Asociación Argentina
de Mecánica Computacional

Mecánica Computacional Vol XXIX, págs. 3019-3036 (artículo completo)
Eduardo Dvorkin, Marcela Goldschmit, Mario Storti (Eds.)
Buenos Aires, Argentina, 15-18 Noviembre 2010

LARGE SCALE PARALLEL SIMULATION OPTIMIZATION ON A NETWORK OF HETEROGENEOUS WORKSTATIONS

Patricia A.P. Costa, Eduardo L.M. Garcia, Bruno Schulze and Hélio J.C. Barbosa

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

{pcosta, bidu, shulze, hcbm}@lncc.br

Keywords: simulation optimization, distributed computing, aquifer remediation, genetic algorithm (GA)

Abstract. This paper analyses results from large scale parallel computations obtained from a distributed approach which uses numerical simulation and optimization techniques to automatically find remediation solutions to a hypothetical contaminated aquifer. The remediation strategy is based on withdrawal, which requires the removal of contaminated groundwater from the aquifer by pumping. The design of the remediation system involves the choice of the number of wells to be installed, their locations and pumping rates, with the goal of maximizing the amount of contaminant extracted, while minimizing the cost of the system. The optimization strategy adopted, a Genetic Algorithm (GA), requires a large number of calls to the numerical simulation model of the aquifer, which tends to be computationally expensive. To overcome this drawback, the numerical simulations are executed in parallel, using a network of heterogeneous workstations. Performance of the parallel approach, using 95 and 190 cores, is analysed.

1 INTRODUCTION

Recently, there has been a significant increase in the capacity as well as in the availability of computer 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, in particular those that make use of numerical simulation optimization, since the repeated execution of simulation models through the optimization cycles tends to be computationally expensive. Large scale parallel and distributed computing can be a way to obtain faster response, enabling a new generation of scientific and engineering applications that uses simulation-based optimization. However, the utilization of the resources is not trivial and achieving this goal requires the development of software components to make efficient use of the available resources.

In this paper we evaluate and present further development of a distributed approach used to automatically find remediation solutions to a hypothetical contaminated aquifer. This approach has been studied by the authors and was presented in (Costa et al., 2008, 2010). Here we analyse its behavior with large scale parallel experiments, using a network of heterogeneous workstations.

Groundwater contamination from leakage, spills, or dumping of toxic substances is widely regarded as one of the leading environmental problems of our time. Numerous techniques are available for remediation, including: pump-and-treat (PAT) of contaminated water, hydraulic containment of plumes, and bioremediation (Cunha, 2002). The remediation of groundwater is generally a long term strategy that requires huge investment. An efficient and cost-effective system is essential and computational modeling can aid in a decisive way.

The design of such a system can be solved as a simulation optimization problem, which couples optimization techniques with groundwater flow and mass transport simulators in an attempt to produce optimal remediation designs (Huang and Mayer, 1997). The remediation strategy considered is PAT, which requires the removal of contaminated groundwater from the aquifer by pumping. The purpose of optimal design is usually to determine how many wells to install, where these wells should be located and what pumping rate is required from each well, while minimizing a cost function (J. Guan, 1999). The most commonly used objective for the remediation design is to minimize costs associated with the pumping system in order to meet a specific human health risk target. 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 (Shreedhar Maskey and Solomatine, 2002).

A Genetic Algorithm (GA) (Eiben and Smith, 2003) is used as the optimization technique. The main advantage of GAs over conventional optimization methods lies in their ability to locate global optimum solutions to discrete, non-convex, and discontinuous problems (Goldberg, 1989). However, GAs are criticized since they normally evaluate hundreds or even thousands of scenarios - each requiring a numerical simulation execution - in the course of searching for the optimal solution to a giving management question. This process is extremely time-consuming and computationally expensive. Fortunately, GAs are embarrassingly parallel, and the possible solutions can be evaluated in different computational nodes.

The distributed approach uses the master/worker pattern (Mattson et al., 2004), where the master performs the optimization algorithm and assigns the simulations to the workers, distributed throughout available nodes, which execute instances of the simulator and calculate the quality of the proposed solutions. Experiments with 95 and 190 workers were executed and

used to analyse the performance of the distributed approach with regard to its scalability and its behavior concerning network usage. Since a heterogeneous environment is used, load balance is a critical issue and initial results of a feedback scheduling mechanism that uses information about workers past performance is presented.

The rest of this paper is organized as follows. Section 2 shows the aquifer model. In Section 3 the optimization technique and the characteristics of its implementation are described. Section 4 is dedicated to the distributed approach. In Section 5 computational experiments are presented and analysed. Conclusions are drawn in Section 6.

