14, 3, 15]. Many optimization methods are available and recently there has been a significant growth of interest in using Genetic Algorithms (GAs) for water resource planning and design [16]. A GA was used in this paper and its basic concepts and implementation are described in this section.

3.1. Genetic Algorithms

GAs are a type of Evolutionary Algorithm (EA), a family of computational models inspired by the natural process of evolution. GAs perform global search exploring simultaneously many possible regions of good performance. They combine the generation of new individuals in regions of the search space not yet tested, being responsible for the **exploration** of new areas, and in the vicinity of known good solutions, performing the so called **exploitation** [5].

GAs use the population genetic metaphor. Individuals are representations of a potential solution to the problem at hand (candidate solution) and a population is a set of individuals. The underlying idea behind the technique is: given a population of individuals, the environmental pressure causes natural selection (survival of the fittest) and this causes a rise in the fitness of the population, since the less fit individuals tend to die while the most fit will survive and transmit some of its desirable traits to their offspring. Based on fitness, which defines the overall quality of the individuals, some candidates are selected to breed the next generation by applying the reproduction operators: recombination (also called crossover) and mutation. The new offspring can replace the old individuals according to some parental substitution rule. This process is iterated until a stop criteria is achieved.

In this implementation, an individual has information about the number of extraction wells, their locations and pumping rates. This information is stored in a data structure, as real numbers. The fitness of each individual is calculated through an objective function, based on [17], that has to be minimized:

$$f_{fitness} = CCE + CCTD + FCMS + VCE + VCTD + P * V_{res}, \tag{1}$$

where:

CCE	= capital costs of each extraction well ($\$400K$)
CCTD	= capital costs of treatment plant and discharge piping ($\$0.460K\ per\ m^3/day$)
FCMS	= fixed costs of management and sampling/analysis ($\$415K\ per\ year$)
VCE	= variable cost of electricity for well operations ($\$0.009K\ per\ m^3/day\ per\ year$)
VCTD	= variable cost of treatment and discharge ($\$0.064K\ per\ m^3/day\ per\ year$)
P	= penalty charged over the volume of contaminant not removed from the aquifer (V_{res}).

3.2. Aquifer Remediation

A hypothetical aquifer was used to test the ability and efficiency of the optimization methodology presented in this study. Its hydrological settings are: heterogeneous, confined, and isotropic. The simulations were performed inside a total area of 4,000 by 8,000 meters, with a mesh of 160×320 bilinear square elements, each having a size of $25,0 \times 25.0$ meters. The boundary conditions on the lower and upper sides of the domain are of constant pressure of $50m$ and $0m$ respectively. Zero-flow boundary conditions are imposed along the left and right sides. The porosity of the field was assumed to be 0.2 and its permeability, $100mD$, except on the shaded area (Figure 1), where porosity is 0.05 and permeability $0.1mD$, yielding a variable velocity field along the 8,000 meters of the domain. The initial volume of contaminant within the aquifer is 60,000 units and contaminant transport is simulated for a 5-year period. The contaminant is divided in two plumes: the one in the right side has half of the volume of contaminant than the one in the left (Figure 1(a)). To achieve an acceptable level of water quality, at the end of this period the amount of contaminant must be less or equal to 0.01 units. Figure 1 describes the physical problem: the initial condition of the aquifer is shown in Figure 1(a)), and Figures 1(b), 1(c), 1(d), 1(e), and 1(f) shows the contaminant on days 400, 700, 1100, 1400, and 1800, in a simulation done without the placement of extraction wells.

