Parallel Implementation of Evolutionary Strategies on Heterogeneous Clusters with Load Balancing *

Juan Francisco Garamendi, Jose Luis Bosque

Escuela Superior de Ciencias Experimentales y Tecnologia
Universidad Rey Juan Carlos, 28.933 Madrid, Spain
jf.garamendi@alumnos.urjc.es, joseluis.bosque@urjc.es

Abstract

This paper presents a load balancing algorithm for a parallel implementation of an evolutionary strategy on heterogeneous clusters. Evolutionary strategies can efficiency solve a diverse set of optimization problems. Due to cluster heterogeneity and in order to improve the speedup of the parallel implementation a load balancing algorithm has been implemented. This load balancing algorithm takes into account cluster heterogeneity and it is based on an optimal intial distribution. This initial distribution is determined based on the cluster nodes' computational powers, that are dinamically measured in each slave node by an ad hoc *load-bechmark. The implementation presents very satisfactory parallelization results, both in performance and scalability and Super-linear speedup is reached for several tests configurations. Experimental results show excellent perfomence, increasing the improvements with the load balancing algorithm.*

1. Introduction

Evolutionary algorithms are heuristic based search techniques that have generated a great deal of interest in the last years. Although there is a number of paradigms, such as genetic algorithms, in this paper we will discuss only the *evolutionary strategies* paradigm and its parallel implementation. This paradigm has been successfully applied to solve a large number of optimization problems, including task scheduling in distributed systems [7], image processing [16] or calculation of robot trajectories [11].

*This work has been partially funded by the Spanish Commission for Science and Technology (grant TIC2003-08933-C02).

An evolutionary strategy program may take a long execution time for large size problems. Traditionally, the population size of the algorithm is 50-100 individuals. This amount, even adequate for moderately size problems, is small for current and real applications and an increase of population size yields to a linear increase in the execution time. On the other hand, the evolutionary algorithms are ideally suited for data parallel implementations [13, 14].

Cluster and grid computing are arising a lot of interest, thanks to their excellent price/performance ratio, scalability, fault tolerance and flexibility features [1, 2]. Cluster implementations are becoming nowadays a feasible solution for computer applications with high level of computational power demand. An important issue to take into account in clusters is the system heterogeneity. There are to main reasons that have stimulated the introduction of heterogeneity in clusters. The first one is the clusters flexibility and reconfigurability, that allow to change old nodes or to introduce new nodes very easily. On the other hand, the tremendous improvement rate of PC's and workstations, following the Moore's Law, makes that newly introduced nodes will have very different computing capabilities.

Equilibrating the amount of work assigned to each node in a cluster is a complex problem, even more than for other kinds of parallel systems. Even though load balancing has received a considerable amount of interest, it is still not definitely solved [3, 21, 22]. Nevertheless, this problem is central for minimizing the applications' response time and optimizing the exploitation of resources, avoiding overloading some processors while others are idling. Clusters present additional challenges, since they can easily become heterogeneous, requiring load distributions that take into consideration each node's computational features [15]. Determining the optimal location for processes in a heterogeneous

1-4244-0054-6/06/$20.00

system is a complicated issue but it is necessary to provide efficient, scalable, low overhead and general-purpose strategies capable of handling heterogeneity.

This paper presents a simple, centralized, global and efficient load balancing algorithm for a parallel implementation of an evolutionary strategy on heterogeneous clusters. Due to heterogeneity a new workload index has been proposed. The algorithm takes into consideration both the different node capabilities and the complete system state. It is based on an initial workload distribution where the population is distributed among all of the computational nodes according to their *actual computational power*, estimated from its computational power and of external tasks assignment.

The rest of the paper is organized as follows: section 2 presents previous work on parallel implementations. A description of the parallel implementation and the load balancing algorithm are discussed in section 3. Section 4 shows the experimental results and finally section 5 presents some conclusions and future work.