2 MODELING OF CONTAMINANT TRANSPORT

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, consisting of an elliptic system that comes from the conservation of mass and Darcy's law and a convection dominated convection-diffusion equation expressing the conservation of the contaminant. The differential system, under assumptions of a miscible and incompressible flow in a rigid porous media, is given by (Loula et al., 1999):

$$\mathbf{u} = -\frac{K}{\mu}\nabla p, \tag{1}$$

$$div\mathbf{u} = f, \tag{2}$$

$$\phi\frac{\partial c}{\partial t} + div(\mathbf{u}c) - div(D(\mathbf{u})\nabla c) = \hat{c}f, \tag{3}$$

with boundary and initial conditions

$$\mathbf{u}\cdot\mathbf{n} = 0. \tag{4}$$

$$D(\mathbf{u})\nabla c\cdot\mathbf{n} = 0, \tag{5}$$

$$c(\mathbf{x},0) = c_0(\mathbf{x}), \tag{6}$$

where p and u are the pressure and Darcy's velocity of the mixture, $\phi = \phi(x)$ and $K = K(x)$, the porosity and permeability of the medium, respectively, f is the source and sink terms associated with wells, and $\hat{c} = \hat{c}(x,t)$ is the injected concentration at injection wells and the resident concentration at extraction wells. The diffusion-dispersion tensor D is given by:

$$D(\mathbf{u}) = (\frac{\alpha_{mol}}{\tau} + \alpha_T\|\mathbf{u}\|)\mathbf{I} + \frac{\alpha_L - \alpha_T}{\|\mathbf{u}\|}\mathbf{u}\otimes\mathbf{u}, \tag{7}$$

where τ is the porous medium tortuosity, α_{mol}, α_L e α_T are, respectively, molecular diffusion, longitudinal and transverse dispersion coefficients.

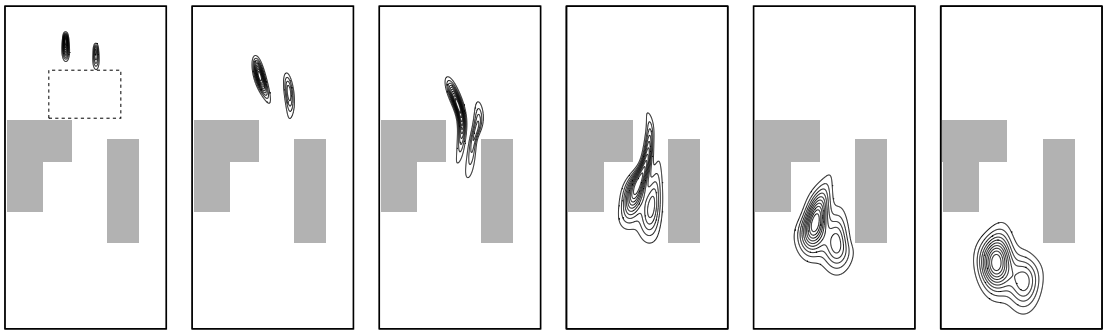
Note that the groundwater flow is considered steady state, while the time dependent nature of the transport equation is retained.

This system of partial differential equation is solved numerically using the Finite Element Method (FEM). The resulting system of linear equations is solved using a direct method which guarantees that all simulations will have the same computational cost.

The numerical simulator requires as input data: the parameters that define the geometry of the aquifer, the porous medium and the fluid's physical properties, the initial 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 a proposed solution.

2.1 A hypothetical aquifer

A hypothetical aquifer was used 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 25.0×25.0 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) Init. cond. (b) 400 days (c) 700 days (d) 1100 days (e) 1400 days (f) 1800 days

Figure 1: Hypothetical contaminated aquifer without remediation.

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 the system. Solutions for problems of groundwater quality control and remediation can be obtained with this technique, in particular, the design of pump and treat (PAT) systems, the most frequent application considered, including (Huang and Mayer, 1997; J. Guan, 1999; Ren and Minsker, 2005; Hilton and Culver, 2005; Chang et al., 2007; Sinha and Minsker, 2007; Kalwij and Peralta, 2008; Zou et al., 2009). The problem setting contains the usual optimization

components, which can assume the following characteristics, for a groundwater remediation system by PAT:

- the decision variables are the number of extraction wells to be installed, their locations and pumping rates;
- the objective function is usually a cost function that has to be minimized; and
- the constraints are defined by the restrictions imposed, such as: the locations where the wells can be placed, and the pumping rates available.

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 (Cai et al., 2001). GAs, as well as other non traditional optimization methodologies, have been proven as more effective at identifying high-quality solutions than analytical techniques, allowing the resolution of more complex nonlinear problems since it does not require continuity of the objective function or other assumptions such as convexity (Hilton and Culver, 2005; Espinoza et al., 2005). However, since GAs are more computationally expensive - requiring a large number of calls to the simulation model - hybrid techniques have been used lately: (Espinoza et al., 2005) proposed a hybrid GA that couples a simple GA with a local search algorithm, (Chang et al., 2007) integrated GA and constrained differential dynamic programming (CDDP), and (Kalwij and Peralta, 2008) used GA and Tabu Search (TS). Another approach was adopted by (Sinha and Minsker, 2007), where a multiscale island GA algorithm used different degrees of spatial grid discretization. In our approach, the numerical simulations required were done in parallel. The implemented GA is described in this section and its parallelization in the next.

3.1 Genetic algorithms

GAs are a type of Evolutionary Algorithm (EA), a family of computational models based on the concept of survival of the fittest (Goldberg, 1989; Colorni et al., 1996). 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** (Eiben and Smith, 2003).

