A Preliminary Analysis and Simulation of Load Balancing Techniques Applied to Parallel Genetic Programming

F. Fernández de Vega[1], J. G. Abengózar Sánchez[2], C. Cotta[3]

[1]Universidad de Extremadura
Mérida, España
fcofdez@unex.es

[2]Junta de Extremadura
Mérida, España
jg.abengozar@extremadura.es

[3]Universidad de Málaga
Málaga, España
ccottap@lcc.uma.es

Abstract. This paper addresses the problem of Load-balancing when Parallel Genetic Programming is employed. Although load-balancing techniques are regularly applied in parallel and distributed systems for reducing makespan, their impact on the performance of different structured Evolutionary Algorithms, and particularly in Genetic Programming, have been scarcely studied. This paper presents a preliminary study and simulation of some recently proposed load balancing techniques when applied to Parallel Genetic Programming, with conclusions that may be extended to any Parallel or Distributed Evolutionary Algorithm.

Keywords: Parallel Genetic Programming, Load Balancing, Distributed Computing.

1 Introduction

Evolutionary Algorithms are nowadays routinely applied for solving search and optimization problems. They are based in Darwinian principles: By means of progressive refinement of candidate solutions, evolution can provide useful solutions in a number of generations.

Nevertheless, EAs and, particularly those employing variable size chromosomes, such as GP, have a problem when facing hard optimization problems: they require large computing resources and time to reach a solution. Researchers have demonstrated for GP, that individuals tend to grow progressively as generations are computed, thus featuring the well known bloat phenomenon [4]. Therefore, a number of factors have led researchers to making use of some degree of parallelism: the large number of candidate solutions -individuals from the population- that must be evaluated every generation; the large number of generations frequently required to reach a solution, and the high computing cost due to fitness evaluations.

Although researchers have deeply studied parallel models when applied to EAs [7][2], few have considered the need of specifically designed Load Balancing techniques. This could be particularly relevant for GP, given the differences in complexity and time required for evaluating each of the individuals of the population -featuring different sizes and structures [4].

This paper addresses these questions for GP using the well known Master Slave model. Using standard tests problems for GP, and by means of simulations, we analyze different load-balancing techniques and their usefulness when running GP on parallel or distributed infrastructures.

The rest of the paper is organized as follows: Section 2 presents Parallel Genetic Programming and Load Balancing principles. Section 3 describes our methodology and Section 4 presents the simulations and results obtained. Finally, Section 5 includes the conclusions.

2 Parallel Genetic Programming and Load Balancing

Genetic Programming was popularized by John Koza in the nineties [3], and rapidly grew with the work of researchers that not only employed it for solving problems, but also developed their mathematical foundations [4].

The main difference with GAs also leads to one of their main drawbacks: the variable size of chromosomes encoding candidate solutions. The size increase that usually happens when the evolutionary process takes place, as well as the difficulty of problems usually addressed, makes frequently necessary the use of some degree of parallelization.

Among the parallel models described in the literature, and analyzed for GAs and GP [7], we are particularly interested in the Master-Slave model. Basically, It tries to simultaneously compute the fitness function for a number of individuals of the population -tasks assigned to slaves- and then evolve the next generation in the master, so that the distribution of new fitness evaluations can proceed. The advantage of this parallel model is that it doesn't introduce any change in the main algorithm. The distribution of tasks -fitness evaluations- must follow some load-balancing policy.

Load Balancing aims at properly distributing computing tasks among processors, so that all of them employ similar time when computing their assigned tasks, therefore reducing *makespan*, i.e., time elapsed from the beginning of the first task to the completion of the last one. It is not always easy to reach that goal: differences in processor architectures and uncertainty in task sizes are some of the factors that influences the problem.

