Parallel Genetic Algorithm Implementation for BOINC

Malek SMAOUI FEKI [a] Viet Huy NGUYEN [a] and Marc GARBEY [a]
[a] *Department of Computer Science, University of Houston, 4800 Calhoun rd, Houston, Texas, 77004*

Abstract. In this paper we present our implementation of a Genetic Algorithm on the BOINC volunteer computing platform. Our main objective is to construct a computational framework that applies to the optimum design problem of prairies. This ecology problem is characterized by a large parameter set, noisy multi-objective functions, and the presence of multiple local optima that reflects biodiversity. Our approach consists in enhancing the iterative (synchronous) master-worker genetic algorithm to overcome the limitations of volatile and unreliable distributed computing resources considering a sufficiently large number of volunteer computers. Though volunteer computing is known to be much less performing than parallel environments such as clusters and grids, our GA solution turns to exhibit competitive performance.

Keywords. Genetic Algorithms, Volunteer Computing, Prairie Optimization, Ecology, Clonal plants

Introduction

Parallel Genetic Algorithms

Evolutionary algorithms (EA), namely Genetic Algorithms (GA), have been proved to be efficient optimization methods. GA's were clearly formalized in the late 80's and beginning of the 90's with the works of D. E. Goldberg and J. R. Koza[1,2]. Since then, they have been successfully applied to complex optimization problems [3,4,5,6]. GA's are inspired from nature. They use the principals of reproduction and natural selection ensuring the dominance of individuals of the most fit species. The GA starts with an initial population of potential solutions of the optimization problem (individuals) that are represented as strings of genes. This population evolves to a better fit population by crossover and mutation of some of the candidate solutions (reproduction process). The individuals participating in the reproduction process are selected according to their fitness to the problem i.e. how close they are to the optimization solution.
According to the objective function computation needs, a GA may require a considerable amount of computation. Thus, one of the most important extensions in GA's was their parallelization. The simplest parallelization scheme is a Global Master-worker distribution of fitness function evaluations [7]. However, Parallel GA's (PGA) models represent a broader class of algorithms with enhanced search strategies [8,9]. These PGA's have been implemented for networks of heterogeneous workstations, parallel mainframes and cluster grids [10].

Volunteer Computing

Nowadays, volunteer computing [11,12] is recognized as a viable and cost-effective parallel framework. Indeed, the access to High Performance Computing (HPC) facilities such as parallel mainframes, grids, elastic clouds, etc ... is not affordable by every scientist. Volunteer computing, however, is an arrangement between the scientist or the team of scientists and a group of volunteers in which computing resources are donated to a research project to satisfy its computation needs. Computing resources are mainly idle cpu cycles of Internet connected PC's owned by individuals or institutions. Obviously, such an arrangement cuts significantly the computation expenses of the project but offers limited freedom of use of the computing resources and lower performance. BOINC [13] is a well known middleware which enables the utilization of such volunteered computing resources. BOINC has a server/client architecture and is project based. In other words, clients running on the volunteered PC's can be attached to different projects running on different, independently administrated, servers. This feature allows resource sharing between projects. However, the main limitation of BOINC is that distributed tasks should be completely independent (embarrassing parallelism). This is due to the fact that commercial Internet is the only available and affordable communication medium between the computing entities.
The master-worker PGA has an obvious embarrassing parallelism and a structure that coincides with BOINC architecture. The main difficulty however is that the hosts of a BOINC project can be extremely volatile and perhaps unreliable. Provided that the project is attracting few thousands of volunteers and these issues are overcome, this framework could be ideal for applications having a large search domain dimension and high fitness evaluation computation requirements.