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. This process is represented in the Algorithm 1.

To apply a GA to a problem, it is necessary: to identify a meaningful representation of the candidate solution, to identify a fitness function that properly measures the quality of each individual, to create a set of genetic operators that can efficiently select, recombine, and mutate the candidate solutions, and to choose a rule for parental substitution. The next subsections briefly describe how these functionalities were implemented.

1: Initialize population
2: **Evaluate individuals**
3: Repeat
4: Select individuals that will become the parents
5: Apply genetic operators to generate offspring
6: **Evaluate new individuals**
7: Select individuals for the next generation
8: Until termination condition is met

Algorithm 1: Genetic Algorithm

3.1.1 Representation of individuals

For the pumping strategy problem, each individual has information about the number of extraction wells, their locations and pumping rates. This information is stored, as real numbers, in a structure such as shown in Figure 2, where:

- n_W : number of extraction wells to be installed
- l_W : list of wells, containing n_W elements

  (x_i, y_i) : x and y coordinates of the i-th well location, $i = 1, .., n_W$

  (q_i) : pumping rate of the i-th well, $i = 1, .., n_W$

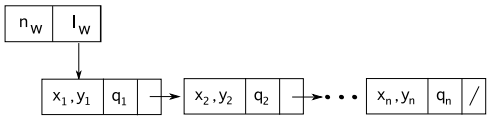


Figure 2: Representation of Individuals

In the experiments performed in Section 5, each individual could have a maximum of 9 extraction wells, and its size was at most 132 bytes.

3.1.2 Fitness function

The quality or fitness of individuals is calculated through an objective function that has to be maximized or minimized. Like many problems of practical interest, remediation of contaminated aquifers is characterized by multiple objectives, such us maximizing the quantity of removed contaminant and minimizing remediation costs. One approach when dealing with multi-objective problems is to build a single function that combines in some way the objectives. Remediation problems are generally optimized through minimization of a function that takes into account the cost of the system and penalty terms, used to enforce remediation standards in order to meet a specific human health risk target (Finsterle, 2004). The fitness function implemented in this work, based on (Geotrans, 2002), uses this strategy, and is shown in Equation 8.

$$f_{fitness} = CCE + CCTD + FCMS + VCE + VCTD + P * V_{res}, \tag{8}$$

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}).

3.1.3 Genetic operators

Genetic operators should be capable to, starting from existing individuals, generate new ones with a higher probability of having a better fitness. The basic operators are selection, recombination, and mutation.

Linear ranking was the selection scheme used, where the individuals are sorted in a list, according to their fitness, and the selection probability of an individual is proportionate to its position in the sorted list, not to its fitness, which could cause premature convergence (Goldberg and Deb, 1991).

Recombination or crossover combines information from two parents in order to create new offspring while mutation changes information from one parent to generate a new individual. For this problem of remediation using extraction wells, four operators were developed: recombination, mutation on the location of wells, mutation on the pumping rates of wells, and mutation on the number of wells. The frequency with which these operators are used is defined by rates that can be fixed or have a linear variation throughout the generations, according to parameters selected by the user in an input file.

3.1.4 Replacement strategy

Replacement strategy is a mechanism used to choose, among parents and the new offspring created, the individuals who will survive, producing the new generation.

In a generational GA with a population of size μ, λ new individuals are created at each generation, with $\lambda > \mu$. The replacement criteria can be one of the following: (i) μ parents are replaced by the μ best individuals of the offspring or (ii) μ parents are replaced by the μ best individuals from the union set formed by parents and offspring.

On the other hand, with a non-generational or steady-state GA, each new individual generated is immediately evaluated and then tested to be inserted in the population.

A strategy often used with generational GAs is elitism, where the K best individuals are copied to the nest generation, avoiding their loss. The implemented GA is generational and includes elitism, copying to the next generation $K = 2$ best individuals.

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 approach, 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/worker paradigm

To execute the numerical simulations in parallel, the master/worker paradigm is used. Master/worker is a simple yet widely used technique appropriate to execute independent tasks under the centralized supervision of a control processor (Beaumont et al., 2001; Mattson et al., 2004). For BoT applications running on a network of heterogeneous computers, master/worker as been the most common choice so far, due mainly to simplicity of implementation, tolerance to worker failures, and simple communication topology (Yero and Henriques, 2007). However, it is important to note that centralized master/worker approach may affect scalability, since the existence of a single master may become a bottleneck. This problem has been studied by many authors (Yero and Henriques, 2007; Silva and Senger, 2009; Chronopoulos et al., 2002). The use of a hierarchical master/worker structure reduces this effect. Nevertheless, the standard master/worker scheme is effective for a moderate number of processors (Brest and Zumer, 2000).

4.1.1 Implementation

Our approach is composed by three main modules:

- **Master**: written in Java, it follows the steps of the optimization algorithm and is responsible for scheduling the numerical simulations in the available workers.
- **Worker**: also written in Java, workers are replicated in the available computational nodes and call the simulator to calculate the quality of the proposed solutions.
- **Simulator**: a Fortran sequential program that solves the mathematical model of the aquifer, using FEM.

