Genetic Algorithms and Grid Computing for Artificial Embryogeny

Sylvain Cussat-Blanc, Hervé Luga, Yves Duthen
IRIT – CNRS – UMR5505
Université de Toulouse – France
{cussat,luga,duthen}@irit.fr

Fabien Viale, Denis Caromel
INRIA Sophia Antipolis – France
{viale,caromel}@inria.fr

ABSTRACT

Genetic algorithms are very demanding in terms of computing time and, when the population size is large, they need days to complete or even fail due to memory restrictions. It is particularly the case for artificial life where each evaluation can take more than one minute to develop an artificial creature, plant or organism. Indeed, creatures are developed in physical and chemical simulators that require important computation resources. In order to create more and more realistic creatures, we propose a grid parallelized version of genetic algorithms. Two possibilities exist to increase them: supercomputers or computational grids. Because of their scalability, we choose computational grid in their works.

Categories and Subject Descriptors: I.2.8 [Artificial Intelligence]: Problem Solving, Control Methods, and search

General Terms: Algorithms, Experimentation

Keywords: Genetic algorithms, Grid computing, Artificial embryogeny

1. INTRODUCTION

Genetic algorithms, by their structure, tend to be easy to parallelize [3]. Three main methods exist to parallelize genetic algorithms : Master/Worker GA, Island GA and Hybrid GA. Whereas the first method gives exactly the same result as a classical GA, their results are not guarantied for the two last methods. However, experiments tend to prove their results are yet similar to classical GA in quality but needs more generation to converge [2].

The main goal of this work is to reduce the computation time of our artificial creature generation. We decide to apply a Master/Worker algorithm to parallelize our genetic algorithm. This algorithm is well adapted to artificial life because creature genome is small and the fitness computing cost is very important. Because of the small size of the genome, the network restriction forced by a Master/Worker architecture deployed on a computational grid will not heavily increase the computation time. Moreover, because the properties of a classical GA are preserved by the Master/Worker algorithm, the number of generations needed by the algorithm to converge and the final solution quality are exactly the same with or without the parallelization. In this paper, we present our method to parallelize genetic algorithm using the grid middleware ProActive. The next session presents this middleware.

GECCO'08, July 12–16, 2008, Atlanta, Georgia, USA.
ACM 978-1-60558-130-9/08/07.

2. PROACTIVE'S MASTER/WORKER API

ProActive is a grid programming middleware which provides, among others, a grid infrastructure abstraction [1]. The ProActive framework contains as well a set of toolkits which hide the inner ProActive concepts from the user and provide high-level APIs to well-known class of parallel problems such as Master/Worker, Branch & Bound or skeletons.

ProActive provides as well a deployment framework used to distribute an application anywhere without having to modify the source code. The key idea of the deployment framework lies within deployment descriptors. These descriptors are used to describe how resources can be acquired in a complex grid environment, for example by providing the chain of commands used to start remote processes. The ressource unit in ProActive is a local or remote JVM process. These resources are abstractly described by any ProActive application as java objects called *nodes*. These nodes are then used by ProActive applications, together with communication, registry and lookup protocols such as RMI.

ProActive provide different paradigms that simplify the use of the middleware. Here, we use the *Master/Worker* paradigm. In master/worker applications, a single *master* process controls the distribution of work to a set of identically operating *worker* processes. The master/worker paradigm has been used successfully for a wide class of parallel applications and is well suited as a programming model for applications targeted to distributed, heterogeneous "Grid" resources. The ProActive approach to Master/Worker applications is to provide a high-level API which:

- allows the user to simply and freely define tasks that will be executed by the workers.
- internally provides an automatic dispatching of work among the workers.
- provides a simple interface for result gathering
- handles fault-tolerance by redispatching tasks from dead workers.
- is implemented using ProActive.

3. EXPERIMENTS AND RESULTS

To study the efficiency of the Master/Worker genetic algorithm, we implement a parallel version of our genetic algorithm library using ProActive and specially the Master/Worker

API. We deploy the application on the french computational grid, Grid5000. This experimentation aim is to study the algorithm behavior on our artificial embryogeny model. This model allows the generation of artificial creatures able to grow in a virtual environment. We imagined an artificial embryogeny model able do create an organ starting from a single cell. This developmental model can be found in [4].

The simulation architecture is interesting for the parallelization because it has been coded using a multithread method. Each cell is an independent thread that perform its own action. This architecture introduces a second level of parallelization in the growth model.