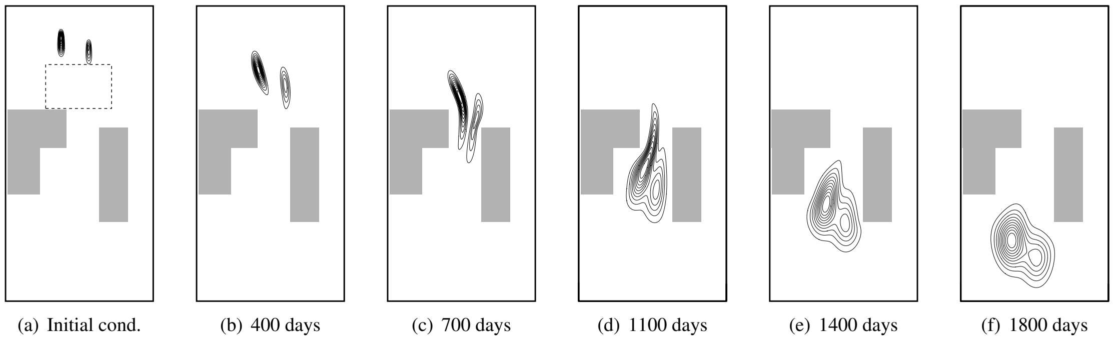
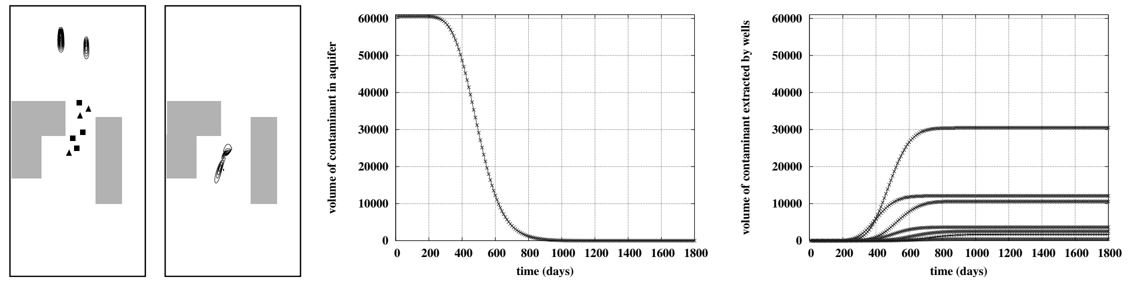

(a) Initial cond. (b) 400 days (c) 700 days (d) 1100 days (e) 1400 days (f) 1800 days

Figure 1: Hypothetical contaminated aquifer without remediation.

The pumping system allows a maximum of nine wells, each with a pumping rate chosen from a set of four possibilities: 100, 150, 200 and $250m^3/day$. Wells can not be placed within the dashed area (Figure 1(a)) and they must be at a minimum distance of 250m from each other.

In this experiment, the optimization was executed 13 times, each time using a distinct “random seed”, a number used to initialize a pseudorandom number generator, that is used in many steps of the GA, including the generation of the initial population. It is expected from a GA that, starting from different initial populations, it may locate the same set of optimal results. In each optimization, 42040 simulations were performed, divided in the following manner: 40 candidates for the initial population and 150 additional generations of 280 individuals each.

The solutions found in each of the executions achieved, with similar costs, the accepted level of water quality using seven extraction wells. Results of remediation using one of these solutions can be observed in Figure 2. Well locations and pumping rates are shown in Figure 2(a) (■ = $200m^3/day$ and ▲ = $250m^3/day$). Figure 2(b) shows the plume of contaminant after 1230 days (the amount of time needed to achieve satisfactory contaminant levels). The volume of contaminant in the aquifer during the simulation time interval is presented in Figure 2(c) while the volume of contaminant extracted by each of the remediation wells is pictured in Figure 2(d). The value of the fitness function was $\$5,292K$ ($CCE = \$2,800K$; $CCTD = 713K$; $FCMS = 1,399K$; $VCE = 47K9$; $VCTD = 333K$).



(a) A remediation solution. (b) 1230 days (c) Volume of contaminant in the aquifer (d) Volume of contaminant extracted by remediation wells

Figure 2: Results of a solution found by the GA.

4. Distributed Approach

In order to search for the optimal solution, the optimization algorithm needs to evaluate, via numerical simulations, a large number of candidate solutions. These evaluations are independent tasks - there is no communication or dependencies among them. Applications composed of independent tasks are often referred in the literature as Bag-of-Tasks (BoT). In this system, the simulations are solved concurrently, using machines available in a local network. The parallel execution guarantees a reduced response time and allows for the solution of more complex computational modeling problems.

