ON SCALABILITY IN SIMULATION AND OPTIMIZATION OF COMPLEX SYSTEMS

Manfred Grauer and Frank Thilo
Information Systems Institute, University of Siegen
Hölderlinstr. 3, D-57068 Siegen, Germany

ABSTRACT

Advances in processing power of modern computer hardware allow the analysis of increasingly complex systems by means of simulation. However, many engineering problems such as design optimization problems in the aircraft industry, facility management in the water industry or design problems in the automotive industry have extremely high computational demands. To solve these problems in reasonable time, parallel computing must be utilized. The availability of affordable parallel computing systems in the form of network of workstations or compute clusters as well as the dawn of grid computing and servcice-oriented architectures is beginning to make parallel computing accessable to a broader community.

To exploit the total processing power of many CPUs, intelligent, scalable algorithms are required. In this paper, the concept of scalability is examined in the context of parallel simulation software systems and the distributed solution of simulation-based optimization problems.

Keywords: computational engineering, direct search, evolutionary algorithm, scalability, parallel optimization

1. INTRODUCTION

Most optimization problems in the field of computational engineering cannot be formulated analytically but are instead modeled by using simulation software systems like NASTRAN, ANSYS, ABAQUS, and many others. These systems provide an accurate simulation of a single design with respect to a given set of parameters. The task of an optimization algorithm is to find the best set of these parameters (decision variables) with respect to an objective function while not violating existing constraints. For each set of decision variables, the corresponding model must be computed, which in turn involves performing a costly simulation run. The simulation-based approach often implicates that the objective function is highly non-linear, and the region of feasible solutions is non-convex or even disjointed.

A suitable optimization algorithm must take these characteristics into account. Firstly, it must be able to deal with the excessive computation times by evaluating several possible solutions in parallel. Secondly, since no assumptions on convexity and smoothness of the objective and constraint functions can be made, gradient-based algorithms are not reliable.

The cost of parallel supercomputers, particularly in the form of compute clusters, has dropped significantly during the last decade, making them available to a wider range of institutions and companies. Furthermore, advances in the area of service-oriented architectures [1] make it easier to use resources beyond geographical and organizational boundaries. The problem of licensing often still prohibits the use of commercial simulation software in these environments, however. Thus, while most of the observations will also apply to large scale computing, the focus in this contribution is on single cluster environments.

The availability of a high degree of parallelism (more than 100 CPUs) helps to reduce the computation time only if the algorithms are able to exploit this larger number of CPUs, i. e. if they are *scalable*. In this paper, the meaning of this term is examined in the context of simulation-based optimization. The general notion of scalability is discussed and a way to compare the parallel performance of heuristic optimization algorithms is described. Three such direct search algorithms are briefly presented and their performance for both a benchmark and a real-world problem from groundwater management is analyzed over a range of 1 to 200 CPUs.

The paper is organized as follows: Section 2 introduces the concept of scalability and particularly focuses on the area of computational engineering. The next section introduces simulation-based optimization, describes three algorithms which are suited for distributed optimization, and presents scalability results for a test problem. In section 4, the findings are validated by experimental results from industrial applications. Section 5 then concludes the paper.

2. THE SCALABILITY CONCEPT

The term *scalability* is often used in different contexts to express that a computer system or an algorithm can cope with an increased workload or is able to solve a given problem faster when resources are added. Adding resources can refer to replacing components like CPUs by faster versions, increasing the main memory, or adding more components, e.g. additonal nodes to a compute cluster. The e-business community has coined the terms *scale up* (or *scale vertically*) and *scale out* (or *scale horizontally*) to denote these [2].

Scalability analysis can be divided into *algorithmic* and *architectural scalability* [3]. While the first focuses on attributes of an algorithm, i.e. the algorithm's sequential portion, its inherit concurrency limits and synchronization costs, the latter examines hardware related aspects as processing capacity, information capacity and connectivity. To predict real runtimes of a prallel algorithm on a given hardware architecture, both kinds or analyses must be taken into account, in particular whenever the communication costs are significant

Different applications have different scalability goals. For online transaction systems or web servers, the performance is often measured in transactions per second or the number of concurrent users that the system can handle before performance degrades. Here, the workload consists of many small tasks which have to be processed in real-time within a given acceptable time. Storage performance is often more important than CPU speed. On the other hand, the field or parallel computing or computational engineering is mostly concerned with a single long-running task whose sequential runtime would be so great that it is imperative to process it in parallel on several CPUs. This is the case for simulation-based optimization.