In our approach, a machine in the network is chosen to become the master, and the worker modules are replicated in the available nodes. Master and workers communicate via Remote Method Invocation (RMI), a mechanism that allows a Java program on one machine to invoke a method on a remote object in a transparent fashion, hiding the low-level communication between them (Li and Baker, 2005; Silberschatz et al., 2004).

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, which follows the steps of Algorithm 1 shown in Section 3.1. On steps 2 and 6 - evaluation of individuals - it partitions the population in blocks of individuals and distributes the blocks to workers. A worker evaluates all individuals of a given block, executing for each evaluation, the numerical simulator and using its output to calculate the fitness function. 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. Note that in order to proceed to the next generation, all individuals of the present generation must be evaluated and therefore the end of each generation becomes a point of synchronization.

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

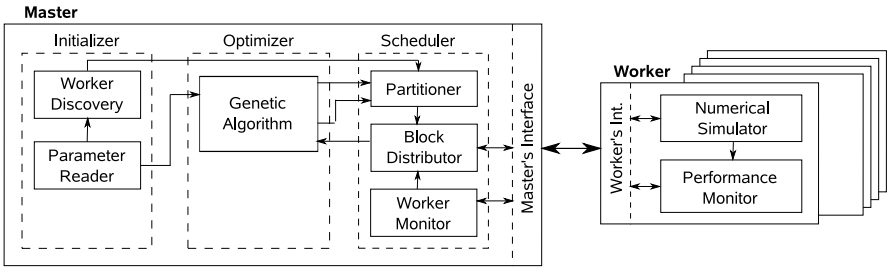


Figure 3: Implementation architecture.

4.2 Load Balancing

For intrinsically parallel applications, the use of many processing elements (PEs) will certainly reduce the response time. However, to achieve high performance in heterogeneous environments, the distribution of computational workload becomes critical.

Load balancing techniques are designed to equally spread the load on PEs, maximizing their utilization while minimizing the total application 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 workers to deliver the results of their evaluations.

Dynamic scheduling of tasks is one of the alternatives for addressing the problem of load balancing. Scheduling refers to the way tasks are assigned to run on the available PEs. As seen in the literature (Penmatsa et al., 2007; Yang et al., 2007; Cheng et al., 2004; Costa et al., 2008), dynamic scheduling is appropriate to achieve load balancing in heterogeneous or homogeneous non-dedicated environments. In general, under such scheduling, the job of size N is partitioned into M task blocks, where M is greater than the number of workers W, and blocks are assigned dynamically to PEs as they become available. Scheduling mechanisms where PEs request new tasks as they become idle are called self-scheduling.

In this implementation, the methods pure self-scheduling (PSS), guided self-scheduling (GSS), and factorial self-scheduling (FSS) were made available. However, in the experiments presented in 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 one individual), and in this way balances the workload. Using this technique, the maximum time a PE will be idle for is the processing time of the slowest PE. 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 (Costa et al., 2008), experiments with 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.

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. In this application, the idle time due to the need of synchronization at the end of each generation is the major cause of overhead and reduction in efficiency.

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 PEs. In (Costa et al., 2010) we looked at alternative ways of measuring and evaluating the parallel performance of a heterogeneous system and a dynamic way of calculating speedup and efficiency - dynamic speedup ($\overline{S}$) and dynamic efficiency ($\overline{E}$) - was proposed. It uses an approximation for T_1, called $\overline{T_1}$, calculated as the sum of the elapsed time of all the tasks performed by the workers ($\sum T_W$). These metrics consider the capacity of the used PEs at the time of application execution and were used to analyse the performance of the experiments done in this paper.

5 COMPUTATIONAL EXPERIMENTS

In section 5.1, experiments were done to demonstrate the use of this methodology. In section 5.2, they were conducted to verify the performance of the approach.

5.1 Optimization simulation model analysis

To test the efficiency and ability of the optimization methodology implemented, the aquifer model described in Section 2.1 was used. 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 $250 m^3/day$. 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 4. Well locations and pumping rates are shown in Figure 4(a) (■ = $200m^3/day$ and ▲ = $250m^3/day$). Figure 4(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 4(c) while the volume of contaminant extracted by each of the remediation wells is pictured in Figure 4(d). The value of the fitness function was $\$5,292K$ ($CCE = \$2,800K$; $CCTD = 713K$; $FCMS = 1,399K$; $VCE = 47K9$; $VCTD = 333K$).