4.1. Master/Slave Paradigm

To execute the numerical simulations in parallel, the master/slave paradigm is used. Master/slave is a simple yet widely used technique appropriate to execute independent tasks under the centralized supervision of a control processor [18, 6]. For BoT applications running on a network of heterogeneous computers, master-slave has been the most common choice so far, due mainly to simplicity of implementation, tolerance to slave failures, and simple communication topology [19]. However, it is important to note that centralized master-slave approach may affect scalability, since the existence of a single master may become a bottleneck. This problem has been studied by many authors [19, 20, 21]. The use of a hierarchical master/slave structure reduces this effect. Nevertheless, the standard master/slave scheme is effective for a moderate number of processors [22].

4.1.1. Implementation

Our approach is composed by three main modules: master, worker, and simulator. The master is a Java module that follows the steps of the optimization algorithm and is responsible for scheduling the numerical simulations to the available workers. The worker, also a Java module, is replicated in the available computational nodes. Workers call the simulator and with its output they calculate the fitness function value, returning it to the master. Master and workers communicate via Remote Method Invocation (RMI). The simulator is a Fortran sequential program that solves the mathematical model of the aquifer, using FEM.

Before the application starts, a number of nodes are chosen by the user to initialize the workers. These nodes have no prior knowledge of the application, but must have all necessary libraries installed. The nodes are assumed to be part of the local network and all the necessary files are copied via SSH.

A machine in the network is chosen to become the master. The master follows the steps of the GA and at each generation, when it has to evaluate the individuals of a given population - the computationally expensive part of the algorithm - it partitions the population in blocks of individuals and distributes the blocks to workers. The workers evaluate each individual of that block, calling the numerical simulator and then calculating the fitness of the individual based on the output of the simulation model. When a worker finishes processing a block, it returns the results to the master and receives another block, until all blocks are evaluated. At the end of each generation, the master sorts all candidate solutions, according to their fitness, and selects the ones who will survive.

A diagram depicting the implemented approach is presented in Figure 3.

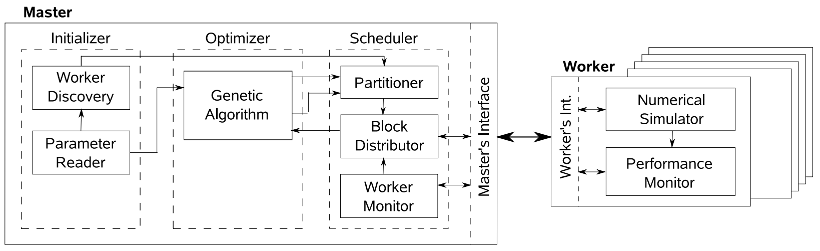


Figure 3: Implementation architecture.

4.2. *Task Scheduling*

Scheduling refers to the way tasks are assigned to run on the available processing elements (PEs). The number of tasks a PE should be assigned is an important issue in parallel computing, particularly in heterogeneous systems. In such systems, in order to obtain high performance, an application must equally distribute the load among PEs, maximizing their utilization while minimizing the total job execution time. We want to avoid that faster PEs become idle while waiting for slower ones to finish their tasks in places that require synchronization. In a generational GA that happens at the end of every generation, when the master has to wait for all of computing resources to deliver the results of their evaluations.

As seen in the literature [23, 24, 25, 26], dynamic scheduling is appropriate to achieve load balancing in heterogeneous or homogeneous non-dedicated environments.

In this implementation, the methods pure self-scheduling (PSS), guided self-scheduling (GSS), and factorial self-scheduling (FSS) were made available. However, for the experiments of this paper only PSS was used. PSS partitions the job (in this implementation, the evaluation of one population) of size N in N blocks of one task (the evaluation of each individual), and in this way guarantees balanced workloads. One drawback of this method is the amount of communications required, but here this is minimized as a local network is used and also because the communication times are very small in comparison to the computation times. In [26], experiments with GAs and the aforementioned scheduling techniques are presented.

4.3. *Performance Metrics*

In general, the primary purpose of parallelizing an application is to reduce the overall elapsed time to obtain the results. However, execution time is not usually the most convenient metric by which to evaluate parallel performance.