The classical metric to quantify scalability behavior is *speedup*, which compares the computation times of a parallel algorithm for different numbers of CPUs *p*, where the related metric *efficiency* is speedup divided by *p* (s. [4]). In the case of heuristic, parallel optimization algorithms, there is no clearly defined goal for which each algorithm's elapsed time could simply be compared. Instead, both the time needed and the quality of the solution must be considered, e.g. by examining the progress of the achieved solution quality over time. To still be able to use the notion of speedup, a target solution quality can be defined and the time needed to reach this level be used as the basis to calculate speedup and efficiency values.

3. SCALABILITY IN DISTRIBUTED OPTIMIZATION

In this section, the general concept of simulation-based optimization is introduced. Some characteristics of this kind of optimization problems are identified which leads to a group of algorithms which can be applied to the problem. Three such algorithms are briefly described and their scalability is analyzed for solving a test problem.

3. 1. Simulation-based Optimization

Optimization problems which arise in the field of computational engineering usually cannot be formulated analytically because of their complexity. Instead, a model of the real problem is created, typically in the form of a data set for a simulation software package. The computational demands of these simulations can be very high, ranging from a few minutes to several days for a single simulation. For optimization, the model is parametrized by some variables which can be chosen within lower and upper bounds. The goal is to find the best set of variables as defined by the objective function which has to be minimized. Possible objective functions are the weight of a constructional element (such as an aircraft wing), the electrical power consumption of a pumping well, the aerodynamic resistance of a car chassis and many more. Furthermore, the set of possible solutions is limited by constraints, e.g. a given ascending force for a plane, the level of groundwater at certain locations or the minimum measures and volume of a car's passenger compartment.

During the course of the optimization, hundreds or thousands of solution candidates must be evaluated, requiring a costly simulation run each time. While the time for a single simulation is significant (typically ranging from a few minutes to several hours or even days), the amount of processing within the optimization algorithm itself is several orders of magnitude lower. Also, the amount of data that must be exchanged between the search method and the simulations is very low. Thus, communication costs are not the defining factor which limits the scalability of the algorithms, but rather their inherent concurrency limits, their synchronisation points and their change in internal parameters in order to adapt to the number of available resources which can cause a drop in effectiveness. The findings are that for this type of simulation-based optimization problems the relation of computation costs to the communication costs are at least 1000 to 1.

The simulation-based nature of the problem means that in general no derivative information is available and no

assumptions can be made about the nature of the search space. This prohibits the use of linear programming techniques or gradient-based optimization algorithms. One class of algorithms which can be applied are so-called direct search methods [5, 6]. There is no excact definition of direct search, but important characteristics are, that these methods do not explicitly use derivative information nor build a model of the objective function. Instead, the basic operation relies on direct comparison of objective function values. To utilize parallel computing resources, parallel direct search methods are needed, which can evaluate several solution candidates simultaneously. Below, three such methods are evaluated.

3. 2. Optimization Algorithms

Three parallel direct search methods are compared: The Distributed Polytyope algorithm (DPA) [7], a parallel implementation (PSS) of the meta-heuristic scatter search [8, 9], and asynchronous parallel pattern search (APPS) [10].

DPA belongs to the class of simplex-based search methods. It generates new solution candidates by applying geometrical operations to a set of previously calculated solutions. This set typically contains 2*n* points (where *n* is the number of decision variables) and is called the polytope. The initial set of feasible solutions is generated by a simple parallel random search strategy. During the main exploration phase, new points are generated by reflecting or contracting existing solutions relative to the weighted center of gravity. The number of operations that is performed in each operation depends on some parameters which can be adjusted to utilize the available number of CPUs. Infeasible points (i.e. points which violate the constraints) are modified by a binary search repair strategy (see Fig. 1). At the end of each iteration, the best of the old and new solutions are selected for the new polytope. After some iterations, the polytope will converge and the optimization is concluded with a parallel local search within the neighborhood of the best known solution.