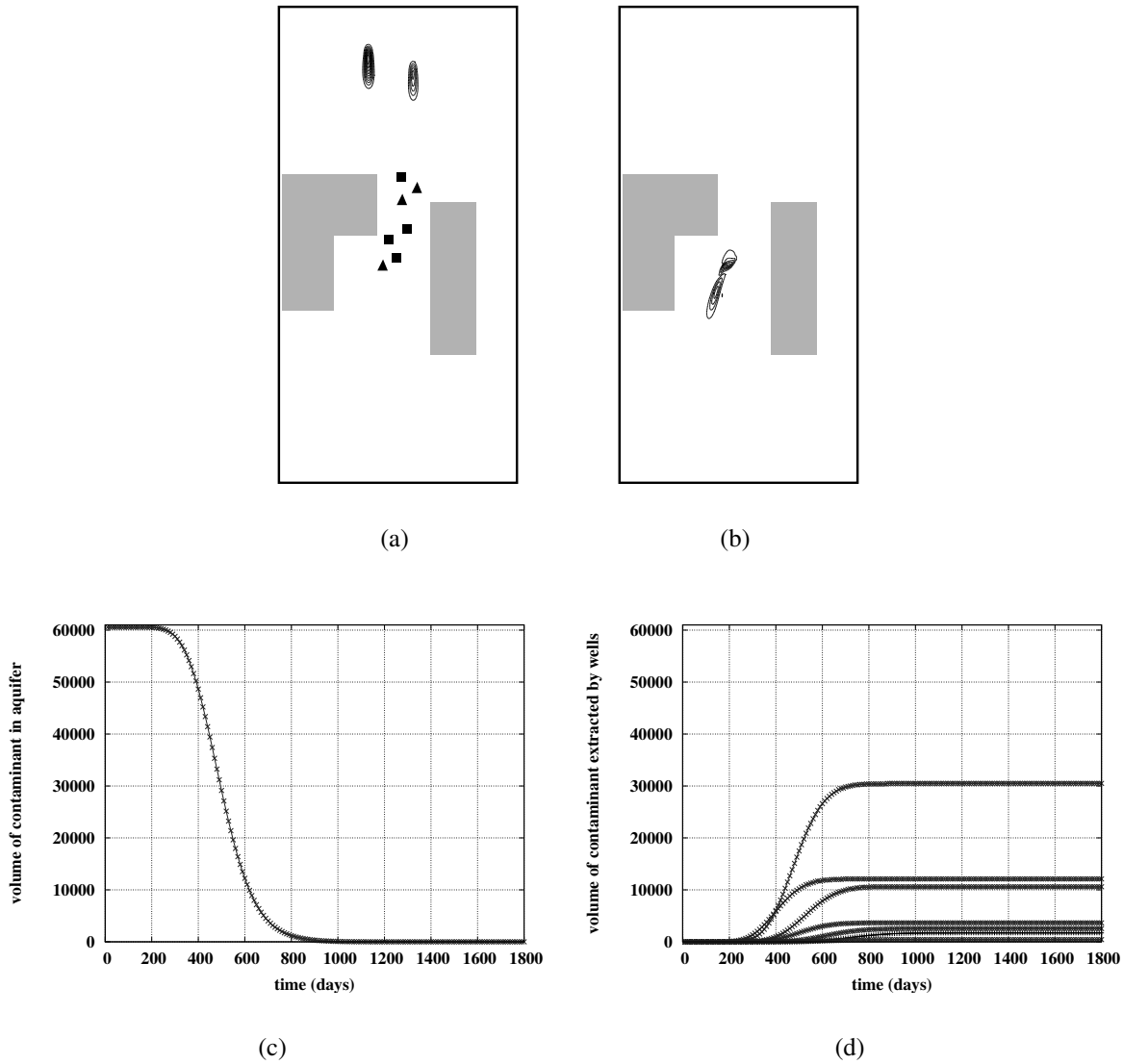


Figure 4: Remediation solution: (a) A initial contamination and extraction wells (contaminant volume: $60,000$); (b) Isoconcentration curves after 1230 days (contaminant volume: 0.01); (c) Volume of contaminant in the aquifer; (d) Volume of contaminant extracted by remediation wells.

5.2 Computational performace analysis

To analyse the performance of the distributed approach, to investigate its scalability, and to verify its influence on the network, large scale experiments with 95 and 190 workers, installed in a heterogeneous group of machines, were performed.

5.2.1 Computational environment

Approximately 40 different machines, distributed in separate labs at LNCC (National Laboratory for Scientific Computing), including one cluster, and connected via LAN were used. These machines 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.

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.

Two sets of 95 workers each were created: list **A** and list **B**, with the same processing capacity. The characteristics of the machines utilized and the number of workers installed in each one of them is listed on Table 1. The processor model and clock show the processing power of the machine and in the third column, the simulator average execution time for the numerical model used in the experiments is presented. Note that 20 workers were installed in machines that execute the numerical simulation in 32s (the fastest ones, which we call worker$_{32}$) and 16 workers were installed in machines that execute the same simulation in 72s (the slowest ones, named worker$_{70}$). In order to have the load balanced, it is expected that faster workers evaluate a larger number of simulations than the slower workers.

Processor Model	**Clock**	**Simulation Avg Time**	**Number of Workers**		
			list **A**	list **B**	list **A** + list **B**
Intel Xeon CPU E5440	2.83GHz	41s	24	24	48
Intel Xeon CPU E5520 (64 bits)	2.27GHz	41s	40	40	80
Intel Xeon CPU E5520 (32 bits)	2.27GHz	46s	4	4	8
Intel Core i7 CPU 920	2.67GHz	32s	10	10	20
Intel Pentium D	2.80GHz	70s	8	8	16
Intel Core2 CPU 6300	1.86GHz	56s	2	2	4
Intel Xeon CPU X7350	2.93GHz	52s	3	3	6
Intel Core 2 Quad CPU Q9300	2.50GHz	45s	4	4	8

Table 1: Characteristics of the machines used.