In this experimentation, we implement a substrate transfer system[1]. All cell specifications are encoded in its genome. The genome size is about 19 kbytes and the simulation duration varies from a couple of seconds (when the organism is unable to do anything) to 120 seconds (when the organism develops itself in the environment and performs the asked action). Due to this important variation, the load balancing given by ProActive will optimize the task repartition amongst the workers. The genetic algorithm parameters for this problem are:

- Population size : 750
- Selection : 7 participants tournament with elitism
- Crossover rate : 65% ; Mutation rate : 5%

Curves in Figure 1 represent the computation time for each generation for one and 50 CPU. Note that the ordinate axis is graduated using a logarithmic scale for the time.

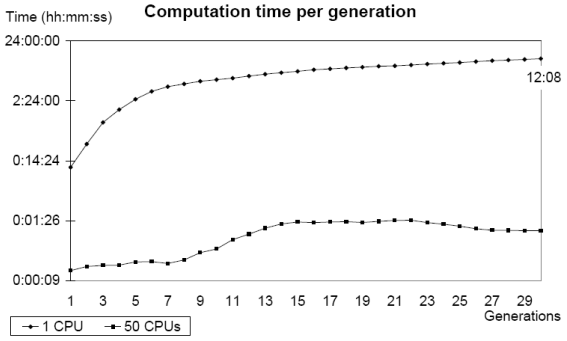

Figure 1: Computation time needed for each simulation generation (smoothed curve). The grid parallelization reduces the computation explosion due to the creature evolution.

First, the two curves increase generation after generation because the computation time to compute the fitness globally increases. Indeed, in the first generations, a lot of creatures are unable to survive. Then, they die immediately, the simulation stops and the computation time for such a simulation is very short (about one second). But, generation after generation, creatures evolve to use the environment's resources and to develop themselves. The computation time to evaluate each creature then grows up to 120 seconds.

The second interesting thing is that the global computation time is highly decreased. The network solicitation is acceptable and the computation time for each creature is sufficient to obtain good results even at the first generation. The difference between the two curves increases generation after generation. This computation time reduction is due to the load-balancing provided by ProActive. Workers always have a fitness to compute even if the last one was very short to compute (because the creature was unable to survive in its environment for example).

[1]Videos of creatures generated with this model are available on the website http://www.irit.fr/∼Sylvain.Cussat-Blanc

4. DISCUSSION AND FUTURE WORKS

In this paper, we present a parallel version of genetic algorithms based on grid computing. This version is especially interesting for our specific artificial embryogeny problem. Indeed, genomes have relatively small sizes and important computation needs. To parallelize this problem, we use a Master/Worker genetic algorithm. Results given by such an algorithm are very promising. The most interesting result is that the generation's computation time explosion due to creature evolution is reduced thanks to load-balancing. It significantly reduces the global computation time. The use of a computational grid implies some restrictions: it is almost impossible to repeat the same experimentation many times. Indeed, grids commonly have a reservation system to reserve a set of computers. Due to this management, it is hard to have exactly the same set two times on the bounce. Moreover, the network behaviour between two experiments can be very different and can impact the global computation time.

Another problem is the master memory explosion. Whereas a genetic algorithm uses an important quantity of memory, the use of ProActive middleware increases the problem. Indeed, each genome is encapsulated in a task to be sent to worker. This memory explosion can be reduced using an island GA or an hybrid GA because of the population distribution on the grid but the convergence time can be increased [2]. It could be interesting to implement an island algorithm using ProActive and to deploy it on the grid to compare it with our Master/Worker GA. Because the population is distributed amongst different computers, the memory problem would then be reduced.

In the future, we want to develop the model to increase the number of cells. We also want to develop tissues, organs, then primitive creatures and, on the long range, assemble organs to develop a complete creature. To manage more and more cells, ProActive Master/Worker API must also be enhanced to manage more and more nodes.

Acknowledgment :Experiments presented in this paper were carried out using the Grid'5000 experimental testbed (www.grid5000.fr).

5. REFERENCES

[1] F. Baude, D. Caromel, L. Mestre, F. Huet, and J. Vayssière. Interactive and descriptor-based deployment of object-oriented grid applications. In *High Performance Distributed Computing*, 2002.

[2] R. Bianchini and C. Brown. Parallel genetic algorithms on distributed-memory architectures. *Technical Report (revised version), University of Rochester*, May 1993.

[3] E. Cantù-Paz. A survey of parallel genetic algorithms. *Technical report 95004, Illinois Genetic Algorithms Laboratory, Urbana, IL*, 1997.

[4] S. Cussat-Blanc, H. Luga, and Y. Duthen. Artificial Embryogeny and Grid Computing. *Technical Report IRIT/RR–2008-10–FR*, IRIT, 2008.