Target application: the Virtual Prairie project

The Virtual Prairie (ViP) project is a study of the dynamics of clonal plant populations and which goals are guiding engineering of prairies and helping biodiversity preservation [14,15,16]. The application simulates the growth of prairies of clonal plants (c.f. figure 1) using a parametric Individual Based Model (IBM) for each plant and a set of rules defining the interactions between the different plants. The model may take more than 16 variable parameters per species, and the dimension of the search space for a prairie with half a dozen of species end up with about 100 parameters to optimize! The model is highly stochastic: two runs with the same set of parameter values would produce different outcomes. Thus, each simulation outputs a number of prairie performance metrics. These are averaged over multiple runs of the simulation for the same parameter set. So, the actual outputs of the application are averages of prairie performance metrics which can be optimized using GA's. The choice of the objective function (performance metric) depends on the goals of the study which is not often completely known. In order to study the model and investigate its emerging properties, parameter space browsing have been carried out for isolated plants and for competing plants within prairies thanks to the ViP BOINC project [16,15]. More simulations are still in progress on this project. Detailed information can be found at the web site http://vcsc.cs.uh.edu/virtual-prairie/. The computation requirements of a prairie simulation ranges from few minutes to few hours depending on the problem size and the time scale. Once the objective function is

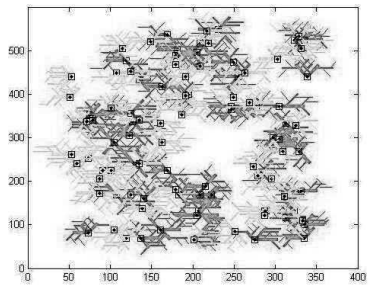

Figure 1. Example of a single run prairie simulation output.

clear, the ViP project can make good use of a PGA implementation on top of BOINC to fine tune the optimization process.

This paper aims at describing the methods used for PGA implementation on BOINC and the challenges that faced this work. While we have carried out optimizations using the virtual prairie application, we have set up a simplified benchmark application with multiple competitive maximum that can be better used for evaluation purposes.

1. Genetic Algorithm Library

The intuitive way to tackle this project is to leverage an open source GA library such as PGAPACK [17], a Parallel Genetic Algorithm library developed at Argonne National Lab by David Levine in 1995. We used this library to write a master-worker GA code optimizing the Virtual Prairie application with 16 parameters. We tested this code on a 72 cores SiCortex machine. The GA converged in less than 100 generation to the global maximum. The result was confirmed by the space browsing already carried out on BOINC (described above). However, using this library, we recognized the complexity and constraints that this tool would impose, when combining it with BOINC components. So, we decided to write our own GA library (MCS_GA) to gain more control over the data structures. MCS_GA offers a multitude of different GA operators: selection methods, crossover and mutation operators. In addition, it provides MPI implementations of multiple parallel genetic algorithm models including master-worker, multi-deme (multi-population) and steady-state.

2. PGA Model for BOINC

2.1. How does BOINC operates?

As mentioned previously, BOINC [13] has a server/client architecture. The main components of the server are (c.f. figure 2) work handling daemons and scripts, a data base, a file repository and a web interface. The vital work handling daemons for most projects are the work generator, the feeder, the scheduler, the validator and the assimilator. BOINC uses abstractions namely workunits and results for work handling. A workunit is a reference to an application and one or a set of input files. A result is a reference to a workunit and one or a set of output files. Workunits and results are manipulated through the data

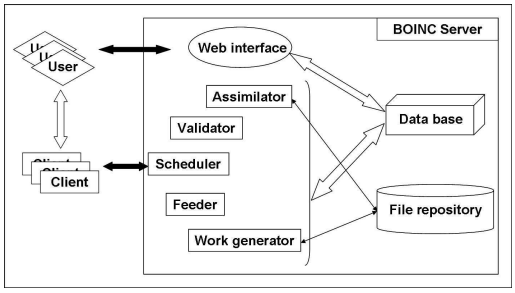

Figure 2. BOINC architecture.

base. Typically, the work generator creates or retrieves input files and creates corresponding workunits in the data base. The feeder creates results for the unprocessed workunits found in the data base. The scheduler will send the unprocessed results to clients requesting work. The validator verifies the correctness of output files returned by the clients after the computation is done. The assimilator aggregates and post processes these outputs.