If we refer to Parallel GP, some detailed analysis of Parallel GP has been published in the last decade, particularly for the Island models [2], but no specific study on load-balancing techniques has been recently published. We must go back to 1997 to find the first papers considering the importance of Load-Balancing when using Master-slave versions of Parallel GP [1]. Usually authors have considered the application of Load Balancing techniques when addressing other problems [8], [9].

This paper tries to continue this area of research, by analyzing new load-balancing techniques that has been successfully developed recently. In this context, it is relevant the work by Yang and Casanova, that defines new load-balancing policies that are based in task sizes and different ordering principles [11], [12]. Next section considers the application of the proposed tasks ordering to GP.

3 Methodology

In our study, we will consider the use of a Master-Slave GP model. Tasks to be distributed and run simultaneously will consist of the fitness evaluation for each of the individuals. Therefore, we will have as many tasks as individuals in the population. The main goal is to analyze the application of different load-balancing policies. We must be aware that in GP two individuals with the same size may feature different complexities: this is due to the use of different functions within the program structure [3]. Measuring sizes or complexities may thus lead to different results when using load-balancing techniques.

When evaluating load-balancing techniques, a number of factors must be considered. As described by Yang and Casanova [11], [12], equation (1) describes the communication time for the master with a given slave *i*:

$$Tcomm_i = nLat_i + \frac{chunk_i}{B_i} + tLat_i \qquad (1)$$

where $nLat_i$ refers to the time required for beginning the communication, $chunk_i$ is the amount of information including in task *i*, B_i is the communication rate, and $tLat_i$ is the time elapsed since the master finishes the sending of $chunk_i$ until slave *i receives* the last byte. In the meanwhile, the master can begun another communication with a different slave. Both $nLat_i$ and B_i are independent on the data size that is sent. On the other hand, computing time for a given slave *($Tcomp_i$)* can be evaluated as described in equation 2:

$$Tcomp_i = cLat_i + \frac{chunk_i}{S_i} \qquad (2)$$

where $cLat_i$ is the time required for the slave to begin the running of the task, and S_i the speed of the processor. These values do not depend on the size of data to be processed.

As described below, some simplifications will be considered for this preliminary analysis. Specifically, we will focus on computing time, given that all the simulations and analysis will be performed on a single processor. The processor speed will be used as the basis for a simulated homogeneous distributed system, with all of the processors sharing the same features.

We have employed for the simulation two well-known GP problems: the artificial ant on the Santa Fe trail, and the even parity-12. A complete description of both problems can be found in [3][5][6]. The experiments have been run using

Evolutionary Computation in Java (ECJ), and the basic parameter configuration included in the tool. ECJ has been developed by ECLab[1] (Evolutionary Computation Laboratory), George Mason University, Washington DC.

As stated above, all the simulations have been run on a single computer: Intel Centrino Duo 1,7 Ghz. For both problems 100 individuals have been employed in the population, and 50 generations have been computed. All the remaining parameters have been employed as defined in ECJ for both problems, so that the replication of the experiments can be easily performed. Some changes in the source code have been applied so that the computing time -the only information of interest for the simulation- can be computed. Therefore, we obtain the computing time for each individual evaluation. This basic information obtained in a run, is then considered when evaluating the performance that a given load-balancing policy will obtain in a parallel or distributed infrastructure, whose processors would share exactly the same features as the one employed for the simulation. Of course, with the data obtained, conclusions that may be drawn could be easily extrapolated to other infrastructures whose features are known.

4 Simulation and Results

We have computed the evaluation time for each of the individuals, and then the evaluation time per generation. This is the total computing time required for running experiments in a single processors. Moreover, given that task completion time in a single processors heavily depends on other tasks that are run on the background -due to the operating system, cron tasks, etc- we have performed each of the experiments 10 times using the same random seed, so that we know that exactly the same individuals are generated every generation, every run. We have then computed the average time per individual, which provides a good approximation for their actual computing time.