One of the most important reasons for measuring and evaluating parallel performance is to verify how efficiently available resources are being utilized and to discern whether actual performance improves if the parallel environment changes, such as by using more PEs. It is usually expected that, when the number of PEs in the parallel system is increased, the computation time decreases or problems of larger size can be solved. The capability of a parallel system to increase its performance with the increase in the number of PEs is called scalability. It is necessary to have a way of evaluating parallel performance, assessing whether it is improving or not, and then identifying the factors affecting performance and where adjustments can be made to improve it.

Measuring and evaluating performance of heterogeneous systems is not straightforward. In particular, conventional techniques used for homogeneous and dedicated systems are not appropriate and must be adjusted to consider the variation of the computing power of the different workstations. In this section, we look at traditional ways as well as to alternative ways of measuring and evaluating the parallel performance of a heterogeneous system.

4.3.1. Speedup and Efficiency

Common techniques for evaluating parallel performance have often been speedup and efficiency. Speedup (S) is a dimensionless metric that measures the performance gain obtained by the parallel implementation of an application, while efficiency (E) measures the fraction of time for which the PEs are usefully employed, denoting the effective utilization of computing resources. They are mathematically given by

$$S = \frac{T_1}{T_P} \qquad \text{and} \qquad E = \frac{S}{P}, \tag{1}$$

where P is the number of PEs, T_1 is the total elapsed time of an execution using one PE, and T_P is the total elapsed time using P PEs.

In an ideal parallel system using P PEs, with computation evenly decomposed into P tasks, and running on a homogeneous and dedicated environment, speedup is equal to P and efficiency is equal to one. In practice, ideal behavior is not achieved because while executing a parallel algorithm, the PEs spend some time performing tasks which are not central to the computations of the main algorithm, such as: synchronization, communication, task scheduling, and additional processing, not necessary when using only one PE. The time spent with these peripheral computations denotes its overhead.

4.3.2. Related Work

The traditional parallel metrics above are popular ones but for heterogeneous systems they are also controversial, since in a heterogeneous environment the different capacity of the PEs must be considered [7, 8, 9, 10]. Although it is easy to measure the total elapsed time for a parallel execution (T_P), we need to define which single PE should be used to calculate T_1 and evaluate speedup. Several approaches have been reported in the literature. Yero and Henriques [19] present a work where the execution time of the fastest machine in the system is used for T_1. On the other hand, according to Colombet e Desbat [7], when using different PEs, the speedup can no more be implicitly relative to the execution time on one PE, but must be expressed explicitly relative to the execution time on certain PEs, because they are no longer identical. Ramos-Hernández and Tokhi [8] define a virtual PE whose performance characteristics correspond to the average characteristics of all computers in the cluster.

Zhang and Yang [9] and Michailidis and Margaritis [27] also evaluate speedup comparing total parallel elapsed time to elapsed time on the fastest PE, but they use the concept of power weight to determine efficiency. The power weight of a PE refers to its computing power relative to the fastest PE in a system and its value is less than or equal to one. The potential power of the heterogeneous system is the sum of the power weights of all PEs in the system. Efficiency is then calculated as speedup divided by the total potential power.

Crowl [10] suggests the concept of linear speed, which uses the amount of work done per unit time instead of elapsed time. In this technique, one compares the actual amount of work done per unit time by the whole system with the potential amount of work that could be done by the system, the latter calculated as the sum of the amount of work that could be done on each PE per unit time. Post [28] proposes “linear efficiency” as an extension of linear speed as defined by Crowl. Linear efficiency is calculated as the ratio between the actual linear speed achieved by a parallel execution and the potential linear speed of the system.

Pastor and Orero [29] present a definition of efficiency as the ratio between the best response time achievable for solving a specific problem in a given system and the real response time achieved during the algorithm execution. The best response time will be obtained if the workload is evenly distributed among all of the nodes (perfectly balanced distribution) and if there is no overhead time.

4.3.3. Dynamic Speedup and Dynamic Efficiency

The techniques described above consider the potential power of the heterogeneous system. However, at the time of an application execution, some of the PEs can be subject to external loads since, in practice, a heterogeneous NOW is often a nondedicated system. Therefore, at that moment, its **available** and **potential** capacity are not the same.