2 Previous and Related Work

Traditionally there are two kinds of parallel evolutionary strategies, the master-slave model and the island model [8]. In the master-slave model, the master stores the population, executes the mutations and transfer the individuals to the slaves that perform the evaluation of the function [9]. This approach is used when the evaluation function operator is the most cost operation in the algorithm.

In the island model, the population is partitioned equally among the slave processors, and each processor runs the algorithm on its local population [6]. Then the master process selects the best solution among the slaves' results. In the island model the network traffic is not heavy because iterations of the sequential algorithm are performed in each processor [8].

In evolutionary strategies there are not migration of individuals from one slave to other, like happens in genetics algorithms [4, 8], yielding a null communication overhead along slave nodes. Therefore parallel evolutionary strategies with an island model are a suitable model for clusters.

Van Velduizen et al. ([18]) suggest the use of general techniques for load balancing. [12] compares two load balancing strategies: a static scheme in which workload is distributed using a Round Robin policy at compilation time; and a dynamic scheme where the tasks are sorted in downward order and in each iteration a new task is assigned to the least loaded node. On the other hand [10] presents a dynamic load balancing algorithm for a genetic algorithm. This algorithm is based on the migration of individuals among the cluster nodes.

None of the previous parallel implementations, neither genetic algorithms nor evolutionary strategies, take into account the cluster heterogeneity. This paper presents a load balanced parallel implementation of evolutionary strategies on heterogeneous clusters.

3 Parallel Implementation for Heterogeneous Systems

3.1 Motivation

Evolutionary strategies have a number of features that make suitable to yield good performance benefits with a parallel implementation on a PC cluster. These features are the following:

1. Since there are no data dependencies among the individuals of a population, the exploitation of data parallelism can be done just dividing the total workload (the number of individuals) among N independent nodes, without synchronization points.

2. It is a fine granularity problem, therefore a great parallelism degree can be achieved. This improves the algorithm's scalability when the number of nodes is increased and allows to keep the load balance even on a heterogeneous cluster.

3. Unlike genetic algorithms, the network traffic is not heavy because there are not migrations of individuals among different slave nodes. Therefore the communication overhead is very low.

3.2 Master-Slave Strategy

The parallel strategy is based on a farm model [20] in which a *master* process distributes the data to be dealt with upon a set of *slave* processes. After receiving its population, each *slave* can start computing the algorithm and sends back the partial results to the *master*, once it has finished processing them.

On heterogeneous clusters the population distribution should be done proportional to the nodes' computational capabilities. Otherwise if the workload were evenly distributed, the application response time would be determined by the slowest node, which would finish its work last. Additionally, in a non-dedicated cluster the node capabilities depends on both the different node computational attributes as well as the different amounts of external tasks assignments.

In order to introduce the load balancing algorithm a multiphase approach is used, where there are two

tightly interleaved phases separated by a synchronization point [19]. In this case the first phase is devoted to the load balancing algorithm and the second phase involves the evolutionary strategy itself.

- *Initial balanced distribution*: in this stage the workload is distributed among all of the cluster's nodes according to their *actual computational power*. The computational power is estimated from the nodes' computational capabilities and from the external tasks. Each slave process runs a *load-benchmark* to determine its own computational power. Based on the load-benchmark results a computational power index is calculated on each slave process and this value is sent to the master process. When the master collects the computational power indexes of all of the slave nodes, it calculates the total cluster computational power and the best distribution of the population (see section 3.3), and sends a message to every cluster nodes with the number of individuals that it has to evolve.

- *Evolutionary strategy*: Afterwords receiving its population the slave process evolves it, a finite and fixed number of generations. When the slave process reaches the number of steps it comes back the better individual of its population to the master. Then the master evaluates all these individuals and decides if the algorithm should go on or should stop.

