Studying the influence of Synchronous and Asynchronous Parallel GP on Programs Length Evolution

G. Galeano[1], F. Fernández[1], M. Tomassini[2], L. Vanneschi[2]

[1] Computer Science Department, University of Extremadura
C/ Calvario, s/n. 06800 Mérida, Spain

{ ggaleano, fcofdez} @unex.es
http://atc.unex.es/pacof

[2] Computer Science Department, University of Lausanne
{ Marco.Tomassini, Leonardo.Vanneschi} @iis.uni.ch

ABSTRACT

In this paper we present a study of parallel and distributed genetic programming models and their relationships with the bloat phenomenon. The experiments that we have performed have also allowed us to find an interesting link between the number of processes, subpopulations and the model we should use when applying parallelism to GP. We study the synchronous and asynchronous version of the island-model in GP domain.

I. INTRODUCTION

It is well known that Genetic Programming (GP) programs tend to increase their size as population evolves [10,11]. This phenomenon is in some respect negative since it requires a large amount of computer resources to be managed. Several alternatives have been proposed to control that problem. In [11] some of these proposals are described: firstly, by placing a universal upper limit either on tree depth or program length; secondly, by incorporating a penalty which is proportional to program size; and finally, tailoring the genetic operations.

To our knowledge, the problem of bloat has always been studied by using sequential GP and panmictic populations. Nevertheless, when we studied different parallel models for GP in some of our previous research, we found an interesting relationship between program size evolution and the number of populations and individuals we employ (see [7]). We stated there that this relationship has much to do with the problem of bloat. In [5] and [6] we defined the concept of parameters region of effort; This characterises performance curves in parallel GP. Although we indicated that this region had a lot to do with the difficulty of problems and the optimum number of individuals that are required for solving problems, it can now be analysed again in the light of the new influences detected from the parallel and distributed GP models. This idea is more widely studied in this paper by means of a couple of benchmark problems and several set-ups.

We have also analysed the synchronous and asynchronous parallel GP models when completing our previous study on the influence of the number of individuals and communication time. We can thus have an idea about how the best performances are obtained.

At the same time, this study has also shown us some important links between firstly the total number of individuals and secondly subpopulations, and the number of processors we use to obtain results. Sometimes, when the brute force is not available, we must carefully consider the computer resources we have and GP models we must use to obtain the best results.

This paper is structured in the following way: Section 2 presents parallel models that are commonly used in evolutionary algorithms. In section 3 we describe the benchmark problems we have employed. Section 4 deals with the bloat phenomenon. Section 5 presents an study of synchronization models. In section 6 we show some ideas that establish a relationship between number of processors, populations and synchronization models. Finally, section 7 presents our conclusions.

II. PARALLEL MODELS FOR EVOLUTIONARY ALGORITHMS

One common problem when evolutionary algorithms are used for solving real-life problems is the large amount of computer resources that are required. Parallel models have been applied to Evolutionary Algorithms (EAs) over the last few years [1,2,3,8 and 15]. This is done by taking the concept of subpopulation: the main population is divided into several smaller ones that evolve at their own pace, exchanging individuals. The idea is to search different portions of the search space and also promote diversity by means of those migrating individuals (other configurations are also possible, see [13]).

If we focus on GP we see that individuals feature different sizes and complexities, and this means that evaluating different individuals may require very different lengths of time. Therefore, when we use several subpopulatios for

0-7803-7282-4/02/$10.00 ©2002 IEEE

devising solutions -the well known island-model- each of the subpopulations will evolve generations at a very different pace. At the same time, this is very important when we use several processors to compute subpopulations: depending on the way we synchronise subpopulations, we will obtain different results. This is one of the issues we investigate in this paper. Specifically, we study the synchronous and asynchronous versions of the island-based parallel and distributed GP. When we use the synchronous model, all processes -subpopulations- synchronise when sending and receiving individuals. Nevertheless, if we employ the asynchronous model, each population sends and receives individuals according to some internal measurements. No synchronization exists in this second model. Both parallel models are also employed to study the size evolution of individuals.

III. EXPERIMENTS

We have taken a couple of benchmark problems that are widely used in GP literature: the ant problem (see [11]) and the even parity 5 problem [9]. Summarizing, the ant problem tries to find some pieces of food which are positioned along a path on a two dimensional grid. The even parity function takes a number of Boolean inputs and returns TRUE only if an even number of inputs are true.