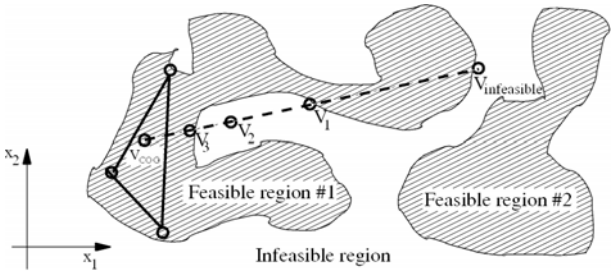

Figure 1: Repair of the infeasible solution $v_{Infeasible}$ in a search space with disjoint feasible regions; a binary search towards the weighted center of gravity v_{COG} produces the points v1, v2 and v3

PSS can be viewed as an evolutionary approach, but it has some attributes in which it differs from most algorithms of this class. At the heart of the algorithm is the so-called reference set which is initially built by a diversification algorithm which semi-randomly creates at least 100 points and tries to spread them evenly among the range defined by the bounds for each decision variable. The size of the reference set is fixed and typically in the range of 10 - 20. In each iteration, all possible pairs and 3-tuples which contain at least one new solution are combined to create new candidate solutions. The combination is performed as a simple linear combination with a random factor, independently for each component (decision variable) of the solutions. This typcally results in several hundred new points which are then evaluated simultaneously. The new reference set is then built, choosing both the best and most diverse of the newly created and old solutions. Infeasible solutions are moved towards a known feasible solution in a step called path relinking and are added to the evaluation queue for the next iteration. Whenever the standard deviation of the objective function values in the reference set drops below a given threshold, new random solutions are created to increase diversity. After a fixed number of these steps, the algorithm terminates.

The third algorithm, APPS, belongs to the group of pattern search optimization algorithms which use a set of search directions to create new points based on the best known solution so far. By default, APPS uses two search directions for each decision variable, namely the set of plus and minus unit vectors. Thus, 2*n* new points are created and evaluated in parallel and infeasible points are discarded. The best of the new points is then chosen as the new starting point for the next iteration. If no better point could be found, the step length is decreased. The algorithm terminates when the step length drops below a given threshold. However, APPS does not work iteratively, but instead works asynchronously, i.e. it does not wait for all points to be

evaluated before it creates new candidate solutions, but can instead continue as soon as one evaluation has finished and has resulted in a new best solution. In this respect, it differs from the synchronous approaches of both DPA and PSS.

3. 3. A Test Problem

To allow an extensive scalability analysis over a wide range of problem dimensions and numbers of CPUs, a mathematical test problem is defined based on the well-known Rosenbrock function (s. [11]). The evaluation of this function is trivial and takes no significant amount of time. To mimic the temporal behaviour of a real, simulation-based problem, an event-based simulation is used which keeps track of virtual wall clock time. Each evaluation of the objective function is assigned a fixed amount of virtual time. To validate the results, a subset was compared with those of a real distributed optimization on a compute cluster. For times of 10s per evaluation, the average error in wall clock times is already below 0.5% and is even lower for the more typical simulation times of at least several minutes.

As explained in Section 2, speedup is defined as the ratio of the time needed to reach a given solution quality on one CPU and the corresponding time for p CPUs. Figures 2, 3, and 4 depict the speedup of DPA, PPS, and APPS for solving several high-dimensional (n = 10 … 100) Rosenbrock problems. Up to 200 (virtual) CPUs have been used and each point is the average of 200 optimization runs with different pseudo random number generator seeds. For DPA it can be seen, that the speedup is good up to about 30 CPUs for all problem sizes. However, the efficiency drops significantly for higher numbers of CPUs and small dimensins of the problem. The 100-dimensional problem scales well beyond a CPU count of 100.

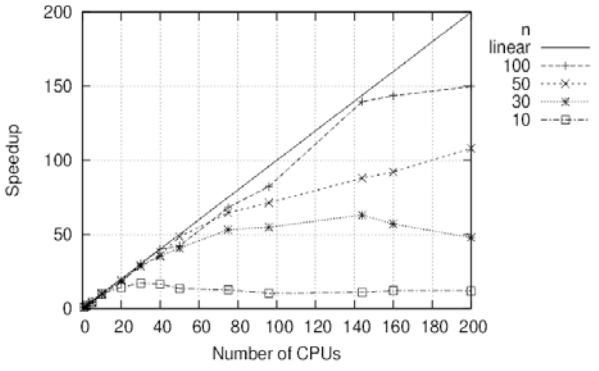

Figure 2: Relative speedup of DPA for solving 10- to 100-dimensional Rosenbrock problems on 1 to 200 CPUs (average of 200 runs)

In contrast, PSS exhibits almost linear speedup over the full range of problem sizes and numbers of CPUs. The efficiency only drops slightly as the degree of parallelism is increased. As with DPA, the algorithm scales better for higher number of decision variables, although the effect is much less visible.

