ARTICLE IN PRESS

Available online at www.sciencedirect.com

SCIENCE DIRECT®

ELSEVIER

J. Parallel Distrib. Comput. ▮▮▮ (▮▮▮▮) ▮▮▮–▮▮▮

Journal of Parallel and Distributed Computing

www.elsevier.com/locate/jpdc

A general framework to understand parallel performance in heterogeneous clusters: analysis of a new adaptive parallel genetic algorithm

Victor E. Bazterra[a,b,1], Martin Cuma[a,b,2], Marta B. Ferraro[a,b,3], Julio C. Facelli[a,b,*]

[a] *Departamento de Física, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Ciudad Universitaria, Pab. I (1428), Buenos Aires, Argentina*

[b] *Center for High Performance Computing, University of Utah, 155 South 1452 East Rm 405, Salt Lake City, UT 84112-0190, USA*

Received 22 September 2003; received in revised form 2 August 2004

Abstract

This paper presents a general model to define, measure and predict the efficiency of applications running on heterogeneous parallel computer systems. Using this framework, it is possible to understand the influence that the heterogeneity of the hardware has on the efficiency of an algorithm. This methodology is used to compare an existing parallel genetic algorithm with a new adaptive parallel model. All the performance measurements were taken in a loosely coupled cluster of processors.

Keywords: Heterogeneous parallel environment; Parallel genetic algorithms; Performance analysis

1. Introduction

When developing parallel applications, performance measurements of a given algorithm on a computational platform are of great importance. Through these measures, it is possible to understand how the application interacts with the computer system exposing not only the algorithm's limitations, but also suggesting possible improvements that may lead to a better utilization of the underlying hardware.

Numerous publications have discussed the importance of the parallel performance measurement and there are a number of different metrics defined for this purpose [8]. When evaluating the performance of any parallel application the most important parameter is the elapsed time (wall time) which measures how long a user has to wait for the program to complete its execution. From the measurements of the elapsed time, other metrics can be defined with the aim of measuring more subtle features that may affect its parallel performance. Common machine independent performance measurements are the speedup [4], the scaled speedup (where the problem size increases with the number of processors) [11] and the serial fraction [13]. For parallel applications, the elapsed time accounts not only for the computational work but also for any contributions arising from synchronization, communication and I/O. These factors have been identified as the barriers to obtain perfect scalability in homogeneous parallel systems. When the algorithm is used in a heterogeneous parallel system, the diversity of the hardware adds additional constrains that degrade even more the scalability of the application. The purpose of this paper is to present a framework in which this degradation can be quantified.

In the last 10 years, heterogeneous distributed computer resources have become a reliable and low cost solution to the problem of massive computation. For example, loosely

* Corresponding author. Center for High Performance Computing, University of Utah, 155 South 1452 East Rm 405, Salt Lake City, UT 84112-0190, USA. Fax: +1 801 585 5366.

E-mail address: julio.facelli@utah.edu (J.C. Facelli).

[1] VEB is the recipient of a Fellowship from CONICET at the Universities of Buenos Aires and Utah.

[2] Scientific Applications Programmer at CHPC.

[3] MBF is a Member of Carrera del Investigador del CONICET at the University of Buenos Aires.

0743-7315/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2004.09.011

coupled clusters of processors [5] are becoming increasingly popular in most high performance computer centers. More recently, developments in GRID infrastructure [5,10] are generating the protocols necessary to share computational resources between geographically separated centers. In this context, many different computer systems with diverse number of processors, speeds, communication switches, I/O subsystem and OSs are loosely connected through high latency networks making the diversity of resources an important factor in the overall performance of parallel programs executing in these environments. Unfortunately, most of the existing performance metrics have been applied to homogeneous parallel environments. Only a few reports have been published in the literature to address parallel performance in heterogeneous environments [14,16–18,20]. These reports have introduced the concept of heterogeneity, defined by the differential loads on a homogeneous group of machines [21], or by the differences between the resources associated to the participating machines [3].

The main objective of this paper is to describe a general framework to understand parallel efficiency of heterogeneous systems in which, as it is common practice in scientific computing, the nodes are dedicated to a particular job running one program. Our performance evaluation framework follows the concepts proposed in Ref. [9], depending on two models, one for the heterogeneous computer system and the other for the algorithm running in that environment. But in contrast with the work in Ref. [9], which focus on the sequential performance of heterogeneous architectures, we focus on parallel systems with homogeneous architectures, i.e. computational clusters, for which the diversity arises from the different clock rate or performance of the different nodes in the cluster.

In Section 2, general models for a heterogeneous computer system and for a parallel algorithm are introduced. Both models are combined in Section 3 to define performance metrics for heterogeneous computers. The concept of idle time is introduced in Section 4 to take into account the effects of interprocessor communication and synchronization. In the Section 5, we describe a new Parallel Adaptive Genetic Algorithm which uses a global parallelization scheme [2] designed to maximize load balance in heterogeneous environment by dynamically adjusting the processor loads. We demonstrate that this algorithm is a viable alternative to the Cantú-Paz algorithm [6,7] to overcome the performance limitations predicted by our model when it is used in a heterogeneous system. In Section 6, the performance measurement of the new algorithm in a homogeneous and a heterogeneous environment are compared showing that the achieved measurements agree well with the predictions of our model.

2. Heterogeneous computer and algorithm models

A heterogeneous computer system composed by a collection of n-processors with different speeds and interconnected by a network will be described by a configuration C_n given by $\{v_1, \ldots, v_n\}$ where v_i is the relative speed of the ith processor, i.e. the number of operations that a processor can complete in a unit of time. Following Ref. [9] we assume that the processors are *nondecomposable*, that is, multiple algorithms cannot be executed on the same processor simultaneously. This assumption is based on the fact that in general, heterogeneous systems are operated as capacity computing resources in which some kind of scheduler assigns processors to different jobs according to prescribed rules that try to balance resource allocation while maximizing the resource utilization [12].