5.2.2 Experiments Description

To allow the analysis of the distributed approach with regard to its scalability and its behavior concernig network usage, different experiments were performed, using two different workloads (7700 and 15400 candidate solutions), two different amount of workers (95 and 190), and three different sets of workers (list **A**, list **B**, and list **A** + list **B**). These experiments were named **A**, **B**, $\mathbf{A}_B$, $\mathbf{B}_A$, **C**, and **D** and were defined by a set of optimization parameters and a group

of workers used to execute the optimization. Table 2 shows the GA parameters that influence the performance of the distributed approach: the number of generations and the number of candidate solutions in each generation, the total number of evaluations performed, the number of workers used, and the name of the list of workers. Experiments **A**, **B**, **C**, and **D** were conducted at different times. Aiming at identifying a possible bottleneck in the shared resources, mainly in the network, experiments **A** and **B** were performed simultaneously, and were denoted $\mathbf{A}_B$ and $\mathbf{B}_A$.

Exp.	Number of Generations	Candidate Solutions	Number of Evaluations	Number of Workers	List of Workers
A, $\mathbf{A}_B$	11	700	7700	95	list **A**
B, $\mathbf{B}_A$	11	700	7700	95	list **B**
C	11	700	7700	190	list **A** + list **B**
D	11	1400	15400	190	list **A** + list **B**

Table 2: Experiments

5.2.3 Scalability results

Table 3 reports the results of the experiments **A**, **B**, **C**, and **D**, presenting the following performance metrics: the elapsed time of parallel execution (T_P), the sum of workers processing time ($\sum T_W$), dynamic speedup ($\overline{S}$), dynamic efficiency ($\overline{E}$), the number of individuals processed per minute (Throughput), and the average time of each generation. Speedup and efficiency were calculated as shown in Section 4.3 and the number of PEs used to calculate efficiency was the number of workers added by one, accounting for the PE used to execute the master.

Exp.	T_P	$\sum T_W$	$\overline{S}$	$\overline{E}$	Throughput (ind/min)	Generation Avg Time
A	65.05min	5403.95min	83.07	0.87	118.37	354s
B	64.37min	5395.12min	83.81	0.87	119.62	353s
C	33.62min	5394.89min	160.47	0.84	229.03	183s
D	64.92min	10873.0min	167.48	0.88	237.22	354s

Table 3: Performance metrics

Experiments **A**, **B**, with the same size (7700 simulations) and using the same amount of computational resources, presented similar results: approximately, they all completed the optimization in 65min, which is 83 times faster (speedup) than if it had been done with one PE with an average processing power, using the resources with an efficiency of 87%, showing a stable behavior of the distributed approach.

In experiment **C** the efficiency dropped, since the amount of resources used was doubled while the size of the problem remained the same (7700 simulations). As expected, in experiment **D**, where the workload was increased in the same proportion as the amount of resources, the efficiency was higher than in **C**, and reached the same level of experimentss **A** and **B**. With a better utilization of the cores than experiment **C**, experiment **D** presented a higher throughput.

These results demonstrate that the efficiency can be kept fixed as the amount of computational resources and the size of the problem - the number of simulations - increase, showing that the distributed approach is scalable.

5.2.4 Communication influency

Table 4, similar to Table 3, presents performance metrics of experiments $\mathbf{A}_B$ and $\mathbf{B}_A$, which are the same as experiments **A** and **B**, respectively, except that $\mathbf{A}_B$ was executed at the same time as $\mathbf{B}_A$. Since they were executed simultaneously, they shared network resources, and this could have caused an increase in T_P (elapsed time of parallel execution). However $\mathbf{A}_B$ presented the same metrics as **A** while $\mathbf{B}_A$ presented the same metrics as **B**, leading us to conclude that the distributed approach does not overload the network.

Exp.	T_P	$\sum T_W$	$\overline{S}$	$\overline{E}$	Throughput (ind/min)	Generation Avg Time
$\mathbf{A}_B$	64.92min	5431.57min	83.67	0.87	118.61	354s
$\mathbf{B}_A$	64.55min	5429.15min	84.11	0.88	119.29	352s

Table 4: Performance metrics of experiments $\mathbf{A}_B$ and $\mathbf{B}_A$ (A and B executed simultaneously)

The number of communications that occured during each experiment are shown in Table 5. In our approach, communication happens every time the master has to send a block of individuals to a worker and again when that worker sends the block of individuals, with their fitness calculated, back to the master. In the experiments presented, PSS (pure self-scheduling) strategy was used and since PSS divides the task (evaluation of a population) of size N in N blocks of size 1, during each experiment, $2 \times N$ communications happened . Each block has 1 individual and each individual has 132 bytes. Experiments $\mathbf{A}_B$ and $\mathbf{B}_A$ were performed simultaneously and therefore, during their execution time $2 \times 2 \times 7700$ communications ocurred. The size of the each individual (132 bytes) is considered small, but the amount of communications required could be a bottleneck, which did not happen.