Performing the same experiments a number of times is useful for obtaining conclusions (14 times in [14] and 14, 25 or 60 times in [8]). We have decided to run each of our experiments 50 times. We have employed the *padgp tool* [5,6] and all the experiments have been conducted in a PC-LINUX 350 Mhz environment.

The *padgp tool* implements the *master/slave* model, each slave being a subpopulation. The master process is in charge of managing messages, while the communication topology is the ring: process n sends its individuals to process $n+1$.

The main GP parameters we have employed are the following ones: Crossover probability 98%; Mutation probability 50% for the ant problem and 5% for the even parity. Each population exchange 10% of their individuals each 10 generations for the ant problem, and each 5 generation for the even parity problem. We have employed the *padgp tool* (see [6] and [4]) which implements the master/slave model, each slave being a subpopulation. The communication topology is the ring: process n sends its individuals to process $n+1$.

IV. THE BLOAT PROBLEM

As mentioned above, there is an increase in size of GP programs when GP populations evolve. This has been fought by establishing limit values for the maximum size of individuals, or by penalizing the individuals' fitness values with a factor that is proportional to their size.

Nevertheless, in [7], we presented preliminary results which showed that when none of the above solutions are applied to island-based parallel GP, the individuals' length evolution varies according to the number of individuals and subpopulations we employ. In order to confirm that idea, we have performed a larger set of experiments.

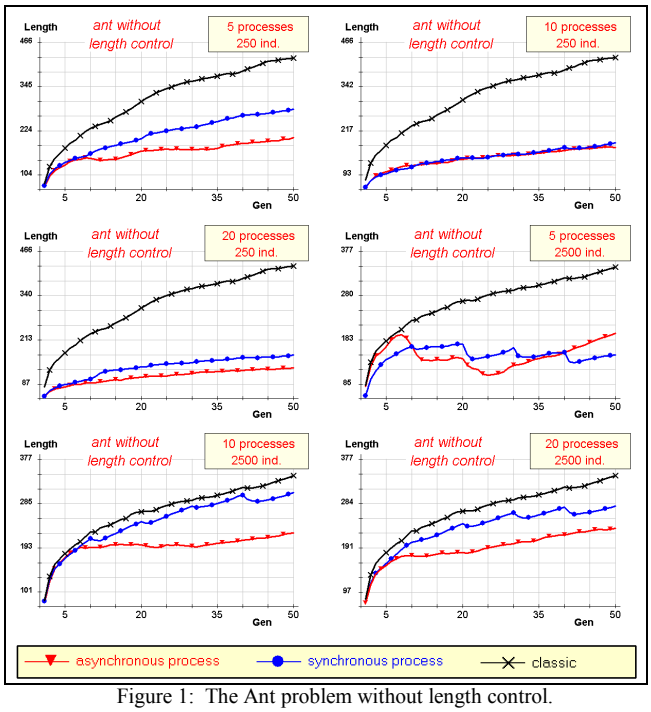

Figure 1: The Ant problem without length control.

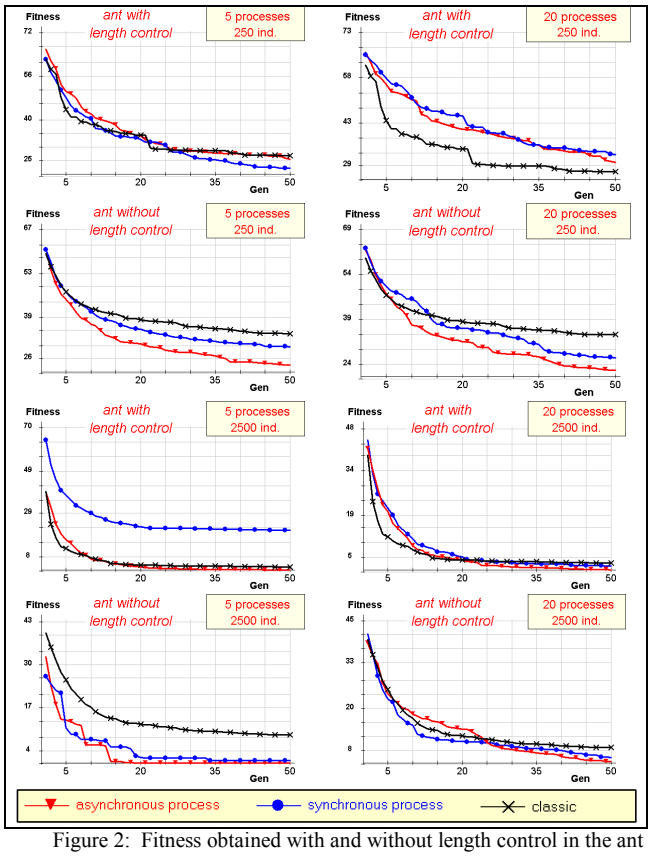

Figure 2: Fitness obtained with and without length control in the ant problem.