The network communications can be specified by the pair-wise latency and the bandwidth between processors [9]. Since the network performance depends on many factors and it is usually quite difficult to measure, in our model we focus on the effects of the heterogeneity of the hardware and use the concept of idle time T_{id} to take into account these factors. T_{id} is defined as the amount of time that a processor is not executing an instruction of the running algorithm. This quantifies the loss of performance due to the combined effects of network latency and synchronization delays.

We will represent a parallel algorithm requiring K independent computational operations running on n processors by $A_n(K)$. When this algorithm runs in the system C_n, it distributes either by static or dynamic mechanisms the K operations among the n available processors. Under these circumstances, the elapsed time of a parallel program will depend on the assigned processors for the job and the algorithm strategy to distribute K operations in that particular group of processors. In this paper, we present a stochastic model of performance that takes into account the different strategies that an algorithm can present under different system loads. The distribution of K operations on the parallel system is defined by the probabilities set $\{p_1, \ldots, p_n\}$, which gives the frequency at which the algorithm assigns operations to the processor i.

3. Parallel performance metrics

3.1. Total and parallel elapsed time

In addition to the common factors that affect performance in homogeneous environments, the parallel performance of an algorithm on a heterogeneous computer system depends on how well the distribution of the work required by the algorithm match the ability of the processors to perform these tasks. We will designate as a parallel system the combination of an heterogeneous computer system and a parallel algorithm $\Gamma_n = \{C_n; A_n(K)\}$.

For simplicity, we will consider first the performance model of an ideal algorithm for which the communication and synchronization times are negligible. The effects of these delays will be included later using the concept of idle time. This approach allows to clearly separate the performance

issues associated with heterogeneity from those due to network and synchronization delays.

For an algorithm $A_n(K)$ running C_n, the average elapsed time, T_i, spent on the ith processor of C_n is

$$T_i = Kp_i/v_i, \tag{1}$$

where p_i is the frequency at which the algorithm assigns operations to the processor i and v_i is its relative speed.

Always there is a processor of C_n that takes the longest time to run the Kp_i operations assigned to it. Therefore to complete all the steps in $A_n(K)$, we have to wait until this last processor finishes its tasks. The time used by this processor determines the elapsed time necessary to run $A_n(K)$ on C_n. Therefore for the parallel system $\Gamma_n = \{C_n; A_n(K)\}$, the elapsed time is given by

$$T_n = \max\ T_i = K\ \max\{p_i/v_i\}, \tag{2}$$

while the total computer (CPU) time, T, used by the algorithm in all the nodes that have been assigned to the job is the sum of all T_i

$$T = \sum_i T_i = K \sum_i p_i/v_i. \tag{3}$$

The most important factor determining the performance of an algorithm in heterogeneous parallel systems is how efficiently the algorithm uses the available resources. The distribution set $\{p_1, \ldots, p_n\}$ determines this efficiency. The following two propositions give the distributions that maximize the efficiency of the algorithm and determine the relationship between the elapsed time and the total computer time for an arbitrary parallelization strategy.

Proposition 1. *Given a parallel system C_n and a parallel algorithm $A_n(K)$ with a probability set $p_i = v_i/\sum_k v_k$, for any other algorithm $B_n(K)$ with a different probability set, the execution of the algorithm $B_n(K)$ requires a longer parallel elapsed time than $A_n(K)$ (see proof in appendix).*

Proposition 2. *Given a parallel computer C_n and a parallel algorithm $A_n(K)$, it is always true that $T \geqslant T_n \geqslant T/n$. The equalities $T = T_n$ and $T_n = T/n$ are true when $p_i = \delta_{ij}$ for a given $1 \leqslant j \leqslant n$ and when $p_i = v_i/\sum_k v_k$, respectively (see proof in appendix).*

In other words, Proposition 1 states that the most efficient algorithm has a probability set proportional to the speed of processors, $p_i \propto v_i$. In this way, the algorithm compensates for the heterogeneity of the parallel system by sending more operations to the faster processors. For example, given a parallel computer C_n and a parallel algorithm $A_n(K)$ with a probability set $p_i = v_i/\sum_k v_k$, it is easy to prove that the parallel elapsed time is $T_n = K/\sum_i v_i$, which corresponds to executing the algorithm on the parallel computer with a compute capacity equivalent to a serial computer with the effective speed given by the sum of speeds of all the processors. This corresponds to the case of perfect linear scaling for homogeneous parallel systems.

The second proposition states that in the optimal distribution the total computer time is equally distributed on all the processors, i.e. all the processors run the same amount of time regardless their different speeds. In the other extreme when only one processor is used, the total computer time is equal to the parallel elapsed time.

3.2. Speedup and efficiency in a heterogeneous system

A very common performance metric in homogeneous parallel systems is the *speedup* [1,3,4,8,9], which measures the decrease in the elapsed time between the parallel and serial execution of the program. In the case of heterogeneous systems, it is possible to extend this definition comparing the elapsed time of the parallel execution with the serial execution on the fastest processor. Given a parallel computer C_n and algorithm $A_n(K)$ the speedup is the ratio between the elapsed time ($T_{1,\min}$) of the best serial version of $A_n(K)$ on the fastest processor in C_n and the parallel elapsed time [1]:

$$s = \frac{T_{1,\min}}{T_n}. \tag{4}$$

This definition coincides with the definition of speedup in homogeneous systems, when all the processors have the same speed. Other definitions are possible based on the average processor speed of the available processors, but the selection of the base timing does not change the results presented here. Considering that the best serial version of the algorithm $A_n(K)$ has to complete K operations in the faster processor, taking an elapsed time of $T_{1,\min} = K/\max\{v_i\}$, the speedup for a heterogeneous system can be written as

$$s = \frac{1}{\max\{v_i\}\max\{p_i/v_i\}}. \tag{5}$$

The maximal and minimal values for the speedup are given by the following proposition.