2.2. GA model design

2.2.1. Implementation and challenges

The design of a master-worker GA using BOINC work handling daemons consists in incorporating the generational loop in the work generator. This latter would create input files containing each the gene string of one or a set of individuals of the population. It would create the corresponding workunits and leave the floor to the feeder and scheduler to scatter the jobs. Knowing that the assimilator is the unit responsible for assimilating computation results, it should then notify the work generator of the availability of current generation fitness results as soon as they are ready.
BOINC uses redundancy to deal with the volatility of clients and unreliability of their results. Thus, for each workunit (in our case fitness evaluation(s)), multiple replicas of the same task (result) are produced and sent to different clients. The validation process will ensure that, for each workunit, the output produced by the different hosts match. The replication and validation process may lead to multiple scenarios:

- The optimal situation is that initially two results are sent out to performing hosts. The two clients immediately process the tasks and send back the data, right away. The two outputs match and the evaluation is done.
- In most cases, clients do not start computation immediately because they might be busy doing other computations for other projects. That is why each task has a deadline and the client should return the results before the deadline is met. In case the deadline is passed, the server will issue another job replica. The server creates another job replica also in case the output of the first ones do not match. These situations can significantly slow down the whole process since the above described synchronous master-worker PGA has to wait for them to continue with the next generation.

2.2.2. Iterative versus steady-state algorithms

To overcome the above limitation, there are two possible solutions:

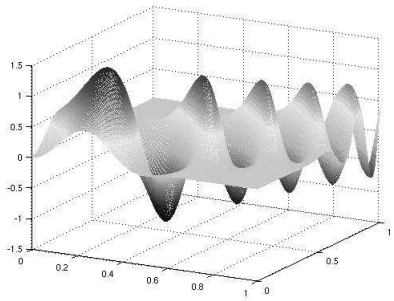

Figure 3. Plot of the benchmark function g

- One can use a master-worker steady-state algorithm where there is no iterations: the population is continuously updated by creating new individuals when clients request work and replacing old individuals by the new ones when clients return the new fitness values. Desell et al. described their implementation of this algorithm in [18] (where they call it Asynchronous Genetic Search)
- An alternative solution is to conserve the iterative character of the master-worker model while tweaking the algorithm and the BOINC project configuration.

We adopted the second solution here for two main reasons. First, for our target application, the optimization purpose is not to find the absolute optimum but rather end up with a sample population that represents regions of the domain around a number of significant maxima: multimodal optimization. Niching techniques in GAs allow such procedure. However, all the known niching techniques have been designed for iterative GAs. Thus, we need to prepare the ground for such addition to the algorithm. Besides, although earlier studies showed that steady-state PGAs may work as well as (or even outperform) iterative GAs, our experience with steady-state was not as successful. Indeed, using the MPI implementation of the steady-state algorithm and the iterative master-worker algorithms in MCS_GA we carried out some performance and robustness comparison through benchmark optimizations. We ran 20 instances of the maximization of the benchmark function g (c.f. figure 3):

$$g : [0,1] \times [0,1] \rightarrow R$$

$$g(x,y) = \frac{\sin(\mu^2(x^2+y^2))\exp^{-\lambda(x-y)^2}}{\log(\theta+x^2+y^2)} \; where \; \mu = 4, \; \lambda = 12 \; and \; \theta = 2$$

with the same selection, crossover and mutation operators and GA parameters using the iterative master-worker and the steady-state master-worker algorithms. We chose this benchmark function because several local maximum of the same order coexist, which is typical of our prairie model. Our comparison (c.f. Table 1) on the benchmark showed that though steady-state requires less fitness evaluations, it is less robust: results quality was much poorer.

GAs are not parameter-free methods, so we decided not to tune the GA parameter to our specific target application, but rather to use commonly used mutation and crossover rates. It would be difficult to optimize the parameter set of a GA for complex models such that of the Virtual Prairie application.