Now, the detailed algorithm is presented:

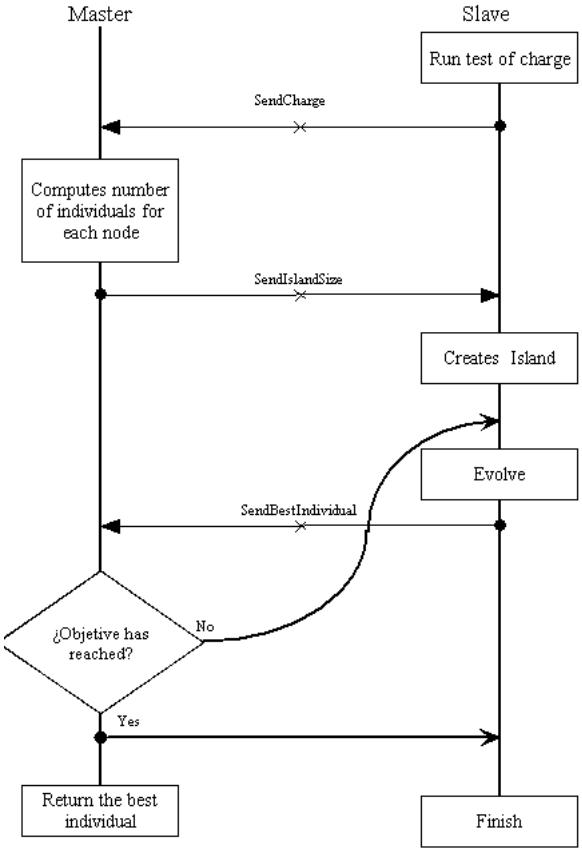


Figure 1. General overview of the process communication

Master

1. Receives the nodes' computational power.
2. Computes the number of individuals that each node must create.
3. Sends to each node the number of individuals that must create and evolve.
4. Waits until nodes finish their work.
5. Collects from the nodes the best individual of each population.
6. Computes the objective function and verify if the objective value has reached. In affirmative case, the algorithm is stopped and the solution is returned, in other case the algorithm goes on.
7. If the algorithm must go on, the master informs to the slaves nodes.

Slave Node

1. Runs the load-benchmark and sends the result to the master.
2. Receives the amount of individuals of its population.
3. Evolves the algorithm over its population a finite number of iterations.
4. Returns to the master the best individual.
5. If the master decides going on with the algorithm, the slave repeats the steps 3,4 and 5.

Figure 1 shows a flow diagram that includes the communication scheme followed by the processes involved in the application.

3.3 The Load-Bechmark for the Initial Distribution

As it has been explained above the load balancing algorithm is based on an optimal initial distribution. And this initial distribution is based on the nodes' computational power, denoted by P_i, and on the total cluster computational power ($P_T = \sum_{i=1}^{N} P_i$). The computational power of a node N_i is estimated based on the execution of a *load-benchmark*.

The load-becnhmark is executed in each of the cluster nodes and is based on running the evolutionary strategy algorithm during a fixed time span, equal for all of the nodes. During this fixed time span the load-benchmark measures the amount of iterations performed by a node with a fixed number of individuals. With these parameters the load-benchmark calculates the average time (t_i) that each node spent to execute one iteration of one individual. Therefore the load-benchmark measures dynamically the node's computational power which is the inverse of this average time: $P_i = \frac{1}{t_i}$.

Based on these parameters the master process determines the optimal distribution for the current cluster state. If S is the population size and P_T is the cluster's total computational power, each slave N_i with computational power P_i will receive S_i individuals, according to the following expression:

$$S_i = \frac{S}{P_T} \cdot P_i \tag{1}$$

This expression is determined by two conditions. First, all of the cluster nodes must take the same response time, in order to avoid idle time. Second, the sum of all of the node populations must be equal to the total population size:

$$\begin{cases} \widetilde{T} = \frac{S_i}{P_i} = \frac{S}{P_T} \\ \sum_{i=1}^{N} S_i = S \end{cases}$$

3.4 Evolutionary Strategy

The evolutionary algorithm has been implemented to find the global minimum of both the Rastrigin and the Ackley functions, in their ten variables version [8]. These functions have a known global minimum value (0 at $f(0,..,0)$) and multiplies local minimums, which difficulties the search of the global minimum.