Proposition 3. *Given any parallel computer C_n and any parallel algorithm $A_n(K)$, it is always true that*

$$\frac{\min\{v_j\}}{\max\{v_i\}} \leqslant s \leqslant \frac{1}{\max\{v_i\}} \sum_k v_k$$

and $s = \frac{\min\{v_j\}}{\max\{v_i\}}$ if $p_i = \delta_{ij}$ where j is the index of the slowest processor and $s = \frac{1}{\max\{v_i\}} \sum_k v_k$ if $p_i = v_i/\sum_k v_k$, i.e. for the most efficient probability set (see proof in appendix).

The most important consequence of this proposition is that there is a maximum speedup that can be achieved in a heterogeneous environment, even when no delays are added by communication or synchronization operations. This maximum speed up depends only on the distribution of the C_n

processor speeds. We can express this maximum possible speedup as

$$s_{\max} = \frac{1}{\max\{v_i\}} \sum_k v_k. \tag{6}$$

All the performance information of an application can be derived by comparing the actual speedup to the value of $s_{\max}$. This comparison can be done using the ratio between these two speedups:

$$e = \frac{s}{s_{\max}} = \frac{1}{\max\{p_i/v_i\} \sum_i v_i}. \tag{7}$$

In homogeneous systems, the quantity defined in Eq. (7) is known as the efficiency of the algorithm, because in these systems the number of processors determines the maximal speedup. Therefore, we propose Eq. (7) as a general definition of the efficiency for both heterogeneous and homogeneous systems. However, neither the speedup nor the efficiency values in heterogeneous systems are as intuitive as in the case of homogeneous systems. To make the comparison between the actual speedup and the maximum obtainable speedups more intuitive is useful to introduce the concept of effective number of processors, given by

$$n_{\text{eff}} = \frac{T}{T_n} = \frac{1}{\max\{p_i/v_i\}} \sum_k p_k/v_k. \tag{8}$$

The effective number of processors can be interpreted as a measure of how many processors in C_n have been used efficiently by the algorithm $A_n(K)$, after taking into account that this effectiveness is already reduced by the heterogeneity of the system. From Proposition 2, follows that n_{eff} has the following properties:

(i) $1 \leqslant n_{\text{eff}} \leqslant n$,
(ii) $n_{\text{eff}} = 1$ when $p_i = \delta_{ij}$ for a given $1 \leqslant j \leqslant n$,
(iii) $n_{\text{eff}} = n$ when $p_i = v_i / \sum v_k$.

That is, n_{eff} has a value between 1 and the number of processors, being equal to one if and only if one processor is used by $A_n(K)$ and equal to n when the algorithm $A_n(K)$ uses the processors with a probability set $p_i = v_i / \sum_k v_k$, which corresponds to the most efficient use of the system in accordance with the values taken by the speedup. The effective number of processors and the speedup of the system are related in the next proposition. Note that n_{eff} can take also values larger than n when the decomposition of the program in smaller parts leads to the removal of memory bottlenecks due to improve utilization of main or cache memories. This phenomena has been observed in other studies and it is commonly known as super-linear speed up [15].

Proposition 4. *Given any parallel computer C_n and any parallel algorithm $A_n(K)$, it is always true that $s \leqslant n_{\text{eff}}$, and the equality $s = n_{\text{eff}}$ is true if and only if v_i are all equal, i.e. homogeneous computer* (*see proof in appendix*).

The interpretation of this statement is simple: if an algorithm does not use all the processors efficiently (low n_{eff}), then it will not be possible to reach high levels of speedup. When the algorithm uses the processors in the most effective way achieving the maximum possible speed up, n_{eff} reaches the value of n. This makes n_{eff} a convenient and intuitive property to evaluate the effectiveness of an algorithm as it follows closely our ideas for homogeneous systems.

3.3. Diversity of configuration

Many metrics to measure the diversity in heterogeneous systems have already been proposed, for example see Ref. [19]. The problem is that these metrics cannot be related analytically to the performance of the parallel system. Therefore, we would like to propose a definition of diversity that describes the variation of the speed of the processors but at the same time gives an idea of the maximum speedup that can be reached by the parallel system. The maximal speedup can be written as

$$s_{\max} = \frac{1}{\max\{v_i\}} \sum_k v_k \text{ or } s_{\max} = \frac{n}{1 + d_{\text{conf}}}, \tag{9}$$

where

$$d_{\text{conf}} = \frac{v_{\max} - \bar{v}}{\bar{v}} \tag{10}$$

and $\bar{v}$ is the average of the processors speed in C_n. The magnitude d_{conf} is zero if and only if the system is homogeneous and if it is not, the maximal speedup can be calculated from the Eq. (9). For these reasons, we define d_{conf} as our metric of the hardware diversity in C_n, calling it *diversity of the configuration.*