We suggest, for BoT applications, a dynamic way of calculating speedup and efficiency that considers the capacity of the system at the time of application execution. It uses an approximation for T_1, called $\overline{T_1}$, calculated as the sum of the elapsed time of all useful tasks (not overheads) performed by the parallel application. In the implemented approach, $\overline{T_1}$ is calculated as the sum of the elapsed time of all the tasks performed by the workers, given by:

$$\overline{T_1} = \sum_{i=1}^{P} T_{W_i}, \tag{2}$$

where T_{W_i} is the elapsed time of the tasks performed by the Worker i on PE i. Using the same set of PEs, $\overline{T_1}$ will vary for distinct executions of the application, due to factors such as sharing of computing resources and parallel program nondeterminism. $\overline{T_1}$ considers the heterogeneity of the resources and since it is calculated at the time of application execution, it measures the system processing power available at that moment.

We adopt the adjective dynamic when $\overline{T_1}$ is used to calculate speedup and efficiency and we denote them $\overline{S}$ (dynamic speedup) and $\overline{E}$ (dynamic efficiency), given by:

$$\overline{S} = \frac{\overline{T_1}}{T_P} \quad \text{and} \quad \overline{E} = \frac{\overline{S}}{P}. \tag{3}$$

Dynamic speedup and dynamic efficiency reflect only the effects of the application overheads, not those of external loads, and therefore they are more useful to help developers to identify, within the application, factors affecting its performance, and then to verify where adjustments need to be made.

5. Performance Experiments

In section 3.2, experiments were performed to demonstrate the efficacy of this methodology. Here, they were conducted to verify the performance of the distributed approach.

The experiments were conducted in a heterogeneous environment. Machines with multicore processors were used and in some cases more than one worker was installed in one machine. The application does not automatically recognize the existence of multiple cores, but it uses a list of machines where the workers will run and, for each worker, the corresponding port used for communication with the master. The number of workers installed in each machine was never greater then the number of cores available in its processor.

The computers used have the following characteristics: Intel Core 2 (1.86 GHz/3 GB), Intel Pentium 4 (1.86 GHz/512 MB), Intel Pentium D (2.8 GHz/1 GB), Intel Core 2 Quad (2.5 GHz/8 GB), Dual Core AMD Opteron (2.6 GHz/64 GB), Intel Xeon (2.93 GHz/132 GB), Intel Itanium 2 (1.8 GHz/64 GB), Intel Pentium 4 (3.2 GHz/1.5 GB), Quad-Core AMD Opteron (2.0 GHz/4 GB), Intel Xeon (1.86 GHz/1 GB), Intel Dual Core (1.8 GHz 1 GB), and Intel Pentium D (3.2 GHz/2 GB). They are distributed in different labs at LNCC and connected via LAN. These computers are grouped under distinct NIS (Network Information System) and run Linux-based OS brands such as: openSUSE, Ubuntu, Mandriva, and Red Hat. This heterogeneity resembles somehow the one of a computational grid.

5.1. Scalability Experiments

Three experiments were performed: expA (with 10 workers), expB (with 20 workers), and expC (with 30 workers). For these experiments, a set with 10 different cores was created, one from each one of the machines described above. This set was used in expA. For expB, with 20 workers, two sets identical to the one utilized for expA were used and in the same way, on expC, with 30 workers, three of the same set were used. This setup was chosen to allow an increase in processing power proportional to the increase in the number of cores of each experiment, facilitating performance analysis. In each experiment, a total of 930 simulations were executed: 30 individuals in the initial population plus 10 generations of 90 individuals, for a total of eleven jobs.

Table 1 presents performance metrics of the three experiments. It can be seen that the total elapsed time for the application execution decreases with the use of more cores, while speedup increases. However, efficiency decreases in a way that was not expected. In section 4.3.1, possible causes of overhead and loss of efficiency in a parallel system were presented. Particularly in this application, the following are possible sources of overhead: GA tasks; scheduling of parallel evaluations, which includes: job partition in blocks of tasks, sequential accesses to block list - a critical section, distribution of blocks of tasks to the workers, management of threads; communication between master and workers; and idleness caused by load imbalance.