In figure 1, we show the ant problem with several configurations: panmictic GP and parallel GP based on the island model, with 250 and 2500 individuals and 5, 10 or 20 subpopulations. When performing these experiments, we have sometimes applied a penalty that is proportional to individual's size within the fitness function. This set of experiments has been labelled "with length control" in the graphs. When this restriction has not been applied, we have labelled graphs as "without length control".

When there is no length control the panmictic –labelled as classic in graphs– model shows a greater increase in size than the parallel models. This happens in almost all the configurations. However, when length control is applied, the classic model sometimes controls the average size of individuals better than parallel models (see figure 3).

The same experiments have been performed with the even parity 5 problem, and similar conclusions can be drawn (see figure 3 and 4). We can observe that when using the parallel model, the higher the number of populations we use, the smaller the increase in individuals size we find.

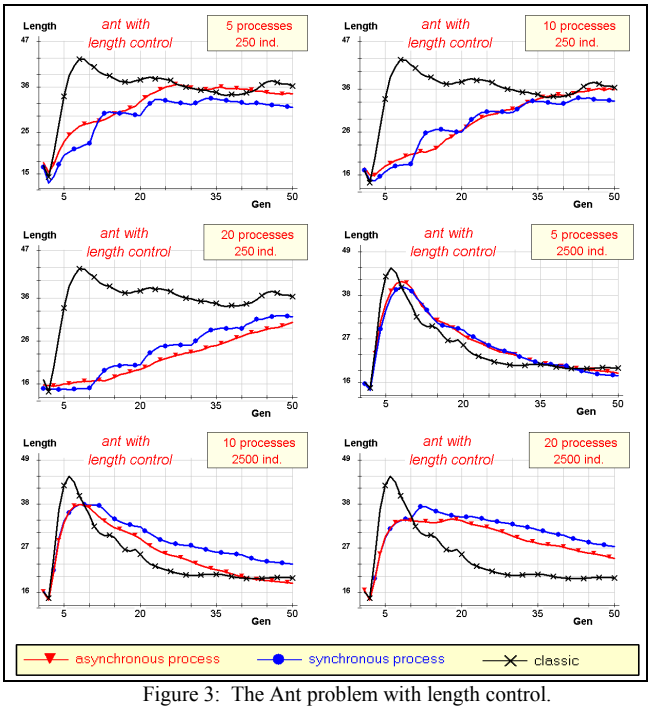

Figure 3: The Ant problem with length control.

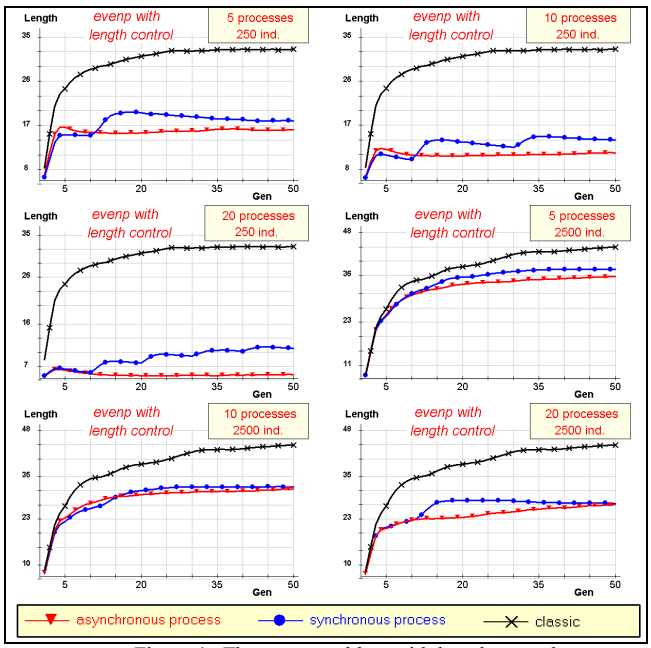

Figure 4: The evenp problem with length control.