3.4. Example of a parallel system and its metrics

In this section, we provide an example of the behavior of the different metrics given above for a simple parallel computer consisting of two processors. The speeds of the processors are $\{v_1 = v, v_2 = hv, h \geqslant 1\}$ and the running algorithm has a probability set given by $p_1 = p$ and $p_2 = 1 - p$. The diversity of the configuration and probability set are controlled by the parameters h and p. The metrics for this parallel system are

$$d_{\text{conf}} = \frac{h-1}{h+1}, \tag{11}$$

$$s = \begin{cases} (hp)^{-1} & \text{for } p \geqslant (1+h)^{-1}, \\ (1-p)^{-1} & \text{for } p < (1+h)^{-1}, \end{cases} \tag{12}$$

$$n_{\text{eff}} = \begin{cases} 1 + (1-p)(hp)^{-1} & \text{for } p \geqslant (1+h)^{-1}, \\ 1 + hp(1-p)^{-1} & \text{for } p < (1+h)^{-1}. \end{cases} \tag{13}$$

Fig. 1 shows the shape of the speedup and the effective number of processors for four diversities ranging from 0 to

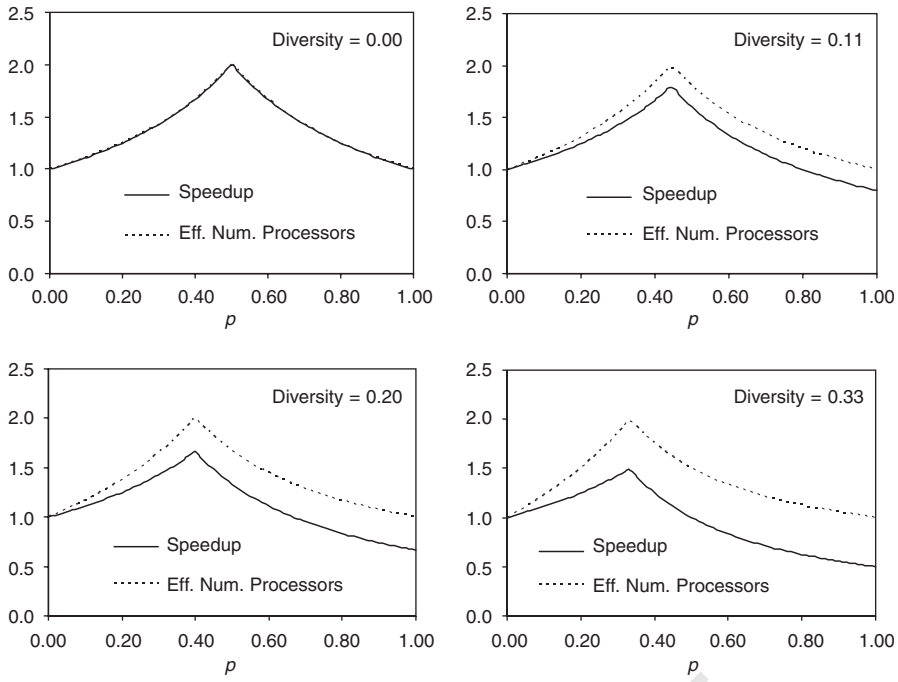

Fig. 1. Speedup and effective number of processors for a system of two processors with nonunity speed relationship.

0.33, corresponding to h ranging from 1.0 to 2.0, respectively.

As discussed above, for the homogeneous system the values of the effective number of processors and the speedup are the same, showing that the maximum efficiency is obtained when the workload is evenly distributed between the two processors. In heterogeneous systems, when the diversity precludes the speedup to be larger than $s_{\max} = (1+h)h^{-1}$, the effective number of processors still reaches the maximum number of processors in the case of the most effective distribution of workloads. Comparison of the values at which the maximum speedup and n_{eff} are achieved, shows a shift of the maximum towards a distribution for which the faster processor is used more often.

As an example, for the case of a diversity of 0.33 we can see that an optimal algorithm has $p = 1/3$ having a speedup 1.5. In the same system an algorithm distributing the job equally between the two processors ($p = 1/2$) will have a speedup of 1.0 because this probabilities set makes inefficient use of this highly heterogeneous environment.

4. Idle time

In the previous section, we have expressed the parallel performance metrics based on parallel elapsed time without taking into account the idle time incurred by the processors due to communication and synchronization tasks. A network and communication model is necessary to fully understand the impact of the idle time on the performance model, here we have simplified our model by introducing an average idle time. We define this average idle time as the difference between the parallel elapsed time calculated in Section 3 and the measured *total parallel elapsed time*, T_{tot}

$$T_{\text{id}} = T_{\text{tot}} - T_n. \tag{14}$$

Using the value of *total parallel elapsed time* from Eq. (4), the calculation of the system speedup is straightforward

$$s_{\text{tot}} = \frac{T_{1,\min}}{T_{\text{tot}}} = \frac{T_{1,\min}}{T_n + T_{\text{id}}}$$

or

$$s_{\text{tot}} = \frac{s}{1 + T_{\text{id}}/T_n}. \tag{15}$$

From Eq. (14), it is apparent that the idle time reduces the speedup of the parallel system by $s_c = s(1+f)^{-1}$. Therefore, the effects of the idle time on the performance can be interpreted as an additive increment to the diversity of the system given by

$$d_{\text{id}} = T_{\text{id}}/T_n, \tag{16}$$

which is the ratio between the idle time and the parallel execution time. In the Section 6, we show that from a given distribution $\{p_i\}$ of the algorithm and a total elapsed time T_n, it is possible to determine the relative contributions to the total diversity due to the processor heterogeneity from those arising from the idle time.

5. The adaptive parallel genetic algorithm

5.1. Introduction

The genetic algorithms are a family of search techniques rooted on the ideas of Darwinian biological evolution. These methods are based on the principle of survival of the fittest. Considering that each string or *genome* represents a trial solution candidate of the problem, at any generation the *genomes* or "individuals" compete with each other in the population for survival and produce off-springs for the next generation by prescribed propagation rules. Operators analogue to crossover, mutation and natural selection are employed to perform a search able to explore and learn the multidimensional parameter space and determine which regions of that space provide good solutions to the problem.