	Iterative master-worker	Steady-state master-worker
average nbr of evaluations	5615.2	2335.45
result average	1.330928	1.235792
result standard deviation	0.057996	0.177931
best result obtained	1.350497	1.350497

Table 1. Iterative versus steady-state comparative study

2.2.3. The iterative solution

We present here the heuristic rules that we added to the GA running on top of BOINC. These heuristics aim mainly at controlling the time required for the evaluation of the objective values of the population at each generation.
As soon as jobs for one generation are created, a deadline is fixed for the assimilator to report their outcome to the work generator. This deadline is initialized by an experimental estimation which depend (1) on the average execution time of the objective function evaluation and (2) the average time needed for the scheduler to scatter the jobs (depending on the population size). When, the assimilator meets this deadline two scenarios are possible:

- All the results were already received and sent to the work generator.
- Some results are still missing. In which case, the assimilator would notify the work generator of the end of computation if it already received at least 90% of the populations individuals fitness values or will extend the deadline if this threshold is not yet met.

• To lower the risk of missing some critical information in the remaining 10% individuals, the work generator tries to ensure that the 90% individuals that have been evaluated would be the best fit ones of the generation. So, at creation time, the work generator favors some individuals by assigning higher priorities to their corresponding jobs. Higher priorities are assigned to individuals with higher probability of being more fit. This probability is inversely proportional to the distance between each new individual and the best individuals of the past generation. Such probability can be also computed according to other similarity criteria between new and old individuals.
• To speed up the processing of each generation, jobs with higher priorities are sent only to reliable hosts which have a short turnaround time and low error rate. Besides, these jobs see their initial deadline reduced by a given ratio according to their priority level. That is why these jobs will have a very good chance of being returned before the generation deadline fixed for the assimilator.
• Finally all the jobs are replicated more than twice while their outputs are assimilated as soon as the two first matching results are received. This reduces the effect of late and erroneous results. This technique is affordable only with BOINC because the computing resources are abundant and cheap.
The job deadline, the priority of the job, and the redundancy factor of the job can be dynamically adjusted according to the behavior of the first generations. A statistical model that can be dynamically fitted on the run would be presented in a companion journal paper.

2.3. Tests and results

The ViP BOINC project served as a testbed for our GA. We ran maximizations of the benchmark function g described in section 2.2.2 using a population of 100 individuals, proportional selection and two-point crossover. The GA converges, when for 10 generations, the maximum obtained remains unchanged. The computation of the benchmark function is very fast which is not the case of real applications. So, we added 200 GFLOP of dummy cpu usage in the objective function to simulate a real application.
We ran the same maximizations using the master-worker MPI version of MCS_GA on:

- SiCortex computer using 100 cores (700 MHz).
- Beowulf cluster using 34 Opteron processors (2 GHz).

	BOINC	SiCortex	Beowulf
objective function computation time (avg)	variable	1270 s	365 s
GA execution time (avg)	63988 s	71305	67384
average result obtained	1.350497	1.350497	1.350497

Table 2. Performance of GA on different platforms

Table 2 details the results and performance obtained. It shows that the techniques used in our GA solution on BOINC allowed to obtain a comparable performance to that of a medium size SiCortex or a small size Beowulf cluster, when we imposed relatively short deadlines for the jobs (2 hours). However, BOINC offers two more advantages:

- the costs of acquiring, powering and maintaining clusters is much higher than those of a BOINC server setup and maintenance. So, BOINC becomes a viable solution for scientists who can not afford access to parallel computers.
- with the abundance of the computing resources offered by BOINC (the ViP project has more than 2000 hosts), we can perform optimizations with large population sizes without dramatically increasing the overall execution time. The only overhead that will be added is the time needed for scattering the jobs and which is much lower than the objective function computation cost. Large populations are much needed for multimodal optimizations.

3. Conclusion

In a quest for exploiting a cheap and abundant computation resource, that is volunteer computing, this paper describes the implementation of a genetic algorithm on top of BOINC. Though the iterative master-worker parallel genetic algorithm fits the distributed character of this environment, the volatility and unreliability of the resources offered by BOINC slows down significantly the process. The enhancements added to this algorithm combined to the proper use of BOINC features (like priorities) provided execution times competitive with those of much more expensive parallel computing resources. We expect that a single optimization of a virtual prairie with 5 species would require a dedicated Beowulf cluster with a thousand processor during several month. Because in our continuous dialogue with the interdisciplinary team of ecologists, we keep improving our model as more experimental result in the field come, we view volunteer computing as the best solution to our ViP project.