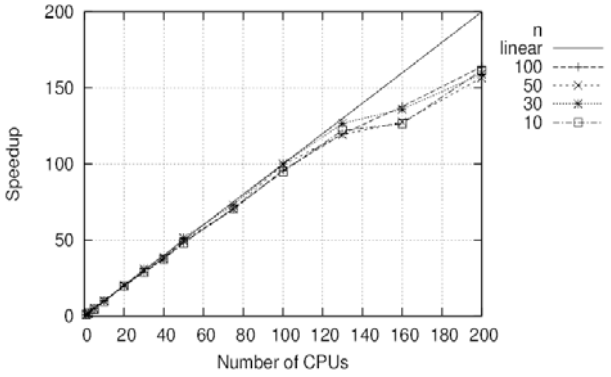

Figure 3: Relative speedup of PSS for solving 10- to 100-dimensional Rosenbrock problems on 1 to 200 CPUs (average of 200 runs)

For APPS, the picture is not so clear. The achieved speedups are worse than those of PSS and the correlation between problem dimension and efficiency is much weaker than for DPA. For some numbers of CPUs, the speedup is actually lower than with a smaller number of CPUs. A possible explanation is that because of the asynchronous nature of the algorithm, the resulting search path can strongly depend on the number of available CPUs.

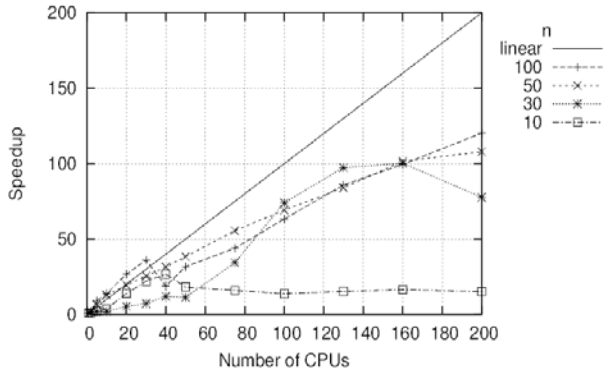

Figure 4: Relative speedup of APPS for solving 10- to 100-dimensional Rosenbrock problems on 1 to 200 CPUs (average of 200 runs)

Of the three algorithms, PSS has by far the highest efficiency when the number of CPUs is much higher than the dimension of the search space. This does not necessarily imply that the optimization process is faster when comparing the absolute time to reach the same solution quality. Fig. 5 and 6 compare the absolute

performance by displaying the algorithms' progress over time. In Fig. 6, with $n = 10$ and $p = 100$, PSS operates at a much higher parallel efficiency than both DPA and APPS. However, DPA is able to reach objective function values of 60 or higher faster than APPS and reach better values after about the same time. For a CPU count of 10 in Fig. 5, DPA outperforms both PSS and APPS significantly over the whole spectrum (note that both axes are logarithmic).
For example, it takes DPA an average of about 25s to find a solution with an objective function value of 100, whereas PSS and APPS need 90s and 80s, respectively.

The results indicate that each algorithm exhibits different scalability characteristics. Scatter search has the highest efficiency for large numbers of CPUs, in particular for problems with few decision variables. However, it takes more absolute time to reach a given solution quality than the other two algorithms when using only a small number of CPUs.