In this approach, the idle time resulting from the moments of synchronization (which happen at the end of every generation when the master has to wait for all of computing resources to deliver the results of their evaluations) is considered the main factor that increases overhead and consequently reduces efficiency. The use of dynamic scheduling methods minimizes the load imbalance, but since there are several moments of synchronization, 11 in this experiment, this is still a critical issue.

# of cores used	**10**	**20**	**30**
elapsed time for application parallel execution (T_P)	56min	31min	21min
total workers processing time ($\sum_{i=1}^{P} T_{W_i} = \overline{T_1}$)	535min	533min	536min
Dynamic Speedup ($\overline{S}$)	9.49	16.89	25.01
Dynamic Efficiency ($\overline{E}$)	0.95	0.84	0.83

Table 1: Main performance metrics for experiments with 10, 20, and 30 cores.

Table 2 presents complementary measurements of the three experiments. T$_{worker}$ max is the sum of processing time of the worker that took more time performing evaluations and T$_{worker}$ min is this sum for the worker that took less time performing evaluations. The amount of evaluations performed within this time is in parenthesis. The minimum and maximum amount of individuals evaluated by a worker is represented by "# of sim". The time of one evaluation performed by the fastest worker is T$_{sim}$ min and by the slowest worker, T$_{sim}$ max.

Analyzing the values shown, it can be observed that the fastest worker performed 134 evaluations in expA, but in expB, despite proportional increase of processing power, this worker was assigned less than half of this load (62 evaluations). The slowest worker evaluated 52 individuals in expA, but on expB it performed more than half of that load. The same kind of comparison can be made between expA and expC.

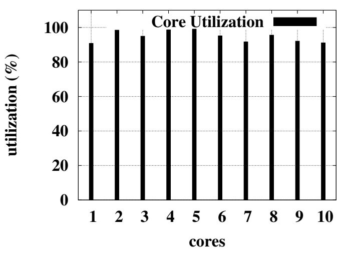
# of cores used	**10**	**20**	**30**
Total cpu time (simulations)	530min	528min	531min
T$_{worker}$ max (# of sim.)	3359.49s (103)	1875.63s (31)	1272.43s (21)
T$_{worker}$ min (# of sim.)	3077.79s (93)	1398.39s (62)	923.61s (41)
max # of sim. per worker	134	62	41
min # of sim. per worker	52	31	21
T$_{sim}$ min (# of sim.)	24.07s (134)	22.55s (62)	22.53s (41)
T$_{sim}$ max (# of sim.)	60.05s (52)	60.50s (31)	60.59s (21)
average simulation time	34.54s	34.41s	34.58s

Table 2: Complementary measurements for experiments with 10, 20, and 30 cores.

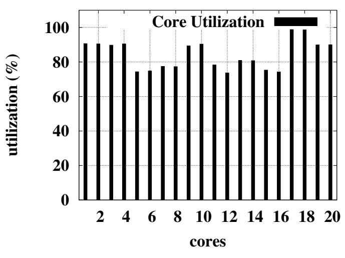
Figures 4(a), 4(b), and 4(c) show utilization of the cores in expA, expB, and expC respectively. In the first experiment, a high utilization of cores is achieved, but in the other two, idle time can be detected. It is also worth to note that in expB, where two cores of each kind were used, the utilization of each pair is similar and in expC, the same behavior can be observed for each group of three cores.

5.2. *"All in One Bag" Experiment*

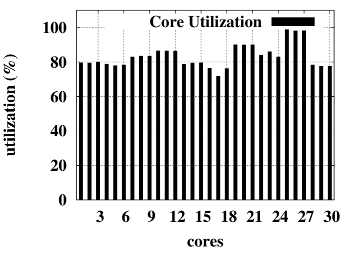
To confirm the hypothesis that in generational GAs, despite the use of dynamic scheduling, the idle time of processing elements at the end of each generation is still the determinant cause of increase in overhead and reduction



(a) expA



(b) expB



(c) expC

Figure 4: Cores utilization.

in efficiency, a new experiment was executed, named expD: it had the same characteristics of expC, except that the 930 evaluations were all done in one generation ("All in one Bag"), as the initial population, producing only one moment of synchronization, instead of eleven as in expC. The efficiency increased significantly, changing from 0.83 to 0.98, as it can be verified in Table 3, showing that the distributed system does not introduce excessive additional costs and the application, naturally parallel, is scalable.