The evolutionary strategy implemented is the following:

1. A population P of μ random solutions is created. Each solution is called $\overline{X}_j$, with $j \in \{1,..,\mu\}$.
2. If the stop condition is not reached, a vector $\overline{Z} = (z_1,..,z_n)$ is created for each solution j of P. The component z_i is obtained from normal distribution: $\overline{Z}_j \sim N(0,\sigma_j^2)\ \forall j \in \{1,..,\mu\}$
3. A new individual $\overline{Y}_j$ is stemmed from $\overline{X}_j$ and $\overline{Z}_j$: $\overline{Y}_j = \overline{X}_j + \overline{Z}_j$
4. If $f(\overline{Y}_j) < f(\overline{X}_j)$, then $\overline{X}_j$ is replaced by $\overline{Y}_j$. Update every Ps_j.
5. To repeat the steps 2, 3 and 4 k-times.
6. To evaluate each solution $\overline{X}_j$. If stop condition is reached, return the solution. In other case, change every σ_j according to:
   - $\sigma_j^{new} = \sigma_j \cdot c$ if $Ps_j < threshold\_for\_\sigma$
   - $\sigma_j^{new} = \frac{\sigma_j}{c}$ if $Ps_j > threshold\_for\_\sigma$

   Where Ps_j is the number of times that σ_j causes changes in $\overline{X}_j$, divided by k, and a typical value of c is $\frac{1}{5}$.
7. Set every Ps_j to zero and return to the step 2.

4 Experimental Results

In order to evaluate the performance improvements of the parallel implementation described above, a set of tests were performed on a heterogeneous cluster. Three objectives were pursued in the tests presented here:

1. To verify the feasibility of applying a parallel solution based on a PC cluster to two different optimization problems based on evolutionary strategies: the minimization of the Rastrigin and the Ackley functions. The performance improvements of the parallel version with respect to the sequential version, with different population size will be analyzed.
2. To evaluate the improvements achieved by the load balancing algorithm on a heterogeneous cluster. A number of experiments have been performed for testing the behavior of the load balancing algorithm, comparing the application total execution time in both cases with and without load balancing algorithm.
3. The load balancing algorithm is based on the measures of the nodes' computational power estimated from an *ad hoc* load-benchmark. The time executing the load-benchmark has an important impact on the load balancing algorithm accuracy

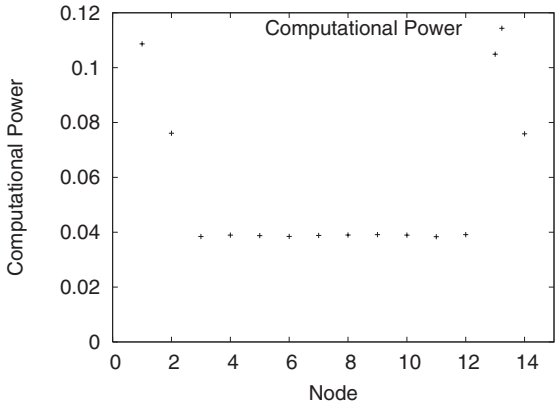


Figure 2. Nodes' computational power of the cluster

and therefore on the response time of the parallel version. The last objective of the tests were to improve the load balancing algorithm determining the most suitable time span for the load-benchmark execution.

4.1 Experimental Setup

The tests were performed over a 14 node PC cluster connected through a Gigabit-Ethernet network, called *Helios*. Each node is a 1.8 GHz AMD XP processor with DDRRAM 512 MB of main memory and 512 KB of cache memory. The PC's operating system is Red Hat Linux 8.0 (v. 2.4.22). The application was developed using C++ language, GNU tools and the MPI/LAM 7.0.4 library [5, 17].

In order to make the cluster heterogeneous, extra load was introduced by executing different number of local tasks in each node. Consequently, differently loaded nodes yielded different response times, since in computer systems tasks share the processor time using a Round Robin scheduling policy. Figure 2 presents the nodes' computational power, experimentally measured for this cluster.