Figure 1 shows computing time required for each of the experiments along the 50 runs. First of all, we notice that Even Parity-12 is harder than the Ant problem. Although this is not new, this information is relevant when considering the effect of load balancing policies for task distribution. The Figure also shows the maximum depth of individuals. We see that the Ant problem quickly reaches the maximum depth allowed (17 levels, as described in the literature). Again, this information is of interest if a relationship between size, depth and computing time is to be used for deciding tasks distribution and the load balancing technique to be used.

4.1 Analysis of Different Load-Balancing Policies

Let us consider now the situation on a homogeneous distributed system when the Master-Slave topology for Parallel GP is employed. We will analyze the results that would be obtained for different Load-Balancing policies when the main goal is avoiding processors idle time, consequently improving makespan.

1 http://cs.gmu.edu/~eclab/projects/ecj/

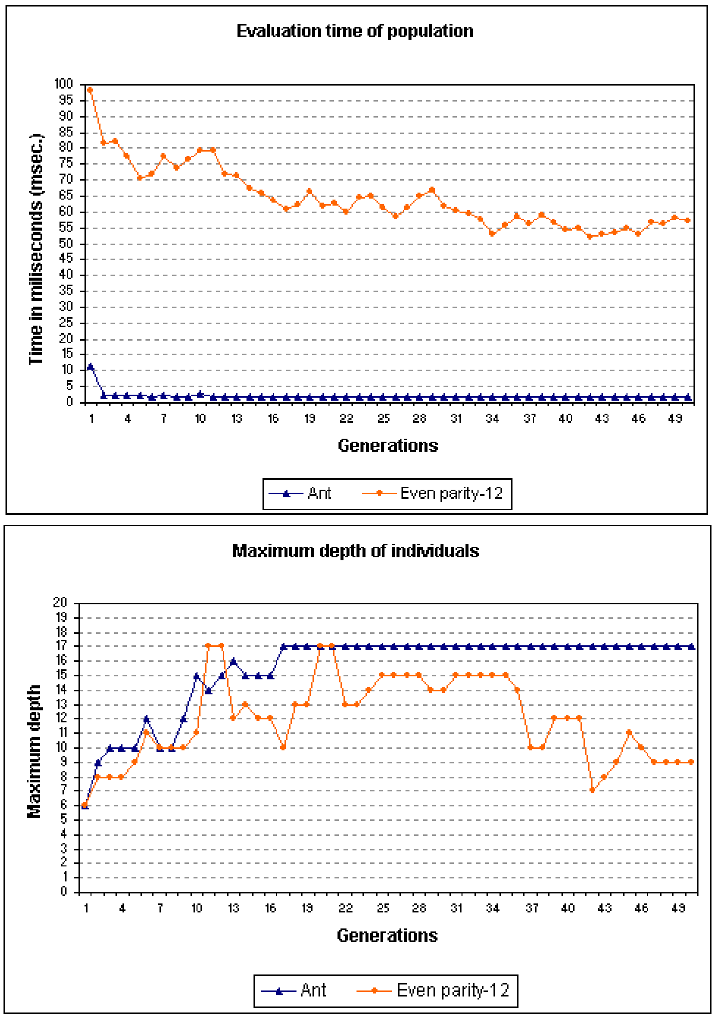

Fig. 1. Evaluation time and maximum depth per generation.

Figure 2 shows a typical diagram with different steps required for sending individuals, computing fitness in the slaves, and returning back results. This is useful to see how communication and computing time can overlap, thus reducing makespan. The relationship between communication time and computing time is also relevant when deciding the policy to be used.

Some preliminary conclusions can be drawn from the figure. When the total time required for evaluating the whole generation is short -this happens when fitness is computed quickly-, when compared with the latencies and total communication times of tasks (see the case of the ant problem, with low computing time per individual) the best choice would be to send as much individuals as possible in a single task. This way communication time is reduced.