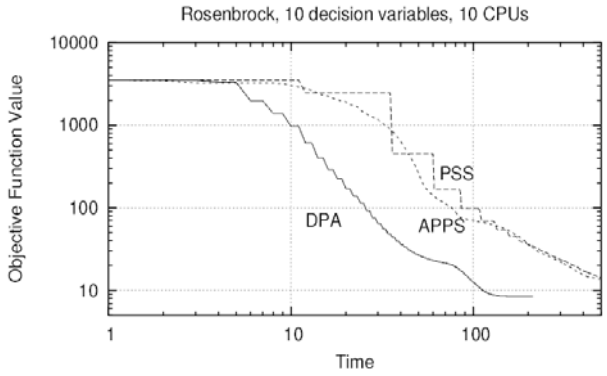

Figure 5: Comparison of the solution quality over vcirtual time; 10-dimensional Rosenbrock problem, solved using 10 CPUs (average of 200 runs)

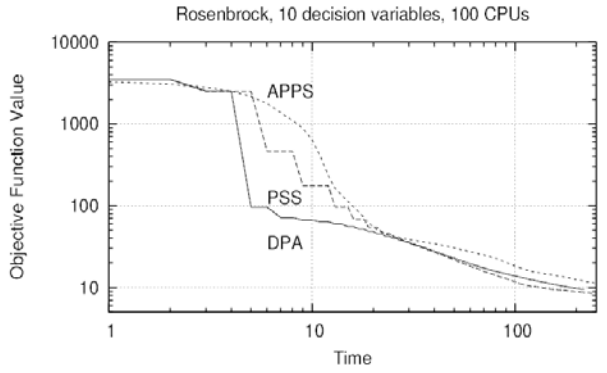

Figure 6: Comparison of the solution quality over virtual time; 10-dimensional Rosenbrock problem, solved using 100 CPUs (average of 200 runs)

4. SCALABILITY IN INDUSTRIAL APPLICATIONS

The aforementioned algorithms have been used to solve several real-world optimization problems. To achieve this, they have been integrated into the OpTiX optimization environment which presents a common abstract interface of the problem to the algorithms and handles the distribution and scheduling of the distributed simulation runs (s. [12]). All computations have been performed on the Rubens cluster at the University of Siegen which consists of 128 Dual-Opteron nodes at 2 GHz with 2GB of RAM and running SuSE SLES 8.1 operating system.

The past and current problems include a design problem of an aircraft wing, several design and hybrid control problems in groundwater and pollution management, a multi-stage problem in metal-sheet forming and optimization of casting processes of automotive parts. In the following, the algorithms' performance for solving one of the groundwater problems is analyzed.

The FEM-based software package FEFLOW [13] has been used for modeling and simulation of the Binsheimer Feld problem [14]. Mining activities at the lower Rhine cause massive subsidence in nearby areas. To prevent surface water logging, extensive draining measures must be performed. Several pumping stations are controlled by an adaptive strategy that changes the current pumping rate in dependence of the measured water level in a control point as depicted in Fig. 7. The strategy for each pump is defined by 6 to 9 switch levels. For optimization, three pumps are considered, resulting in a total of 22 switch levels which constitute the decision variables. The objective function to be minimized is the sum of energy costs of the pumps over the course of 5 years of operation. As constraints, a minimum depth to water table must be maintained at 11 critical observation points at all times. This is a problem of optimal design of a hybrid control system [15] with an additional non-linear cost function.

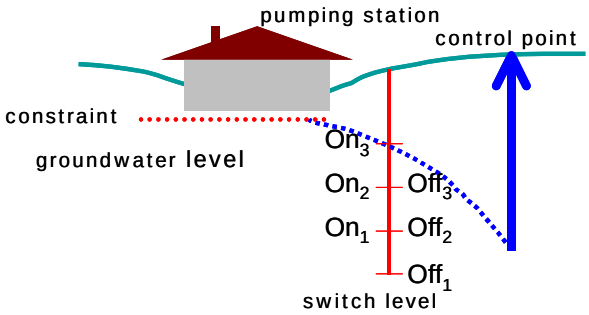

Figure 7: Adaptive pumping station control, depending on switch levels and groundwater level in control observation point

A single simulation of this problem on a 2GHz Opteron system takes about 170s to 350s, depending on how often and how strongly the pumping strategy changes the pumping rates. Fig. 8 depicts the distribution of the simulation times as a histogram. The spread in evaluation times can be expected to reduce the performance of the synchronous algorithms DPA and PSS because they have to wait for all of the simulations to finish at the end of each iteration before they can continue. The CPUs on which the simulations finished early are thus idle for a fraction of the time. In contrast, the asynchronous pattern search should not suffer from this problem.