The fitness of the individual is given in general by a function that is, called the objective function. An example of this function can be the energy of a crystal or of an atomic cluster, where the objective is to find the structures with minimal energy. In many practical applications, the evaluation of the objective function requires large amounts of computer time, and it is generally the limiting factor on the problem size that can be solved. A possible solution to this problem is to use a global parallelization of the genetic algorithm. Such parallelization scheme relies on the simultaneous evaluation of the fitness of the individuals belonging to the same population in every generation.

A well-known example of a parallel genetic algorithm was implemented by Cantú-Paz [7]. This algorithm, which was designed for a homogeneous parallel computer system, distributes the same number of individuals per processor. Therefore, a large fluctuation in performance can be expected in heterogeneous environments. Using Eq. (5) the maximum possible speedup for this algorithm is given by

$$s = n \frac{\min\{v_i\}}{\max\{v_i\}}, \tag{17}$$

which for typical clusters with processors speeds covering a factor 4 could be as low as 0.25n. To correct this problem, while retaining the static distribution of workloads, it is possible to use runtime information of the heterogeneity of the processors to make the distribution of individuals nonhomogeneous. This approach is highly undesirable because it requires that the program interacts with the resource management software, which contains the speeds for the processors allocated to the job. Because there are no standard interfaces for supplying this information, the implementation of this strategy results in a nonportable program. This is an undesirable property for scientific applications that are supposed to run on numerous systems. The second alternative is to modify the algorithm into an adaptive scheme in which the processor loads are assigned dynamically during the run of the job. We had adopted this last scheme that is, described in the next section.

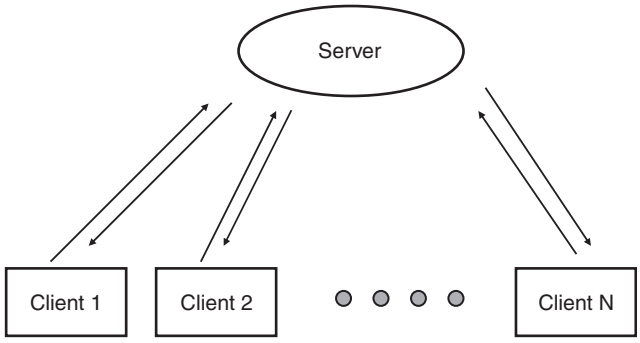

Fig. 2. Basic diagram of the Adaptive Parallel Genetic Algorithm.

5.2. Structure of the Adaptive Parallel Genetic Algorithm

The Adaptive Parallel Genetic Algorithm was implemented as an effective alternative for heterogeneous environments, since it automatically changes the number of individuals to be evaluated on each processor depending on the speed at which the individuals are processed in the nodes. This new algorithm was programmed using the GALib library (http://lancet.mit.edu/ga/) for the GA implementation and the MPICH library (http://www-unix.mcs.anl.gov/mpi/mpich/) for the intercommunication between processors.

The algorithm uses a server–client blocking message architecture, in which the *server node* is responsible for the distribution of work and the evolution of the genetic algorithm. The *client nodes* evaluate the fitness function for the individuals and return their values to the server node (see Fig. 2). The algorithm is divided into two units, the client program and the server program. Communication between processors is necessary for distributing and/or balancing the evaluation of a population over the nodes. Note, that this implementation is useful only when the evaluation time is much larger than the communication time between processors.

5.3. Client program

The client program has three states that are described in the following sub sections.

5.3.1. Initialization mode

The client waits for a *YesWork* signal to switch to active mode.

5.3.2. Active mode

When a client node is not working, it sends a message to the server requesting a new individual (*WorkRequest* signal), the server receives the message and the worker waits for the server's response (*YesWork* or *NoWork* signal).

If the client receives a *YesWork* signal, it waits until the information for the next individual is received and then eval-

uates the fitness function for the individual. When this evaluation is finished, the client sends a *WorkEnd* signal to the server and sends the results of the evaluation. If the client receives a *NoWork* signal, it switches to sleep mode and waits until a *YesWork* or a *Shutdown* signal is received.

5.3.3. Sleep mode

When a *YesWork* is received in sleeping mode, the node switches to active mode beginning the active cycle again sending a *WorkRequest* signal. However, if a *Shutdown* is received, the client shutdowns by itself and stops receiving any more signals. This signal is usually sent when algorithm finishes.

5.4. Server program

The server program has two states that are described in the following sub sections.

5.4.1. Initialization mode

The server sends a *YesWork* signal to all the worker nodes that have been assigned to the job.

5.4.2. Active mode

In active mode, the server receives work requests from the clients. The server's goal is to evaluate a complete generation of individuals by spreading the workload over the clients. The server starts by sending an individual to every client from which it received a *WorkRequest* signal. If there are no more individuals to evaluate, the server sends a *NoWork* signal to the client that requested the work. Then, that client passes to sleep mode and stops asking for more work. The switch of the clients to sleep mode prevents the clients from overloading the server with *WorkRequest* signals when the evaluation of a population is almost complete. When the algorithm completes the entire evolution the server sends to every node the *Shutdown* signal and exits the program.

Using this scheme a fast processor will request more work than a slow one and as a result, the algorithm will send more individuals to the faster processors adapting the algorithm to the heterogeneity of the system.

6. Performance measures

6.1. Measurements on a homogeneous system

We first measured the performance of the algorithm in a homogeneous environment. The objective of this measurement was to test the algorithm adaptation in the simplest environment possible and to measure the effects of idle time associated with the loading and synchronization processes.