Exp.	Number of Communications	Size of Individual
A	2×7700	132 bytes
B	2×7700	132 bytes
$\mathbf{A}_B$	$2 \times 2 \times 7700$	132 bytes
$\mathbf{B}_A$	$2 \times 2 \times 7700$	132 bytes
C	2×7700	132 bytes
D	2×15400	132 bytes

Table 5: Amount of communications during the period of each experiment

The use of RMI and PSS for this type of application, using resources connected via LAN, presented satisfactory results.

5.2.5 Better scheduling results

The implemented GA is generational, as seen in Section 3.1.4, which means that all individuals of a given generation have to be evaluated before the calculations of the next generation take place. This produces points of synchronization at the end of each generation, in particular, in these experiments with 11 generations each, there are 11 moments of synchronization.

It is important to analyse the behavior within the generations since it brings information that helps to understand the performance of the distributed approach and identify the causes of loss

of efficiency observed in the experiments presented, even though the application is embarrassingly parallel.

Within each experiment, all generations had similar behavior. The time measurements presented in Table 6 were obtained at the second generation, and are the following: total number of simulations performed at that generation (Total N_{sim}), generation time ($T_{generation}$), average time that each worker spent idle (T_{idle}), for the first and last workers to finish: its total processing time (T_W), how many simulations it performed (N_{sim}), and its average simulation time (T_{sim}), and the difference in time between the first and last worker to finish their tasks (T_{diff}).

Exp.	Total N_{sim}	$T_{generation}$	T_{idle} (avg)	First Worker to Finish			Last Worker to Finish			T_{diff}
				T_W	N_{sim}	T_{sim} (avg)	T_w	N_{sim}	T_{sim} (avg)	
A	700	354s	44.6s	284s	9	32s	354s	5	71s	70s
B	700	353s	42.8s	283s	9	32s	353s	5	71s	70s
$\mathbf{A}_B$	700	354s	42.4s	285s	9	32s	354s	5	71s	69s
$\mathbf{B}_A$	700	352s	40.7s	284s	9	32s	352s	5	70s	68s
C	700	183s	28.2s	126s	3	42s	183s	3	61s	57s
D	1400	354s	42.1s	284s	9	32s	354s	5	71s	70s

Table 6: Performance metrics from the 2^{nd} generation

In experiment A, at the given generation, the fastest worker ($worker_{32}$) executed 9 simulations while the slowest one executed 5 simulations ($worker_{70}$). This reflects the behavior of all $worker_{32}$ and $worker_{70}$ and this balance was obtained with the use of the PSS mechanism. However, right before the fastest worker finished its tasks (at 284s), one of the last simulations was assigned to the slowest worker, causing the fastest one to be idle for 70s, which is the worst case when using this scheduling strategy, as seen in Section 4.2. This generated an average idle time of 44.6s per worker. Similar situations happened on experiments B, A_B, B_A and D.

As concluded in (Costa et al., 2010), in this approach the idle time of PEs at the end of each generation is the main cause of loss of efficiency and the use of an intelligent scheduling mechanism (Zhang et al., 2008; Huang et al., 2004) that uses information from previous generations to choose whether or not to assign more tasks to slower workers, is an alternative way to improve performance. We have been working on a scheduling mechanism, that distributes the last tasks to workers based on their performance on the last generation. We called it feedback scheduling and some initial results obtained for experiments A and D are presented on Table 7.

Exp.	T_P	T_W	$\overline{S}$	$\overline{E}$	Throughput (ind/min)	Generation Avg Time	T_{idle} (avg)
A	59.70min	5341.9min	89.48	0.93	129.98	325s	19.3s
D	60.23min	10751.6min	178.5	0.93	255.67	328s	19.8s

Table 7: Performance metrics for feedback scheduling experiments

It can be observed, when compared with values from Table 3, that T_P (elapsed time of parallel execution) decreased from 65.05min to 59.70min on experiment **A** and from 64.92min to 60.23min on experiment **D**. If an optimization with more generations was executed, as the one on the experiment of Section 5.1 which used 150 generations, the use of this type of scheduling could represent substantial gain in response time.

In Table 6, it was shown that on experiments **A** and **B** without feedback, $worker_{70}$ performed 5 evaluations and $worker_{32}$ performed 9 evaluations. Using feedback scheduling, all $worker_{70}$

performed 4 evaluations, while some worker_{32} performed 10 simulations. With better utilization of the PEs, the average idle time per worker decreased and efficiency went up to 0.93.

6 CONCLUSIONS

The paper showed a distributed approach used to automatically find remediation solutions to a hypothetical contaminated aquifer. This approach has been studied by the authors and was presented in previous publications. The behavior of the distributed approach with large scale parallel experiments using a heterogeneous environment was presented here.

To analyse scalability, different experiments were performed, using two different workloads (7700 and 15400 candidate solutions), two different amount of workers (95 and 190), and three different sets of workers (list **A**, list **B**, and list **A** + list **B**). Results demonstrated that the efficiency can be kept fixed as the amount of computational resources and the workload increase, showing that the distributed approach is scalable.