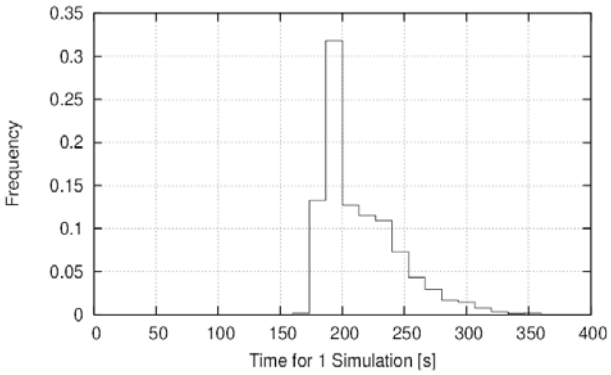

Figure 8: Distribution of the time needed for one simulation of the Binsheimer Feld problem

Table 1 shows the average CPU utilization for the three algorithms while solving the Binsheimer Feld problem. The effect is visible, but not very strong, because most of the simulations will be performed close to the best found solution and the difference in evaluation times within such a small region of the search space is not so big.

algorithm	*avg. CPU utilization*
DPA	91.5%
PSS	95.7%
APPS	98.5%

Table 1: CPU utilization while solving the Binsheimer Feld problem (average of 5 runs, 40 CPUs)

In Fig. 9 and 10, the average solution quality while solving the Binsheimer Feld problem with 10 and 80 CPUs is shown. The starting solution which is passed to the optimization algorithms as an initial guide has a cost of 992765 €. With 10 CPUs, DPA is by far the fastest to find a good solution which has a cost value of less than 800000 €. At 80 CPUs, the performance of PSS becomes similar to that of DPA. In addition, PSS is able to reach the best objective function values of the three algorithms when given enough time. APPS is only able to find solutions of moderate quality, but the solution quality improves when it is given more CPUs. However, even with 80 CPUs, it is not able to get below 810000 €.

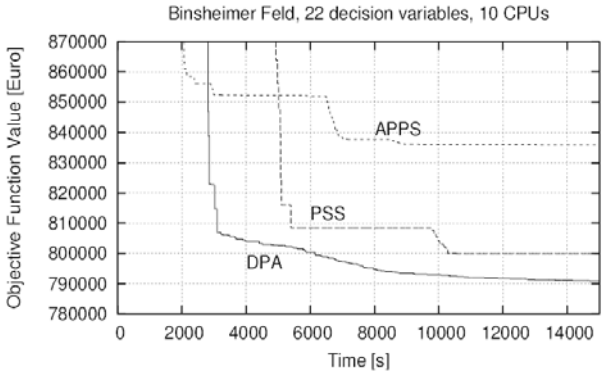

Figure 9: Comparison of the solution quality over wall clock time; 22-dimensional Binsheimer Feld water management problem, solved using 10 CPUs (average of 20 runs)

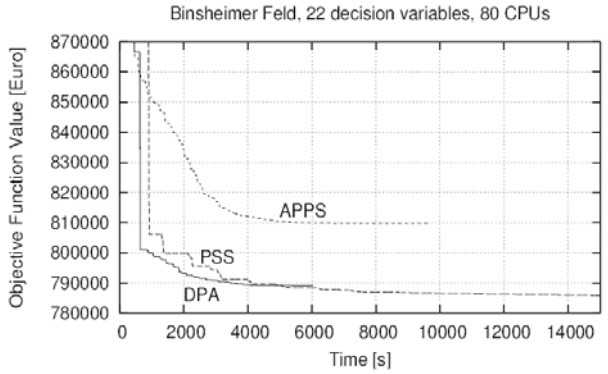

Figure 10: Comparison of the solution quality over wall clock time; 22-dimensional Binsheimer Feld water management problem, solved using 80 CPUs (average of 20 runs)

One way to further enhance the speedup and efficiently utilize larger number of CPUs is multi-level parallelism: The simulation packages that are used for the casting processes and the metal-sheet forming can be run in parallel on several CPUs themselves. This introduces a second level of parallelism into the optimization process: The search algorithm evaluates a set of solution candidates in parallel by starting a simulation per solution, and each of these simulations itself can be run data-parallel on more than one CPU. The resulting degree of parallelism is thus the product of the two. The scalability characteristics of both the optimization algorithm and the parallel simulation software must be taken into account to find the best allocation of resources. The performance of the parallelized simulation is often limited by the latency and bandwidth of the network and thus high-performance network hardware like Myrinet [16] might be required. If both levels of parallelism are combined intelligently, the approach promises to yield significantly higher parallel efficiencies for large numbers of CPUs as would be possible otherwise.