The tests were performed over two different optimization problems based on evolutionary strategies: the minimization of the Rastrigin and the Ackley functions. For each of these problems several tests were performed, increasing the number of individuals, with the algorithm's parameters presented in table 1.

4.2 Results with Homogeneous Cluster

This section presents a number of tests to compare the performance of the parallel and sequential implementations of the evolutionary strategies on the homogeneous cluster. Figure 3 shows the evolution of the response time, given in seconds, of the parallel and the sequential versions increasing the number of individuals for both Ackley (3(a)) and Rastrigin (3(b)) functions.

Algorithm Parameter	Value
Solution error allowed	$5 * 10^{-5}$
*Threshold_for_*σ	0.9
Change factor, c	0.2

Table 1. Parameters of the algorithm used in the experimental results.

Figures 4(a) and 4(b) show the speedup achieved in the cluster for the Ackley and the Rastrigin functions, respectively.

Looking at these figures it has to be noted that the parallel implementation yields a substantial reduction in the application response times, increasing the improvements with the population size. The speedup and the efficiency figures are very close to their maximum achievable value and in some cases outperforms it, yielding a super-linear speedup. This behavior can be explained by the effect of the increment of the cache memory size on the whole system.

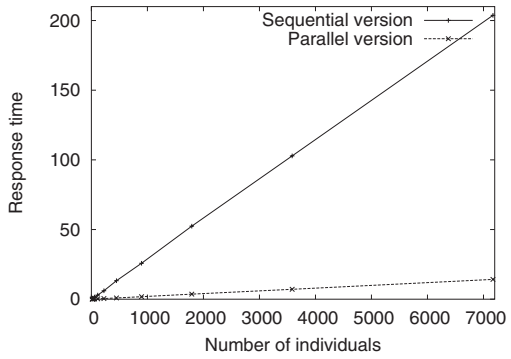
Additionally, it is possible to deduce that this approach shows a high degree of scalability. The maximum values for speedup and efficiency remain almost constant for any cluster configuration when the number of individuals is increased. This is due to the low communication overheads which remains constant with respect to the problem size.

4.3 Results with Heterogeneous Cluster

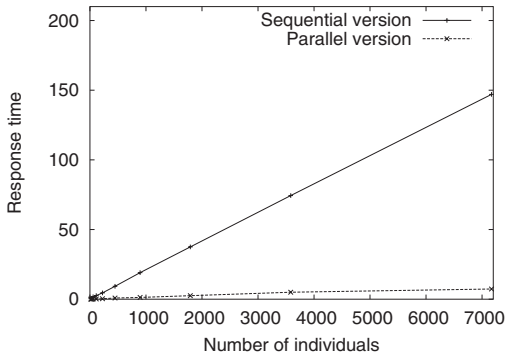
For these experiments two nodes of the cluster were loaded with one additional task and another two nodes were loaded with two additional task, yielding on a heterogeneous environment (from the performance point of view) with the nodes' computational power presented in figure 2. This experiment makes a detailed comparison of the heterogeneous cluster performance with and without load balancing algorithm. The workload distribution has to take into account the differences on the nodes' computational power.

Figure 5 presents the application response times achieved for both Ackley (5(a)) and Rastrigin (5(b))

Speedup is defined as $S_n = T_1/T_n$, where T_1 is the execution time of the algorithm running on only one processor and T_n is the execution time of the parallel algorithm carried out over n processors.

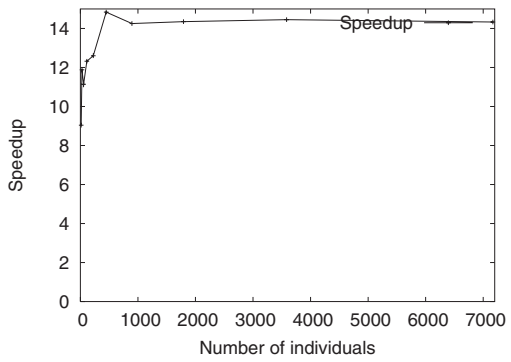


(a) Ackley function.