# of cores used	**30**
elapsed time for application parallel execution (T_P)	17min
total workers processing time ($\sum_{i=1}^{P} T_{W_i} = \overline{T_1}$)	510min
Dynamic Speedup ($\overline{S}$)	29.44
Dynamic Efficiency ($\overline{E}$)	0.98

Table 3: Performance metrics for expD, with only one iteration.

In order to achieve high performance in a heterogeneous parallel system, load balance is extremely necessary. Particularly in this approach, with several moments of synchronization, the idleness of PEs at the end of each generation is the main reason for increase in overhead and consequent loss of efficiency. An alternative way to improve performance is the implementation of an intelligent scheduling mechanism that uses information from previous iterations to choose whether or not to assign more tasks to slower PEs when there are few individuals left to be evaluated.

6. Conclusions

This work shows an evaluation of a distributed simulation optimization approach used to automatically find remediation solutions to a contaminated aquifer. To optimize the remediation solution, a GA was implemented, coupled with a computational model of the aquifer used to evaluate the quality of the candidate solutions. A hypothetical aquifer was used, as it contains the fundamental attributes that allows for the analysis of the implemented approach.

To test the ability and efficacy of the implemented GA, an experiment was performed, using 13 different random seeds, and solutions that achieved, with similar costs, the accepted level of water quality with the placement of seven wells were found in all cases, showing that even starting from different initial populations, the algorithm was able to locate the same set of optimal results.

The executions of the numerical simulations required by the GA were done in parallel. The distributed approach, based in the master/worker pattern, dynamically schedules these simulations among the available nodes. The scheduling mechanism used was pure self-scheduling (PSS), which guarantees balanced workloads but requires a large amount of communication, a drawback that was minimized as a local network was used and the communication times were small in comparison to the computation times.

Traditional performance metrics were described and the use of metrics that consider the available capacity of a heterogeneous system at the time of an application execution, instead of its potential capacity, was proposed for BoT applications. These suggested metrics reflect only effects of the application overheads, not those of external loads.

Therefore, they are more useful to help developers to identify, within the application, factors affecting its performance, and then verify where adjustments need to be made. To the best of our knowledge, the proposed metrics can be used with any kind of BoT applications.

To analyze the performance of the distributed approach, experiments with 10, 20 and 30 cores were conducted and the proposed metrics were used. It was observed that the application - embarrassingly parallel - is scalable, however, due to the existence of several points of synchronization, load distribution has a critical effect upon efficiency. Despite the use of dynamic scheduling that decreases workers idle time, improvements can still be implemented. We suggest, as future work, the use of intelligent or adaptive scheduling techniques which use information from previous iterations to choose whether or not to assign more tasks to slower PEs when there are few individuals left to be evaluated.

References

[1] L.-C. Chang, H.-J. Chu, C.-T. Hsiao, Optimal planning of a dynamic pump-treat-inject groundwater remediation system, Journal of Hydrology 342 (2007) 295–304.

[2] M. C. Cunha, Groundwater cleanup: The optimization perspective (a literature review), Eng. Opt 34 (2002) 389–702.

[3] M. A. J. Guan, Optimal remediation with well locations and pumping rates selected as continuous decision variables, Journal of Hydrology 221 (1999) 20–42.

[4] A. J. Shreedhar Maskey, D. P. Solomatine, Groundwater remediation strategies using global optimization algorithms, Journal of Water Resources Planning and Management 128 (2002) 431–440.

[5] A. Eiben, J. Smith, Introduction to Evolutionary Computing, Springer-Verlag, 2003.

[6] T. G. Mattson, B. A. Sanders, B. L. Massingill, Patterns for Parallel Programming, Addison-Wesley, 2004.

[7] L. Colombet, L. Desbat, Speedup and efficiency of large-size applications on heterogeneous networks, Theoretical Computer Science 196 (1998) 31–44.

[8] D. Ramos-Hernández, M. Tokhi, Performance evaluation on heterogeneous systems, Microprocessor and Microsystems 25 (2001) 203–212.