An objective function with variable evaluation times of the 0.5, 1.0, 4.0 s was used to test the algorithm. These measurements were done on a Beowulf Linux Cluster built with AMD Athlon T-Bird 1333 MHz processors and with a network latency of approximately 5 ms when using the MPICH libraries. The number of generations was 10 and the population size was 32 individuals with a probability for mutation and crossover of 0.001 and 0.9, respectively. The algorithm was run with different number of processors: 2, 3, 5, 9, 17, where in each case one of the processors was used as the server. All the measurements were repeated 10 times, i.e. 100 generations, and the resulting average was used in the calculation of the speedup. The calculations of standard deviations and other statistical parameter of the distribution shows that the results presented are statistically significant. Fig. 3 shows the measured speedup for the algorithm vs. the number of clients for different evaluation times of the fitness function. It is important to note that when the number of clients is one, the algorithm has two nodes, one running as a server and the other running as a client. This is equivalent to a serial version of the program because there is only one client evaluating the fitness of the individuals. This serial version is not the most efficient algorithm because the elapsed time is given by the time required to evaluate the individuals plus an idle time due to the communication between these two processors. Comparing the case with zero clients (best serial version) and one client, it is easy to observe that the amount of extra time required in the case of one client results in a speedup smaller than one.

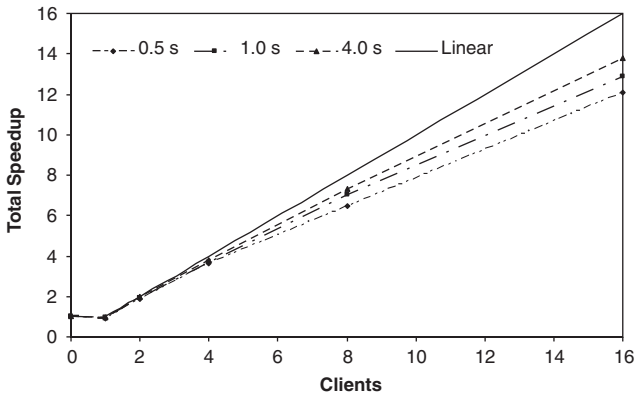

Fig. 3. Total speedup vs. number of clients for a homogeneous run of the Adaptive Parallel Genetic Algorithm. Dashed lines correspond to different objectives functions times. The full line represents the linear speedup.

The effectiveness of the algorithm increases for fitness functions requiring longer compute times and decreases with the number of individuals. This is because when the number of individuals is several times the number of processors, the algorithm has more individuals per processor to be distributed making the loading process more efficient. Furthermore, it is expected that the communication and the synchronization time increase when the number of clients increases. For example, the best speedup is 13.8 for the evaluation of 4 s objective function reaching an efficiency 0.86 with 16 clients, while in the case that eight clients and the same objective function, the algorithm reaches a speedup 7.3 that equates to a higher efficiency of 0.91.

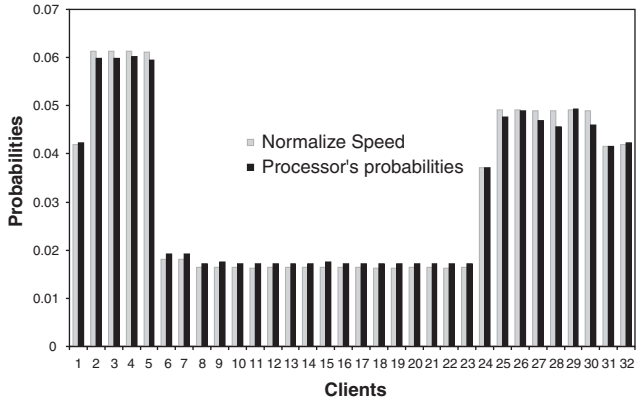

Fig. 4. Normalized speed and processors probabilities for the Adaptive Parallel Genetic Algorithm on a heterogeneous computer.

6.2. Measurements on a heterogeneous system

The measurement on a heterogeneous system was done on a Beowulf Linux Cluster that contains ∼ 400 processors with great variation in the clock speeds (400–1933 Mhz). For this purpose, we run the algorithm one time over 10 generations using 32 clients (33 processors) with a population size of 128 individuals. This is equivalent to the case of eight clients evaluating 32 individuals in the population discussed the last section, in both cases there is a relationship of four individuals per client.

The speed of the processors used in running this job was calculated measuring the total time used in the evaluation of the fitness function divided by the number of individuals assigned to each processor. Fig. 4 shows the normalized speed, defined by $v_i / \sum_k v_k$ of this particular run. We express the processor speeds using the normalized speed because they are dimensionless quantities that are proportional to the probability set of the most efficient algorithm in the heterogeneous environment (see Section 3). The diversity of configuration in this case is 0.96 corresponding to a maximal speedup of ∼ 16.3. As consequence of Eq. (17), the maximal speedup that the Cantú-Paz algorithm could achieve in this environment is 8.5 having an efficiency of 0.52.

The probabilities set for the algorithm can be evaluated from the fraction of individuals assigned to each processor. Fig. 4 compares the normalized speeds and the measured probabilities for the algorithm. The differences between the normalized speed and the probabilities is quite small, suggesting a nearly optimal workload. Using Eq. (5) the calculated speedup for the algorithm is ∼ 15.3, ignoring the idle time. The parallel elapsed and idle time for this run was measured $T_n \sim 361\,\text{s}$ and $T_{\text{id}} \sim 37.8\,\text{s}$ resulting in a 10% reduction of the speedup for a total of 13.9, Eq. (15). Therefore, the efficiency of the algorithm is 0.87, very similar to the values of 0.91 observed for the homogeneous case, showing a much higher efficiency that would be reached by the Cantú-Paz algorithm.