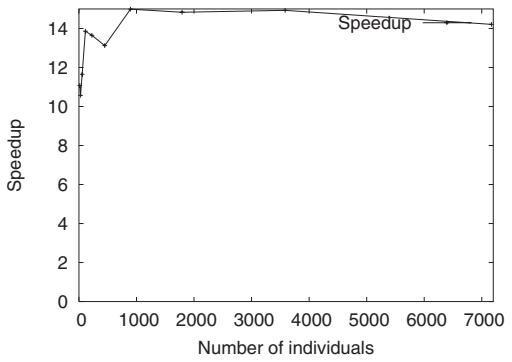


(b) Rastrigin function.

Figure 3. Response times in seconds.



(a) Ackley.



(b) Rastrigin.

Figure 4. Speedup for Ackley and Rastrigin functions.

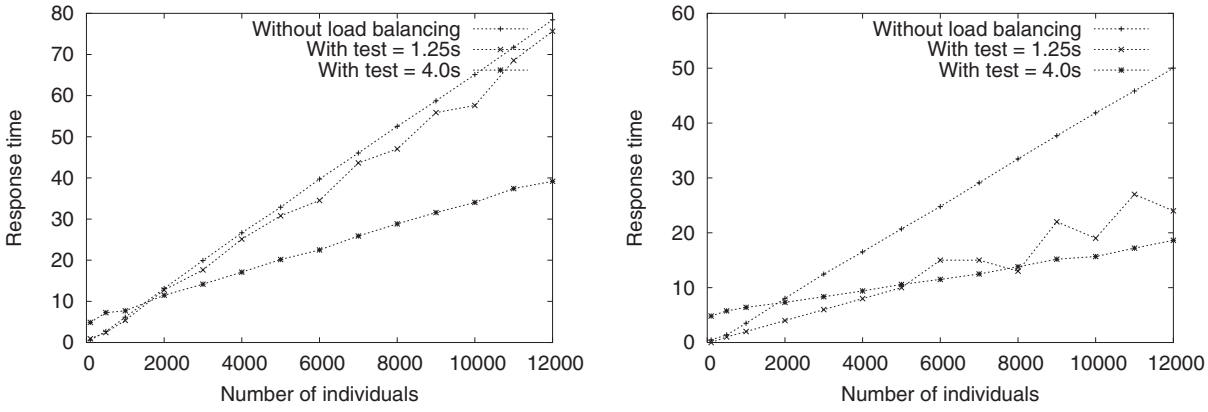
functions, with and without load balancing algorithm. As expected, the load balancing algorithm achieves significant improvements, reaching better results when the number of individuals is increased. A possible explanation for this behavior is that in those cases the load balancing algorithm has more possibilities to distribute the total workload among the different nodes, and reaches a better distribution, proportional to the nodes' computational power.

A major consequence we can point out of these results is that the time span of the load-benchmark has a strong impact on the load balancing algorithm accuracy. If the benchmark time span is too short, the initial distribution of the individuals is not accurate enough to handle the system heterogeneity. On the other hand, if the benchmark time span is too large it has a noticeable impact on the application response time, and therefore the performance benefit achieved by the load balancing algorithm does not outperforms this time. This is the case shown in figure 5, when the population size is small, and the load-benchmark time is larger than the response application time. Therefore, in these cases the response time with load balancing is larger than without it.

In order to determine the influence of the benchmark execution time on the distribution accuracy a theoretical and optimal distribution has been estimated for the heterogeneous cluster.