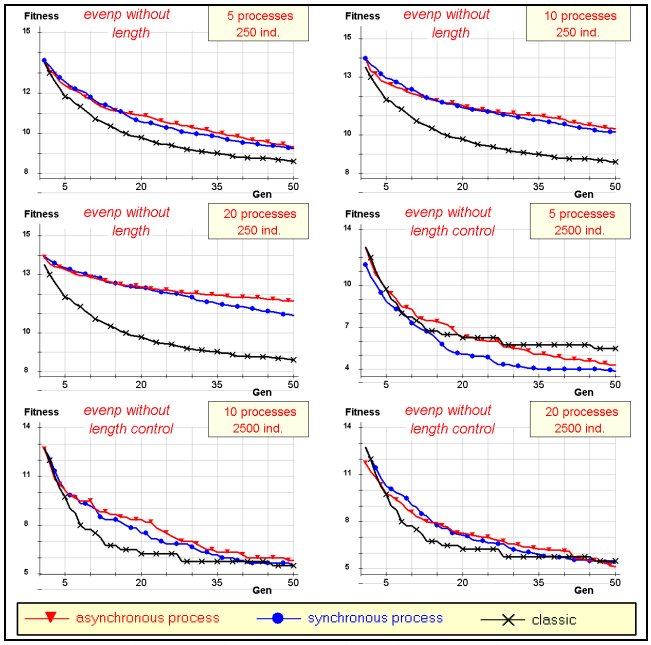

Figure 5: The evenp problem without length control.

We can thus state that length evolution depends greatly on the GP model we use. Both synchronous and asynchronous versions of GP restrict the bloat phenomenon when no penalization is applied within the fitness function. Nevertheless, this control is not so effective as when it is explicitly introduced into the fitness function. Consequently (when the latter is use) differences favouring parallel models are not observed.

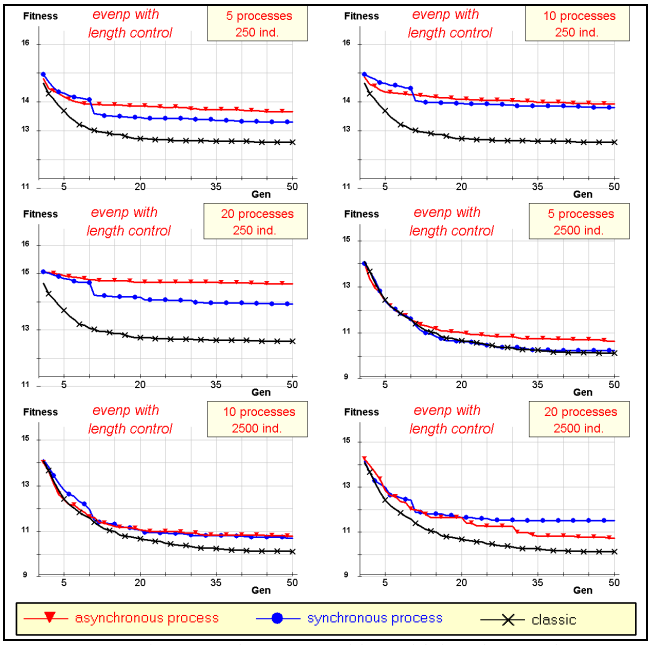

Figure 6: The evenp problem with length control

Figures 2, 5 and 6 show the fitness obtained by all the models studied. We see that sometimes panmictic GP obtains the same or even worse results than parallel GP (synchronous or asynchronous), but this depends on the number of subpopulations we use.

Furthermore, we have seen that a larger number of populations -which in turn means a smaller number of individuals per population- controls the bloat phenomenon better than using smaller number of populations, and this is the reason which allows parallel models to control bloat better than the panmictic model (the model with only 1 population) when no other control is applied. At the same time, given that differences in size are sometimes large and differences in fitness obtained are small, parallel models may obtaind better convergence results when we compare fitness to computing effort. This of course depends on the number of subpopulations we use, which confirms the notion of " region of effort" presented in [5].

V. SYNCHRONIZATION MODEL

We now continue by analysing the time required for evaluating generations within synchronous and asynchronous parallel GP.

Figure 7 shows us the time taken to compute each generation in the ant problem. We show different settings, and the curves can be classified in two groups: those that correspond to experiments with a high number of individuals and those with a low number of individuals.