5. CONCLUSIONS

In this contribution, three different scalable distributed algorithms (distributed polytope, a parallel implementation of scatter search and an asynchronous parallel pattern search) were tested and results from computational experiments with different dimensions (n = 1, … , 100) of the optimization problem and different numbers of available processors (p = 1, …, 200) reported. The analysis was made along the two contradictory objectives of the efficiency of the use of the available processors and the quality of the solution. The experiments were made with an academic problem to test the distributed solution concepts and with an industrial problem from groundwater engineering. In both cases the distributed polytope method showed good results. The intelligent design of scalable algorithms in optimization seems to be an important contribution to deal with future problems which are multidisciplinary, multi-agent, multi-scale and collaborative (s. [17]).

REFERENCES

[1] I. Foster: *Service-Oriented Science*, Science, 308, pp. 814-817, 2005.

[2] B. Devlin, J. Gray, B. Laing, G. Spix: *Scalability Terminology: Farm, Clones, Partitions, and Packs: RACS and RAPS*, Technical Report MS-TR-9985, Microsoft Research, 1999.

[3] G. Brataas, P. Hughes: *Exploring Architectural Scalability*, Proc. 4th Intern. Workshop on Software and Performance, pp. 125-129, 2004.

[4] V. Kumar, A. Grama, A. Gupta and G. Karpysis: *Introduction to Parallel Computing*, Benjamin Cummings, 1994.

[5] T.G. Kolda, M.R. Lewis and V. Torczon*:* "Optimization by Direct Search: New Perspectives on Some Classical and Modern Methods", *SIAM Review*, Vol. 45, Number 3, pp. 385-482., 2003.

[6] M. Wright: "Direct search methods: Once scorned, now respectable", *Proc. Dundee Biennial Conf. In Numerical Analysis*, Addison Wesley, pp. 191-208, 1995.

[7] T. Barth, B. Freisleben, M. Grauer, and F. Thilo: "A Scalable Algorithm for the Solution of Simulation-based Optimization Problems", *Proc. PDPTA 2000*, Las Vegas, pp. 469-475, 2000.

[8] M. Laguna and R. Marti: *Scatter Search - Methodology and Implementations in C*, Kluwer, 2003.

[9] R. Marti, M. Laguna, F. Glover: *Principles of Scatter* Search, European Journal of Operational Research, Vol. 169, Issue 2, Pages 359-372, 2006.

[10] P. Hough, T.G. Kolda, and V. Torczon: "Asynchronous Parallel Pattern Search for Nonlinear Optimization", *SIAM Journal of Scientific Computing*, 23(1), pp. 134-156, 2001.

[11] K. Schittkowski: *Nonlinear Programming Codes*, Springer, 1980.

[12] M. Grauer, T. Barth and F. Thilo: "Grid-based Computing for Multidisciplinary Analysis and Optimization" in *Proc. of the 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conf.*, Albany, 2004.

[13] H.-J. Diersch: *FEFLOW 5.1 – Finite Element Subsurface Flow & Transportation System, User's Manual*, WASY GmbH, Berlin, 2005.

[14] F. Thilo, U. Junghans, M. Grauer, S. Kaden, J. Hillebrandt: "Reducing Groundwater Management Costs by Parallel Simulation-based optimization", *Proc.*

Computing and Control in the Water Industry, CCWI 2005, Exeter, 2005.

[15] P. Antsaklus, X. Koutsoukos, J. Zaytoon: "On Hybrid Control of Complex Systems: A Survey", *Proc. Intern. Conf. on Automation of Mixed Processes: Dynamic Hybrid Systems*, pp. 1-8, Reims, 1998.

[16] N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovic, W. Su: *Myrinet: A Gigabit-per-Second Local Area Network*, IEEE Micro, v.15 n.1, p.29-36, 1995.

[17] Report to the President: *Computational Science: Ensuring Americas´s Competitiveness*, President´s Information Technology Advisory Committee, June 2005.