Figure 6 shows the evolution of the relative error of the load balancing algorithm distribution with respect to the theoretical distribution for both the Ackley and Rastrigin functions. From theses results it has to be noted the following conclusions:

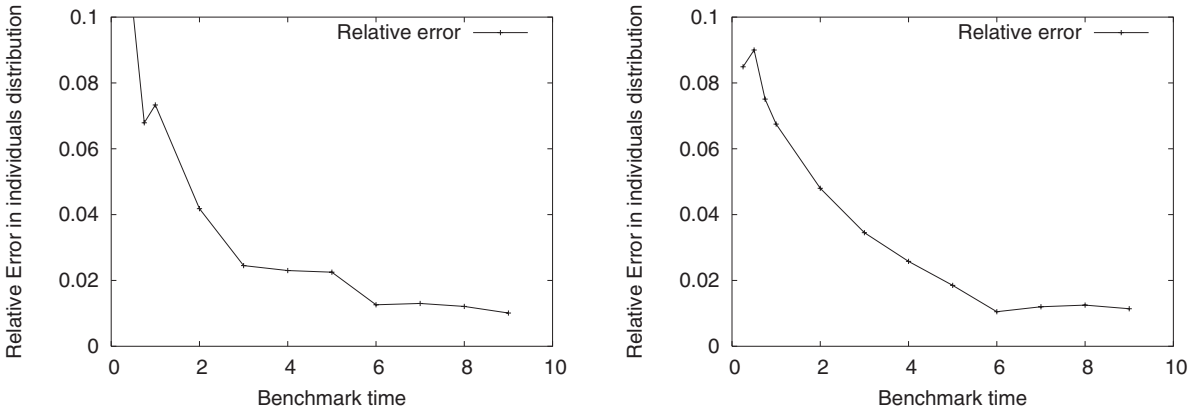
- The benchmark execution time has a strong im-



(a) Ackley function

(b) Rastrigin function.

Figure 5. Evolution of the response times with and without load balancing algorithm.



(a) Ackley function

(b) Rastrigin function.

Figure 6. Influence of the benchmark execution time in the distribution accuracy.

pact on the accuracy of the initial distribution.

- There is a maximum value of this parameter that optimize both the workload distribution and the response application time. This value depends on the P_T and on the cluster heterogeneity.

5 Conclusions and future work

In this paper, a load balancing algorithm for a parallel implementation of an evolutionary strategy has been presented. A remarkable feature of the implemented algorithm is its low communication overhead, which results in very satisfactory parallelization results, both in performance and scalability. Super-linear speedup is reached for several tests configurations. Therefore a major conclusion is that the parallel approach on a cluster is a very good solution to reduce the response time of this kind of applications.

An important issue to take into account is system heterogeneity, originated both by different node computational attributes as well as by different amounts of external load assignments. Therefore a load balancing algorithm has been introduced. The algorithm is centralized, global and it is based on an initial distribution which would be optimal if the nodes' computational power remain constant along the application execution. The initial distribution is based on the cluster total computational power and on the nodes' compu-

tational powers, measured in real time.

All of the experiments performed here have shown that using the load balancing algorithm has resulted in large reductions in the response time. These improvements become more and more relevant as the number of individuals is increased. The overhead introduced by this method depends strongly on the load-benchmark execution time, but on the other hand, this has an important impact on the distribution accuracy.

Future work includes the development of a dynamic and distributed load balancing algorithm in order to compare both strategies. Additionally, the development of more systematic and precise methods for estimating the relative node computational power. Finally we plan to migrate this application to a grid architecture.

References

[1] G. Bell and J. Gray. What's next in high-performance computing? *Communications of the ACM*, 45(2):91–95, 2002.

[2] R. Buyya, editor. *High Performance Cluster Computing, Volume 1: Architecture and Systems*. Prentice-Hall PTR, 1999.

[3] S. K. Das, D. J. Harvey, and R. Biswas. Parallel processing of adaptive meshes with load balancing. *IEEE Trans. on Parallel and Distributed Systems*, (12):1269–1280, 2001.

[4] F. de Toro, J. Ortega, and B. Paechter. Parallel Single Front Genetic Algorithm: Performance Analysis in a cluster system. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)*. IEEE Computer Society, 2003.