If we focus on curves obtained when employing a higher number of individuals (2500) we can remark on an interesting issue: when we use the synchronous model, pre-migration generations take less time to compute. These effect becomes evident two or three generations before the migration phase. Something similar can also be observed when using the asynchronous model. But this time the reduction of time happens during the final generations of each experiment. This occurs when length control is applied and also when it is not employed

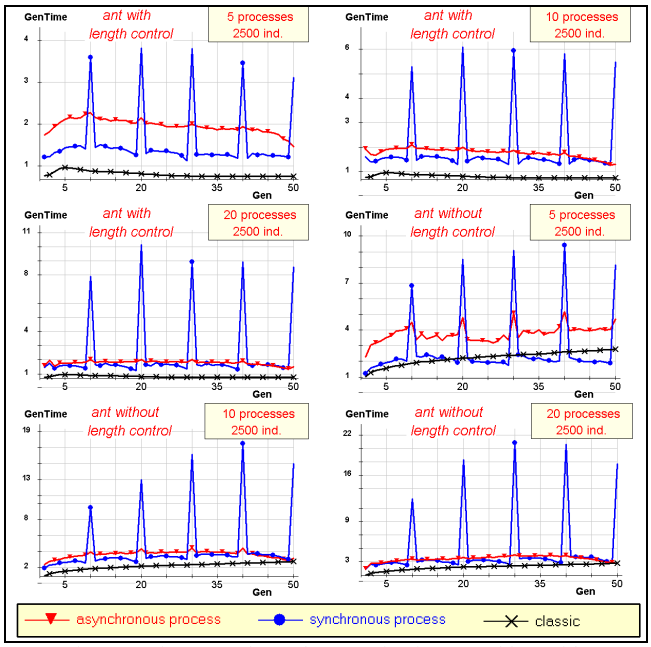

Figure 7. Time spent by each generation in ant problem with 2500 individuals

In order to explain that circumstance, we must remember how the Operating System works. When we use a monoprocessor system, which is endowed with a multi-process operating system (Linux), several processes can run simultaneously, thanks to the assignment of pieces of processor time -quantums- to different processes. Each process is assigned a quantum cyclically, until it finishes. When a process stop for an input/output operation, it is not assigned new quantums before it receives data. Its corresponding quantum is distributed among the other processes.

Padgp uses *mpich* [12] which is an implementation of MPI. In turns, *mpich* uses *sockets* for exchanging information between processes. The picture becomes complete if we say that Linux consider *sockets* as input/output operations. This means that each time a process is awaiting data from another process, the first one will become *idle* and will give its corresponding quantum to other processes, which in turn will compute quicker.

This operating system feature is the source for the temporal reduction in generations that are close to the migration generation. Within the synchronous model, if a process begins a migration generation, it sends its best individuals to the master process and becomes *idle* before individuals from other populations arrive. Other populations that have not yet arrived at the migration generation will use the supplied quantum in order to compute more quickly, thus

arriving earlier at the migration generation, and also providing new quantums for other populations. Sumarising, this issue causes the last few generations before the synchronisation step to do their work in less time on average (see figure 6), while migration generations require much longer.

Something similar happens within the asynchronous model: the lack of synchronization helps each population to compute at its own pace, never waiting for individuals, only receiving them when they are in the buffer. Due to the different speed of each process, the quicker ones will finish before the remaining ones. When they finish, their *quantums* are also distributed among the remaining ones, which are probably performing their final generations. This helps the last few generations to compute quickly in slower processes, allowing them to finish more quickly and providing in turn new *quantums* to even slower processes. This is the reason for the decrease in generation time for the final generations.

If we bear in mind all the experiments that we have performed using only one processor, we could conclude that the asynchronous model will be the preferred algorithm if we employ a multiprocessor system or even a cluster of computers: each population runs on a different processor, and the synchronous model will not be able to make use of quantums from idle processes.

On the other hand, if we focus on curves that have been obtained with a low number of individuals, the above mentioned effects are not so evident. This is due to the low communication rate (we always take 10% of the population size as the migration rate).

We can thus state that the number of individuals in subpopulations is a critical factor when determining the total time needed for obtaining solutions in parallel models, not only because they require time to compute, but also because they produce bottlenecks in communication processes. We could avoid this problem by suppressing the master process, allowing subpopulations to communicate directly, and also by using several processors. This final issue is studied in the following section.