[9] X. Zhang, Y. Yan, Modeling and characterizing parallel computing performance on heterogeneous networks of workstations, in: SPDP '95: Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing, 1995, p. 25.

[10] L. A. Crowl, How to measure, present, and compare parallel performance, IEEE Parallel Distributed Technology 2 (1) (1994) 9–25.

[11] A. Loula, E. Garcia, A. Coutinho, Miscible displacement simulation by finite elements methods in distributed memory machines, Computer methods in applied mechanics and engineering 174 (1999) 339–354.

[12] A. S. Mayer, C. Kelley, C. Miller, Optimal design for problems involving flow and transport phenomena in saturated subsurface systems, Advances in Water Resources 25 (2002) 1223–1256.

[13] I. M. Kalwij, R. C. Peralta, Simulation/optimization modeling for robust pumping strategy design, Ground Water 44 (2006) 574–582.

[14] C. Huang, A. S. Mayer, Pump-and-treat optimization using well locations and pumping rates as decision variables, Water Resources Research 33 (1997) 1001–1012.

[15] X. Ren, B. Minsker, Which groundwater remediation objective is better, a realistic one or a simple one?, Journal of water resources planning and management 131 (5) (2005) 351–361.

[16] X. Cai, D. C. McKinney, L. S. Lasdon, Solving nonlinear water management models using a combined genetic algorithm and linear programming approach, Advances in Water Resources 24 (2001) 667–676.

[17] Geotrans, Transport optimization hastings naval ammunition depot, Hastings Documentation Formulations (2002).

[18] O. Beaumont, A. Legrand, Y. Robert, The master-slave paradigm with heterogeneous processors, IEEE International Conference on Cluster Computing 0 (2001) 419–426.

[19] E. J. H. Yero, M. A. A. Henriques, Speedup and scalability analysis of master-slave applications on large heterogeneous clusters, Journal of Parallel and Distributed Computing 67 (2007) 1155–1167.

[20] F. A. Silva, H. Senger, Improving scalability of bag-of-tasks applications running on master-slave platforms, Parallel Computing 35 (2009) 57–71.

[21] A. Chronopoulos, S.P., N. Yu, Scalable loop self-scheduling schemes for heterogeneous clusters, in: Proceedings of the IEEE International Conference on Cluster Computing, 2002.

[22] J. Brest, V. Zumer, Solving asymmetric traveling salesman problems using dynamic scheduling on a heterogeneous computing system (2000).

[23] S. Penmatsa, A. Chronopoulos, N. Karonis, B. Toonen, Implementation of distributed loop scheduling schemes on the teragrid, Parallel and Distributed Processing Symposium, IEEE (2007) 1–8.

[24] C. Yang, K. Cheng, W. Shih, On development of an efficient parallel loop self-scheduling for grid computing environments, Parallel Computing 33 (2007) 467–487.

[25] K. Cheng, C. Yang, C. Lai, S. Chang, A parallel loop self-scheduling on grid computing environments, in: Proceedings of the 7th International Symposium on Parallel Architectures, Algorithms and Networks, 2004.

[26] P. A. P. Costa, F. J. Lima, E. L. M. Garcia, H. J. C. Barbosa, B. Schulze, Task scheduling schemes for simulation optimization in computational grids (in portuguese), in: Anais do VI Workshop de Computação em Grid e Aplicações, 2008, pp. 61–72.

[27] Michailidis, Margaritis, Parallel text searching application on a heterogeneous cluster of workstations, in: ICPPW '01: Proceedings of the 2001 International Conference on Parallel Processing Workshops, IEEE Computer Society, Washington, DC, USA, 2001, p. 169.

[28] E. Post, H. Goosen, Evaluating the parallel performance of a heterogeneous system, in: Proceedings of the 5th International Conference and Exhibition on High-Performance Computing in the Asia-Pacific Region, 2001.

[29] L. Pastor, J. L. B. Orero, An efficiency and scalability model for heterogeneous clusters., in: CLUSTER '01: Proceedings of the 3rd IEEE International Conference on Cluster Computing, IEEE Computer Society, Washington, DC, USA, 2001, p. 427.