7. Conclusions

This paper provides a detailed discussion of the effects that hardware heterogeneity has on degrading parallel performance. The paper presents a framework in which it is possible to define a set of intuitive metrics that facilitate the performance analysis of parallel programs running on heterogeneous systems. These metrics also allow the differentiation between the factors leading to performance degradation associated with the heterogeneity of the hardware from those more widely recognized, arising from communications and synchronization delays. Finally, the paper describes a new adaptive implementation of a global parallel genetic algorithm. This algorithm is able to adapt to heterogeneous environment showing a efficiency of 0.87 that can be compared with the maximum possible efficiency of 0.52 by the traditional Cantú-Paz parallel genetic algorithm.

Acknowledgements

Financial support for this research by the University of Buenos Aires (UBACYT-X035) and the Argentinean CONICET (PIP 02151/01) is gratefully acknowledged. This work was partially supported by NSF Grant (INT-0071032). The software for this work used the GAlib genetic algorithm package, written by Matthew Wall at the Massachussets Institute of Technology.

Appendix

Proposition 1. *Given a parallel system C_n and a parallel algorithm $A_n(K)$ with a probability set $p_i = v_i / \sum_k v_k$, for any other algorithm $B_n(K)$ with a different probability set, the execution of the algorithm $B_n(K)$ requires a longer parallel elapsed time than $A_n(K)$.*

Proof. In the case of algorithm $A_n(K)$, the parallel elapsed time for this algorithm is $T_n(A) = K / \sum v_i$. Any other probability set associated to an algorithm $B_n(K)$ can be written as $p(B) = v_i / \sum v_k + \delta_i$. Where the δ_i are increments on the probability sets. Because $p(A) \neq p(B)$, there is a set of $\{\delta_i : \delta_i \neq 0\}$ where $\sum \delta_i = 0$ due to the normalization condition and therefore there is no empty set of $\{\delta_i : \delta_i > 0\}$. Now the elapsed time for every processor in $B_n(K)$ is $T_i = K / \sum v_i + K\delta_i / v_i$ so the parallel execution time for $B_n(K)$ is given $T_n(B) = T_n(A) + K \max\{\delta_i / v_i\}$ but we know that $\max\{\delta_i / v_i\} > 0$ so $T_n(B) > T_n(A)$, that is, the condition we want to prove. □

Proposition 2. *Given a parallel computer C_n and a parallel algorithm $A_n(K)$, it is always true that $T \geqslant T_n \geqslant T/n$. The*

equalities $T = T_n$ and $T_n = T/n$ are true when $p_i = \delta_{ij}$ for a given $1 \leqslant j \leqslant n$ and when $p_i = v_i / \sum_k v_k$, respectively.

Proof. (i) By definition T_n is the maximum T_i, $T = T_n + \sum_{T_i \neq T_n} T$, but because the rest of the T_i different than T_n are always $T_i \geqslant 0$, therefore $T \geqslant T_n$.

(ii) By definition $T_i \leqslant T_n$ therefore $\sum T_i \leqslant n T_n$ or $T \leqslant n T_n$.

(iii) Replacing $p_i = \delta_{ik}$ in the definition of T and T_n can be verified that $T = T_n$.

(iv) Replacing $p_i = v_i / \sum v_k$ in the definition of T and T_n we have

$$T = \sum \frac{K p_i}{v_i} = nK / \sum v_i = K/\bar{v}$$

$$T_n = \frac{K}{\sum v_i} = \frac{K}{n\bar{v}} = \frac{T}{n}. \qquad \square$$

Proposition 3. *Given any parallel computer C_n and any parallel algorithm $A_n(K)$, it is always true that*

$$\frac{\min\{v_j\}}{\max\{v_i\}} \leqslant s \leqslant \frac{1}{\max\{v_i\}} \sum_k v_k$$

and $s = \frac{\min\{v_j\}}{\max\{v_i\}}$ if $p_i = \delta_{ij}$ where j is the index of the slowest processor and $s = \frac{1}{\max\{v_i\}} \sum_k v_k$ if $p_i = v_i / \sum_k v_k$, i.e. for the most efficient probability set.

Proof. (i) First, $p_i/v_i \leqslant 1/v_i$ because $0 \leqslant p_i \leqslant 1$, therefore $\max\{p_i/v_i\} \leqslant 1/\min\{v_i\}$. Replacing this inequality in the definition Eq. (5) we have:

$$s \geqslant \frac{\min\{v_i\}}{\max\{v_i\}}$$

by definition this is possible if $p_i = \delta_{ik}$ where k is the index of the slowest processor.

(ii) Using the Proposition 1, the minimum parallel elapsed time is given when $p_i = v_i / \sum v_k$, but this is precisely the condition where the algorithm reach the maximum speedup. Therefore, the maximum speedup is reached when $p_i = v_i / \sum v_k$. Replacing this value in the Eq. (5) we obtain

$$s = \frac{1}{\max\{v_i\}} \sum_k v_k. \qquad \square$$

Proposition 4. *Given any parallel computer C_n and any parallel algorithm $A_n(K)$, it is always true that $s \leqslant n_{\text{eff}}$, and the equality $s = n_{\text{eff}}$ is true if and only if v_i are all equal, i.e. homogeneous computer.*

Proof. It is easy to see that:

$$\sum p_i/v_i \geqslant \sum p_i / \max\{v_k\} \text{ or } \sum p_i/v_i \geqslant 1/\max\{v_k\}$$

replacing this inequality in the definition of speedup Eq. (5) we have

$$s \leqslant \frac{\sum p_i/v_i}{\max\{p_k/v_k\}} \text{ that is, } s \leqslant n_{\text{eff}}$$

by the definition of the n_{eff} in Eq. (7). $\square$

References

[1] E. Alba, A.J. Nebro, J.M. Troya, Heterogeneous computing and parallel genetic algorithms, J. Parallel Distrib. Comput. 62 (2002) 1362–1385.