To verify the behavior of the distributed approach concerning network usage, 2 experiments were performed, first at distinct times and then simultaneously. Since they were executed simultaneously, they shared network resources, and this could have caused an increase in elapsed time of parallel execution, which did not occur and led us to conclude that the distributed approach does not overload the network. The use of RMI and PSS for this type of application, using resources connected via LAN, presented satisfactory results.

Since the GA implemented is generational, the idle time of PEs at the end of each generation is the main cause of loss of efficiency. The use of an intelligent scheduling mechanism that uses information from previous generations to choose whether or not to assign more tasks to slower workers, was suggested in a previous work as an alternative way to improve performance. Initial results of a feedback scheduling technique, that is being developed, were presented and showed better performance of the approach: reduction of parallel execution time and of idle time and increase in speedup, efficiency, and throughput.

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.

REFERENCES

Beaumont O., Legrand A., and Robert Y. The master-slave paradigm with heterogeneous processors. *IEEE International Conference on Cluster Computing*, 0:419–426, 2001.

Brest J. and Zumer V. Solving asymmetric traveling salesman problems using dynamic scheduling on a heterogeneous computing system. 2000.

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

Chang L.C., Chu H.J., and Hsiao C.T. Optimal planning of a dynamic pump-treat-inject groundwater remediation system. *Journal of Hydrology*, 342:295–304, 2007.

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

Chronopoulos A., S.P., and Yu N. Scalable loop self-scheduling schemes for heterogeneous

clusters. In *Proceedings of the IEEE International Conference on Cluster Computing*. 2002.

Colorni A., Dorigo M., Maffioli F., Maniezzo V., Righini G., and Trubian M. Heuristics from nature for hard combinatorial optimization problems. *International Transactions in Operational Research*, 3:1–21, 1996.

Costa P.A., Garcia E.L., Schulze B., and Barbosa H.J. Evaluation of a distributed numerical simulation optimization approach applied to aquifer remediation. *Procedia Computer Science*, 1(1):7 – 16, 2010. ISSN 1877-0509. doi:DOI:10.1016/j.procs.2010.04.003. ICCS 2010.

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

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

Eiben A. and Smith J. *Introduction to Evolutionary Computing*. Springer-Verlag, 2003.

Espinoza F.P., Minsker B.S., and Goldberg D.E. Adaptative hybrid genetic algorithm for groundwater remediation design. *Journal of water resources planning and management*, 131(1):14–24, 2005.

Finsterle S. Demonstration of optimization techniques for groundwater plume remediation using itough2. Technical Report, Earth Science Division, Lawrence Bekerley National Laboratory, 2004.

Geotrans. Transport optimization Hastings naval ammunition depot. 2002. Hastings Documentation Formulations.

Goldberg D.E. *Genetic Algorithms in search, optimization and machine learning*. Addison Wesley Publishing Company, 1989.

Goldberg D.E. and Deb K. A comparative analysis of selection schemes used in genetic algorithms. In G.J. rawlins, editor, *Foundations of Genetic Algorithms*. Morgan Kaufmann Publishers, 1991.

Hilton A.B.C. and Culver T.B. Groundwater remediation design under uncertainty using genetic algorithms. *Journal of water resources planning and management*, 131(1):25–34, 2005.

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

Huang C.Q., Chen D.R., and Hu H.L. Intelligent agent-based scheduling mechanism for grid service. In *Proceedings of the Third International Conference on Machine Learning and Cybernetics, Shangai*. 2004.

J. Guan M.A. Optimal remediation with well locations and pumping rates selected as continuous decision variables. *Journal of Hydrology*, 221:20–42, 1999.

Kalwij I.M. and Peralta R.C. Non-adaptive and adaptive hybrid approaches for enhancing water quality management. *Journal of Hydrology*, 358(3-4):182 – 192, 2008.

Li M. and Baker M. *The Grid Core Technologies*. John Wiley An Sons, 2005.

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

Mattson T.G., Sanders B.A., and Massingill B.L. *Patterns for Parallel Programming*. Addison-Wesley, 2004.

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

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

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

Silberschatz A., Gakvin P.B., and Gagne G. *Operating System Concepts*. John Wiley and Sons, Inc, 2004.

Silva F.A. and Senger H. Improving scalability of bag-of-tasks applications running on master-slave platforms. *Parallel Computing*, 35:57–71, 2009.

Sinha E. and Minsker B. Multiscale island injection genetic algorithms for groundwater remediation. *Advances in Water Resources*, 30:1933–1942, 2007.

Yang C., Cheng K., and Shih W. On development of an efficient parallel loop self-scheduling for grid computing environments. *Parallel Computing*, 33:467–487, 2007.

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

Zhang Y., Coleman P., and Pellon M. A new load balancing scheme for distributed multi-agent simulations. *Journal of Simulation*, pages 955–961, 2008.

Zou Y., Huang G.H., He L., and Li H. Multi-stage optimal design for groundwater remediation: a hybrid bi-level programming approach. *Journal of Contaminant Hydrology*, 108:64 – 76, 2009.