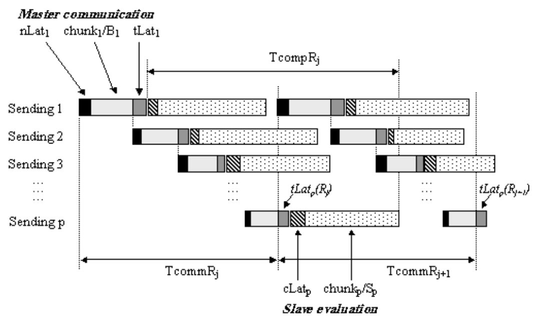

Fig. 2. Transmission and evaluation times in a parallel infrastructure.

On the other hand, if fitness evaluation takes long time, it is better to send individuals to processors in a round-robin fashion, so that communication time overlaps as much as possible with computing time. In this case, the decision about the size of tasks, and therefore the number of individuals to be included in every task has to be decided. A number of alternatives are available for generating tasks: (i) *Balanced Tasks*: All of the task will require the same computing effort and time. (ii) *Unbalanced Tasks*: according to Yang and Casanova [11] and [12], unbalanced tasks may be of interest in some circumstances for reducing makespan.

We will now analyze the computing time obtained for each of the individuals in both problems considered, ant and even-parity-12, and considering that the same number of individuals are sent to every processor. We consider a distributed system with 5 slave processors. We have 100 individuals per generation, so 20 rounds are required, sending 1 individual per round per slave. Of course, other possibilities are available.

If we compute the total time required for all the fitness evaluations, we obtain 1,9130 milliseconds for the ant problem and 58,6960 milliseconds for the Even Parity 12. This is the time employed by a sequential system, and the basis for the analysis.

4.2 Analyzing Task Ordering and Submission

When all the tasks are balanced -requiring the same computing effort-, a round-robin mechanism will send tasks in the following way: the first round task 1 -first individual from the population- is sent to slave 1, task 2 -second individual- to slave 2, and so on. Second round will proceed again by sending task n+1 to slave 1, n+2 to slave 2, etc. Every chunk submitted -task- requires initiating a communication operation with a slave. Therefore, the total communication time will strongly depend on the number of rounds and the number of slaves.

Regarding GP, notice that communication time of a task will be influenced by individuals size, while computing time by both size and complexity of individuals. If this complexity is low, then total computing time will be dominated by communication time. Processors will be idle long time. In this case, an infrastructure with low communication latencies will be a must: both supercomputer platforms or commodity clusters using optimized network connections will be required. Researchers could also consider the possibility of increasing population size, so that more individuals are available, larger tasks can be conformed and processors will thus spent more time computing. Given that there are idle processors, no reason for using larger populations should keep us from using the resources available.

The second possibility is that computing time is much longer than communication time. Processors would never be idle. In this case, other kind of distributed platforms could be used, such as Grids and Desktop Grids and Cloud Computing infrastructure.

4.3 Applying ordering

Another important factor is the order in which individual are sent. Several possibilities exist:

Random ordering: If we randomly pack individuals in tasks every generation, then, there will be random differences among completion time of tasks every round. The total time for a round is given by the task that takes longer. After considering the computed time for each of the individuals in the experiment, and computing the total time for every task in a round (5 tasks per round, given that 5 processors are considered), we have computing the total time of the experiment as the addition of all the rounds' largest task. We have thus obtained 0,4746 milliseconds for the ant problem and 17,6341 for the even-parity-12. This is better than the sequential time, but in can be even improved with better balancing techniques, as described below.

Weighted Factoring: Hummel et al. describe in [10] *Weighted Factoring* model. They consider task sizes and apply a descending ordering when submitting them to slaves. Therefore, we will consider first that the most complex tasks are is sent firstly. The advantage of this model is that for each of the rounds, all the tasks are similar, so the differences between computing time will be smaller. In the case of GP, this is only useful if we allow the algorithm to perform several rounds per generation. If a single round is to be performed per generation, the algorithm cannot work.