[2] E. Alba, J.M. Troya, Improving flexibility and efficiency by adding parallelism to genetic algorithms, Statist. Comput. 12 (2002) 91–114.

[3] J. Al-Jaroodi, N. Mohamed, H. Jiang, D. Swanson, Modeling parallel applications performance on heterogeneous systems, in: Workshop on Advances in Parallel and Distributed Computational Models, Nice, France, 2003.

[4] R.S. Barr, B.L. Hickman, On Reporting the Speedup of Parallel Algorithms: A survey of Issues and Experts, Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX, 1991.

[5] F. Berman, G. Fox, T. Hey, Grid Computing: Making The Global Infrastructure a Reality, Wiley, London, 2003.

[6] E. Cantú-Paz, Illinois Genetic Algorithm Laboratory, 1997.

[7] E. Cantú-Paz, Efficient and Accurate Parallel Genetic Algorithms, Kluwer Academic Publishers, Dordrecht, 2000.

[8] L.A. Crowl, How to measure, present, and compare parallel performance, IEEE Parallel Distrib. Technol. 9 (1994) 9–24.

[9] V. Donaldson, F. Berman, R. Paturi, Program speedup in a heterogeneous computing network, J. Parallel Distrib. Comput. 21 (1994) 316–322.

[10] I. Foster, C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers, Inc., San Diego CA, 1999.

[11] J.L. Gustafson, Reevaluating Amdahl's Law, Comm. ACM 31 (1988) 532–533.

[12] D.B. Jackson, B. Haymore, J.C. Facelli, Q.O. Snell, Improving cluster utilization through set based allocation policies, in: Proceedings International Conference on Parallel Computing, Valencia, Spain, 2001, p. 355.

[13] A.H. Karp, H.P. Flatt, Measuring parallel processor performance, Comm. ACM 33 (1990) 539–543.

[14] P. Morin, ACM Symposium on Applied Computing, ACM, Atlanta, Georgia, 1998.

[15] M.J. Quinn, Designing Efficient ALgorithms for Parallel Computers, McGraw-Hill, New York, 1987.

[16] T.L. Williams, R.J. Parsons, The heterogeneous bulk synchronous parallel model, in: IPDPS2000, Cancun, Mexico, 2000.

[17] L. Xiao, S. Chen, X. Zhang, Dynamic cluster resource allocations for jobs with known and unknown memory demands, IEEE Trans. Parallel Distrib. Systems 13 (2002) 223–240.

[18] L. Xiao, S. Chen, X. Zhang, Adaptive memory allocations in clusters to handle unexpectedly large data-intensive jobs, IEEE Trans. Parallel Distrib. Systems 15 (2004) 577–592.

[19] L. Xiao, X. Zhang, Y. Qu, Effective load sharing on heterogeneous network of workstation, in: Proceeding of 2000 International Parallel and Distributed Processing Symposium, IPDPS'2000, Cancun, Mexico, 2000.

[20] Y. Yan, X. Zhang, Y. Song, An effective and practical performance prediction model for parallel computing on nondedicated heterogeneous NOW, J. Parallel Distrib. Comput. 38 (1996) 63–80.

[21] X. Zhang, Y. Yan, Modeling and characterizing parallel computing performance on heterogeneous networks of workstations, in: Proceedings of the Seventh IEEE Symposium in Parallel and Distributed Processing, SPDPS'95, 1995.

Victor. E. Bazterra is an Associated researcher at the Center for High Performance Computing. Of the University of Utah and recipient of a Fellowship of Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET, Argentina). He was born in Buenos Aires Argentina and received his Licenciado in Physics degree from the University of Buenos Aires, March 2001. He works in parallel scientific computing applications to chemical physics problems, with emphasis on Genetic Algorithms applications to the determination of organic crystals and atomic clusters.

Martin Cuma is a Scientific Applications staff member at Center for High Performance Computing (CHPC), University of Utah. He received Dipl. Ing. degree in Process Control in Metallurgy from Technical University of Ostrava, Czech Republic and Ph.D. in Physical Chemistry from Southern Illinois University in Carbondale. At the CHPC, he works on design, parallelization and optimization of scientific applications, mainly for molecular sciences and engineering.

Marta B. Ferraro is Professor adjunto in the Department of Physics of the Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires and an Independent Researcher of the CONICET (Argentinean Research Council). Dr. Ferraro received her Ph.D. in physics from the University of Buenos Aires in 1985. Dr. Ferraro is interested in the structure and design of molecules and materials using theoretical models based on first principles as well as their magnetic and optical properties. Dr. Ferraro is author of more 50 international publications.

Julio C. Facelli was born in Buenos Aires, Argentina and attended the University of Buenos Aires where he got his Ph.D. in physics in 1982, he is the director of the Center for High Performance Computing and Adjunct Professor of Chemistry, Physics and Medical Informatics. Dr. Facelli has been involved in numerous computer and network related research projects and in many University Committees dealing with Information Technology. He has extensive expertise in Computational Chemistry, Parallel Computing and Advance Network Applications. Dr. Facelli is co-author of more than 125 international per review publications and his research has been funded by NSF, NIH and DOE. He is referee for numerous international publications and funding agencies. He has taught classes in Physics, Chemistry, Computational Sciences and Telecommunication.