VI. ADVANTAGES OF PARALLEL MODELS.

If we look at figure 8, we can see that the best performances are not always obtained with the same model. It greatly depends on the total number of individuals and number of populations. We must now observe figure 6 again, and be aware of the dynamics of synchronous and asynchronous models.

If we carefully study figure 9, we can draw the following conclusion if we attend to both the number of populations and processors we are using. Let M be the total number of processors we are using, and N the total number of populations:

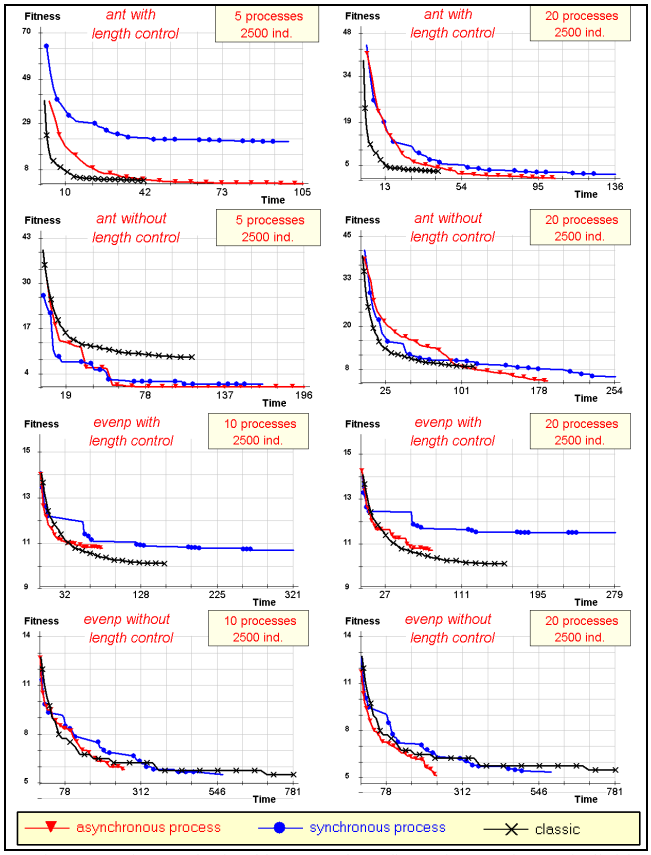

Figure 8: Fitness obtained by several configurations and 2500 individuals

1. If N=1, we are dealing with the panmictic model. There is no interest in using a parallel architecture.
2. If M=1, then we are using a monoprocessor system
   a. If N is large, then we should use the asynchronous or the panmictic model.
   b. If N is nor large nor small, we should use the panmictic model.
   c. If N is small, we better use shynchronous or asynchronous model.
3. If N>>M (N is much larger than M), then N/M=*k*, being *k* a large number. In this case we can apply conclusion 2.a within each of the processors (each of them must evaluate *k* subpopulations). But if we look at all the processors at the same time, the asynchronous model is preferable, given that it obtains better convergence when we are measuring fitness.
4. If N<<M, there are idle processors, and those that are working, are managing only 1 subpopulation. In this case the asynchronous model is also preferable.
5. If N≅M, then N/M≅1, and we are also managing each subpopulation with one processor: The asynchronous model is again the best choice. We save time and also obtain the best convergence results.