[5] M. Forum. A message-passing interface standard. 1995. http://www.mpi-forum.org.

[6] A. Gupta, G. W. Greenwood, and R. Munnangi. Parallel implementations of evolutionary strategies, may 1995.

[7] A. K. Gupta and G. W. Greenwood. Applications of evolutionary strategies to fine grained task scheduling, May 20 1996.

[8] T. Hiroyasu, M. Miki, and Y. Tanimura. The differences of parallel efficiency between the two models of parallel genetic algorithms on pc cluster systems. In *The Fourth International Conference on High-Performance Computing in the Asia-Pacific Region*, volume Volume 2, pages 945–948, Beijing, China, May 2000.

[9] M. A. Ismail. Parallel genetic algorithms (pgas): Master slave paradigm aproach using MPI. *IEEE*, 2004.

[10] N. Neves, A.-T. Nguyen, and E. L. Torres. A study of a non-linear optimization problem using a distributed genetic algorithm. In A. Bojanczyk, editor, *Proceedings of the 1996 International Conference on Parallel Processing. Vol. II Algorithms and Applications, ICPP'96 (Bloomingdale, Illinois, August 12-16, 1996)*, pages 29–36, Los Alamitos-Washington-Brussels-Tokyo, 1996. International Association for Computers and Communications, Pennsylvania State University, IEEE Computer Society Press.

[11] M. Ortmann and W. Weber. Multi-Criterion Optimization of Robot Trajectories with Evolutionary Strategies. In *Proceedings of the 2001 Genetic and Evolutionary Computation Conference. Late-Breaking Papers*, pages 310–316, San Francisco, California, July 2001. Morgan Kaufmann Publishers.

[12] M. Oussaidène, B. Chopard, O. V. Pictet, and M. Tomassini. Parallel genetic programming and its application to trading model induction. *Parallel Computing*, 23(8):1183–1198, 1997.

[13] V. P. Plagianakos and M. N. Vrahatis. Parallel evolutionary training algorithms for "hardware-friendly" neural networks. *Natural Computing*, 1(2-3):307–322, 2002.

[14] S. M. Sait, S. Sanaullah, A. M. Zaidi, and M. I. Ali. Comparative evaluation of parallelization strategies for evolutionary and stochastic heuristics. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 921–922, New York, NY, USA, 2005. ACM Press.

[15] B. Schnor, S. Petri, R. Oleyniczak, and H. Langendörfer. Scheduling of Parallel Applications on Heterogeneous Workstation Clusters. In *Proceedings of the ISCA 9th International Conference on Parallel and Distributed Computing Systems*, volume 1, pages 330–337, Sept. 1996.

[16] H. Shimodaira. Strategies for optimizing image processing by genetic and evolutionary computation. In *12th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2000), 13-15 November 2000, Vancouver, BC, Canada*, pages 151–, 2000.

[17] J. M. Squyres, K. L. Meyer, M. McNally, and A. Lumsdaine. LAM/MPI user guide. 1998.

[18] D. A. Van Veldhuizen, J. B. Zydallis, and G. B. Lamont. Considerations in Engineering Parallel Multiobjective Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, 7(2):144–173, Apr 2003.

[19] J. Watts, M. Rieffel, and S. Taylor. A load balancing technique for multiphase computations. *Proceedings of Hisg Performance Computing*, pages 15–20, 1997.

[20] B. Wilkinson and M. Allen. *Parallel Programing*. Prentice-Hall PTR, 1999.

[21] L. Xiao, S. Chen, and X. Zhang. Dynamic cluster resource allocations for jobs with known and unknown memory demands. *IEEE Trans. on Parallel and Distributed Systems*, 13(3):223–240, Mar. 2002.

[22] A. Y. Zomaya and Y.-H. Teh. Observations on using genetic algorithms for dynamic load-balancing. *IEEE Trans. on Parallel and Distributed Systems*, 12(9):899–911, 2001.