If we perform a simulation using the computing time for each of the individuals in both problems tested -5 processors, 20 rounds per generation, 1 individual sent to each processor per round- ordering them and computing the time for the largest individual in the round- we obtained for the ant problem 0,4082 milliseconds and 12,1201 milliseconds for the even-parity-12 problem. Nevertheless, if we use the size of individuals for the ordering instead of computing time, we would obtain 0,4715 and 14,6687 respectively. This confirms that even when using size for balancing is positive, it is better to use complexity – a kind of computing time estimation. Table 1 summarizes results and shows the differences obtained with each of the models.

5 Conclusions

This paper has presented a preliminary analysis on the application Load-balancing techniques to Parallel Genetic Programming. By analyzing the time required for evaluating each of the individuals in a population, we have studied differences between load-balancing methods that could be applied when using the master-slave model. This preliminary analysis allows us to reach some conclusions of interest. Firstly, problems with short fitness evaluation time must be run on supercomputers or commodity clusters with optimized network connections, and should never be run on Grid infrastructures. Second, weighted factoring approach allows to reduce makespan when compared to previously employed more standard Load Balancing Techniques. Results are sensitive to the use of Complexity or Size during the ordering process.

Table 1. Comparing Load Balancing Techniques.

Model	Ant – Computing Time	EP-12 Comp. Time.
Sequential	1,9130	58,6960
Random distribution	0,4746	17,6341
Weight. Fact. Cmpx.	**0,4082**	12,1201
Weight. Fact. Size	0.4715	14.6687

Acknowledgments. Spanish Ministry of Science and Technology, project TIN2008-05941, and Junta de Extremadura project GR10029 and European Regional Development Fund.

References

1. M. Oussaidène, B. Chopard, O. V. Pictet, y M. Tomassini, “Parallel Genetic Programming: an application to Trading Models Evolution,” *MIT Press*, págs. 357-362, 1996.
2. F. Fernández, M. Tomassini, y L. Vanneschi, “An empirical study of multipopulation genetic programming,” *GPEM*, vol. 4, nº. 1, págs. 21–51, 2003.
3. J. R. Koza, *Genetic programming III.* Morgan Kaufmann, 1999.
4. R. Poli, W. B. Langdon, N. McPhee, y J. Koza, *A field guide to genetic programming*. Lulu Enterprises Uk Ltd, 2008.
5. J. R. Koza, “Evolution and co-evolution of computer programs to control independently-acting agents,” in *First International Conference on Simulation of Adaptive Behavior. MIT Press, Cambridge, MA*, pág. 11 pp., 1991.
6. J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. MIT Press, 1992.
7. E. Cantú-Paz, “A survey of parallel genetic algorithms,” *Calculateurs Paralleles, Reseaux et Systems Repartis*, vol. 10, nº. 2, págs. 141–171, 1998.
8. G. Folino, C. Pizzuti, G. Spezzano, A scalable cellular implementation of parallel genetic programming. IEEE Transactions on Evolutionary Computation. Vol. 7 No.1, pp. 37-53. 2003.
9. N. Wang, A parallel computing application of the genetic algorithm for lubrication optimization. Tribology Letters, Vol. 18, No. 1, pp. 105-112. 2005
10. S. F. Hummel, J. Schmidt, R. N. Uma, y J. Wein, “Load-sharing in heterogeneous systems via weighted factoring,” in *8th annual ACM Symposium on Parallel Algorithms and Architectures*, págs. 318-328, 1996.
11. Y. Yang y H. Casanova, “UMR: a multi-round algorithm for scheduling divisible workloads” in *17th IEEE (IPDPS),* pp 24, 2003.
12. Y. Yang y H. Casanova, “RUMR: Robust Scheduling for Divisible Workloads,” in Proceedings *12th IEEE HDPC'03*, pp. 114, 2003.