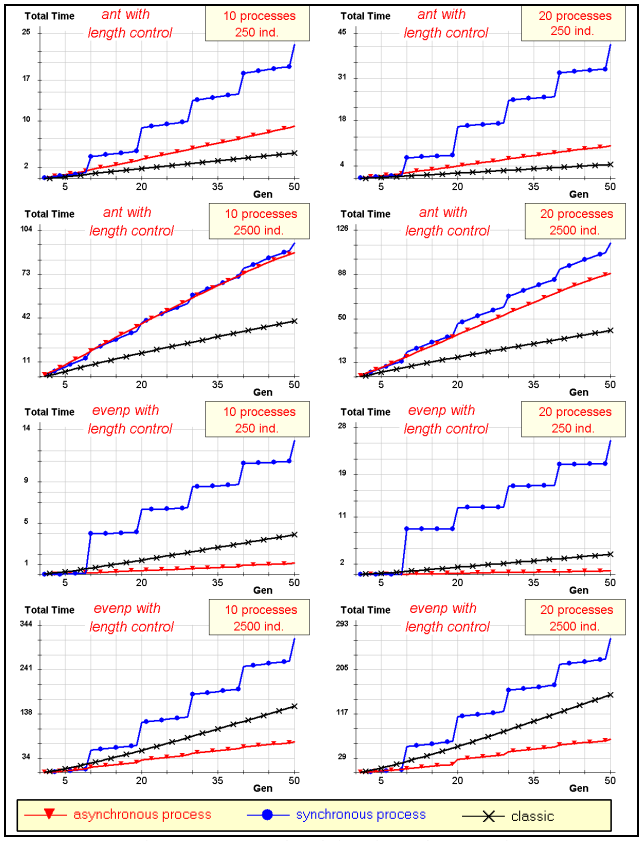

Figure 9: Acumulated time in each generation.

VII. CONCLUSIONS

In this paper we have studied synchronous and asynchronous parallel GP models. We have seen how they can help to control the bloat phenomenon when no other controls are applied. We achieve better control of bloat when we use a larger number of populations, each with a smaller number of individuals.

We have also studied the time required for computing synchronous and asynchronous models, as well as the influence of the Operating System when working on monoporocessor systems. This study has allowed us to draw conclusions about the performance that will be obtained in multiprocessor systems. The asynchronous model is the preferable when we use Parallel and Distributed GP with several processors. If we work on a monoprocessor system, both synchronous and asynchronous models obtain similar results.

REFERENCES

[1] E. Alba, J. M. Troya: "Analyzing synchronous and asynchronous parallel distributed genetic algorithms". Future Generation Computer Systems 17 (2001) 451-465

[2] David Andre and John R. Koza. "Parallel Genetic Programming: A Scalable Implementation Using The Transputer Network Architecture". P. Angeline and K. Kimea editors. Advances in Genetic Programming 2, Cambridge, MA, 1996.

[3] E. Cantú-Paz and D. Goldberg: Predicting Speedups of Ideal Bounding Cases of Parallel Genetic Algorithms. Proceedings of the Seventh International Conference on Genetic Algorithms. Morgan Kaufmann.

[4] F. Fernández, M. Tomassini, L Vanneschi, L. Bucher, "The GP's Tool". http: //www-iis.unil.ch/gpi/tool.html

[5] F. Fernández, "Parallel and Distributed Genetic Programming models, with application to logic síntesis on FPGAs", PhD Thesis. Universidad de Extremadura, February 2001.

[6] F. Fernández, M Tomassini, W.F. Punch, J.M Sánchez: "Experimental Study of Multipopulation Parallel Genetic Programming". In R. Poli, W. Banzhaff, W. Langdon, J. Miller, P. Nordin, T. Fogarti (eds) LNCS 1802 Proceedings of Euro Gp 2000. pp. 283-293.

[7] F. Fernández, G. Galeano, "Comparing Synchronous and Asynchronous Parallel and Distributed GP Models", EuroGP 2002. To appear.

[8] J. R. Koza, F. H. Bennett III, D. Andre, M.A. Keane: Genetic Programming III. Darwinian Invention and Problem Solving. Morgan Kaufmann Publishers. San Francisco.

[9] J. R. Koza: Genetic Programming. On the programming of computers by means of natural selection. Cambridge MA: The MIT Press.

[10] W. B. Langdon, "Quadratic Bloat in Genetic Programming", In Proceedings Genetic and Evolutionary Computation Conference GECCO 2000. Morgan Kauffman.

[11] W. Langdom and R. Poli. "Fitness causes bloat". In P.K. Chawdhry et. Al., editors. Soft Computing in Engineering Design and Manufacturing, pp 13-22. Springer London, 1997.

[12] MPI Forum (1995) MPI: A Message-Passing Interface Standard. http://www.mpi-forum.org/index.htm.

[13] A. Tettamanzi, M.Tomassini, "Soft Computing". Springer Verlag, Heideberg, Germany 2001

[14] W.F. Punch: "How effective are multiple populations in Genetic Programming". Genetic Programming 1998: Proceedings of the Third Annual Conference, J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, D. Goldberg, H. Iba and R. L. Riolo (Eds),Morgan Kaufmann, San Francisco, CA, 308-313, 1998.

[15] M. Tomassini, F. Fernández, L. Vanneschi, L. Bucher, "An MPI-Based Tool for Distributed Genetic Programming" In Proceedings of IEEE International Conference on Cluster Computing CLUSTER2000, IEEE Computer Society. Pp.209-216.