Acknowledgements

We would like to thank K. Crabb for supporting the ViP BOINC project at the University of Houston Research Computing Center as well as D. Anderson, the principal investigator of BOINC from University of California at Berkeley.

References

[1] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, January 1989.

[2] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT press, Cambridge, MT, December 1992.

[3] L. Dumas and L. El Alaoui. How genetic algorithms can improve a pacemaker efficiency. In *GECCO '07: Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation*, pages 2681–2686, New York, NY, USA, 2007. ACM.

[4] Y. Gao, H. Rong, and J. Z. Huang. Adaptive grid job scheduling with genetic algorithms. *Future Generation Computer Systems*, 21(1):151–161, 2005.

[5] S. Karungaru, M. Fukumi, and N. Akamatsu. Automatic human faces morphing using genetic algorithms based control points selection. *International Journal of Innovative Computing, Information & Control*, 3(2):247–256, April 2007.

[6] M. Olhofer, T. Arima, T. Sonoda, and B. Sendhoff. Optimisation of a stator blade used in a transonic compressor cascade with evolution strategies. In I. Parmee, editor, *Adaptive Computing in Design and Manufacture (ACDM)*, pages 45–54. Springer Verlag, 2000.

[7] E. Cantu-Paz. Designing efficient master-slave parallel genetic algorithms, 1997. Available online at: ftp://ftp-illigal.ge.uiuc.edu/pub/papers/IlliGALs/97004.ps.Z.

[8] E. Cantu-Paz. A survey of parallel genetic algorithms. *Calculateurs Paralleles, Reseaux et Systems Repartis*, 10(2):141–171, 1998.

[9] M. Nowostawski and R. Poli. Parallel genetic algorithms taxonomy. In *Proceedings of the third international conference on Knowledge-based information Engineering Systems*, pages 88–92, August 1999.

[10] D. Lim, O. Y.S, Y. Jin, B. Sendhoff, and L. B. A. Efficient hierarchical parallel genetic algorithms using grid computing. *Future Generation Computer Systems*, 23(4):658–670, 2007.

[11] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. Seti@home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.

[12] D. P. Anderson and G. Fedak. The computational and storage potential of volunteer computing. In *CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, pages 73–80, Washington, DC, USA, 2006. IEEE Computer Society.

[13] D. P. Anderson. Boinc: A system for public-resource computing and storage. In *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 4–10, Washington, DC, USA, 2004. IEEE Computer Society.

[14] M. Garbey, C. Mony, and M. Smaoui. Fluid flow - agent based hybrid model for the simulation of virtual prairies. In *to appear soon in the proceedings of the 20th Parallel Computational Fluid Dynamics Conference*, May 2008.

[15] M. Smaoui, M. Garbey, and C. Mony. Virtual prairie: going green with volunteer computing. In *2008 IEEE Asia-Pacific Services Computing Conference*, pages 427–434. IEEE Computer Society, December 2008.

[16] C. Mony, M. Garbey, M. Smaoui, and M. Benot. Optimal profiles for clonal plant growth: a modelling approach. submitted to a journal in ecology, 2009.

[17] D. Levine. Users guide to the pgapack parallel genetic algorithm library, technical report anl-991 8, 1995. Technical report, Argonne National Laboratory, 1995. PGAPACK is available via anonymous ftp at ftp.mcs.anl.gov or from URL http://info.mcs.anl.gov/pub/pgapack/pgapack.tar.Z.

[18] T. Desell, B. Szymanski, and C. Verla Asynchronous genetic search for scientific modeling on large-scale heterogeneous environments In *IEEE International Symposium on Parallel and Distributed Processing, 2008*, pages 1–12, Miami, Florida. April 2008.