Increasing GP Computing Power for Free via Desktop GRID Computing and Virtualization

Daniel Lombraña González*, Francisco Fernández de Vega*, Leonardo Trujillo†, Gustavo Olague†
Lourdes Araujo‡, Pedro Castillo§, Juan Julián Merelo§ and Ken Sharman¶
*University of Extremadura,Spain, Email:daniellg@unex.es, fcofdez@unex.es
†CICESE, México, Email: trujillo@cicese.mx, olague@cicese.mx
‡UNED, Spain, Email:lurdes@lsi.uned.es
§University of Granada, Spain, Email: pedro@atc.ugr.es, jmerelo@geneura.ugr.es
¶Polytechnic University of Valencia, Spain, Email:ken@iti.upv.es

***Abstract*—This paper presents how it is possible to increase the Genetic Programming (GP) Computing Power (CP) for free, via Volunteer Computing (VC), using the well known framework BOINC plus a new "virtualization" layer which adds all the benefits from the virtualization paradigm. Two different experiments, employing a standard GP tool and a complex GP system, are performed –with distributed PCs over several cities– to show the free achieved CP by means of VC, without the necessity of modifying or adapting the original GP source code. The methodology can be easily extended to Evolutionary Algorithms (EAs).**

I. Introduction

Genetic programming (see [13]) is an automated method for creating a working computer program from a high-level problem statement of a problem. GP starts from a high-level statement of "what needs to be done" and automatically creates the required computer program.

GP starts with a population of randomly created computer programs. This population of programs is progressively evolved over a series of generations by using the Darwinian principal of natural selection (survival of the fittest). The evolution process analogs various naturally occurring operations like crossover (sexual recombination), mutation, gene duplication and/or gene deletion. Finally, the individuals are evaluated using a fitness function. The fitness function scores the individuals based on how well the individuals solve the problem.

When real world optimization problems are faced using GP, the computing requirements are usually high. The high requirements and time-consuming features of real world GP problems are due to: (i) the complexity of fitness functions, (ii) the large amount of individuals employed and (iii) the number of iterations which are needed to solve the problem. Therefore, in order to relieve this issue, different parallel approaches were used in the past; for instance the parallel transputer network architecture [4] or a 10 nodes Beowulf style cluster [6]improved later to 1000 Pentiums nodes[1]. Nowadays, large efforts are still carried out to improve and embody parallel techniques to avoid or tackle the time consuming features of GP [9] and any other EA (see the Introduction to the special issue on parallel and bioinspired algorithms [11]).

[1]For further details see http://www.genetic-programming.com/machine1000.html

One of the most promising technologies capable of tackling the high requirements of many applications, and therefore reducing computing time, is the GRID computing paradigm (i.e. [12]). In the last few years, the GRID has become a powerful tool to deal with time-consuming applications from many different fields (see [12]). The GRID harnesses the power of super computers, clusters or desktop PCs –which are distributed over networks– by means of a special software called *middleware*. Depending on the harnessed hardware, there are different GRID approaches. One of the available GRID techniques is known as Desktop GRID Computing (DGC) [2]. The main features of DGC are: (i)it employs cheap hardware (PCs), (ii) it is easy to use and (iii) it has succeeded on a number research projects. Additionally, the DGC has a collaborative feature, allowing users from all over the world to anonymously donate their desktop resources to research projects. Thanks to this social behavior, the scientific research projects can obtain CP free (with no cost at all). As a result of this social characteristic, the DGC model is also known as Volunteer GRID Computing (VGC).

One of the most successful and well known DGC projects is SETI@HOME [1], which employs the BOINC middleware [2]. Thanks to the collaboration of 797600 users, SETI has been able to create a super virtual computer of 456.330 TeraFLOPS[2] for free. There are other available DGC middlewares such as Xtremweb [10] or Condor [15], however BOINC is one of the most employed and extended DGC middleware.

In summary, DGC is a good target to run real world problems and could be also employed to obtain free computing resources for running GP problems. Nevertheless, this technology has a main drawback, the fact that it is difficult to run an application directly without any modification. Therefore, employing a middleware with GP involves adapting the GP source code to the desired middleware. This "source code adapting step" sometimes is so complex and time consuming that using the chosen DGC middleware simply becomes impossible. This is probably one of the main reasons why DGC

[2]Data obtained from http://boincstats.com

1066-6192/09 $25.00 © 2009 IEEE
DOI 10.1109/PDP.2009.25

has not been used with GP before, even when the technology has been available for a decade. To our best knowledge only Chávez et al. [8] has presented the model by using a ported version of LilGP. On the other hand Samples et al. [17] shown the feasibility of the approach for a typical parameter sweep application with GP using a pool of desktop PCs. Nevertheless, the lack of a standard middleware and GP tool has kept this approach from being commonly adopted by researchers.

Therefore, what we propose in this paper is to use the VGC BOINC model plus a virtualization layer in order to:

- Harness a large number of BOINC resources (nowadays BOINC has 2847228 computers collaborating with scientific projects).
- Improve the CP of GP sequential executions thanks to the parallel environment which VGC provides (976.593 TeraFLOPS[3]).
- Show the benefits of running a virtual layer inside a VGC middleware which allows to run customizable execution environments, without modifying the source code of the GP application.

To sum up, our proposal is to increase GP computing power (CP) for free. Moreover, our proposal will, for the first time, allow to run any GP tool inside DGC without any source code modification, thanks to the new proposed virtualization layer inside BOINC. We have chosen to incorporate BOINC as our VGC environment because BOINC is one of the most employed DGC middlewares, with millions of users providing volunteer computing resources. For the virtual layer, we have chosen the widely used free software VMware. The main reason for choosing VMware is because it runs on the same platforms as BOINC.

The rest of the paper includes a description of the BOINC model in Section II; we present the VGC and the virtual layer model in Section III; Section IV shows the experiments and results. We conclude in Section V.

II. The BOINC Model

As described above, BOINC is a middleware that harnesses commodity computer resources for a given project. BOINC has two key features: it is *multiplatform* and *open source*. BOINC uses a master-slave model where the server is in charge of:

- *Hosting the scientific project experiments*. A project is composed by a binary (the algorithm) and some input files. The binary is classified according to the target platform (Ms. Windows, GNU/Linux, MacOSX) and architecture (x86 32-64 bits and sparc).
- *Creation and distribution of jobs*. In BOINC's terminology a job is called "work unit" (WU). A WU describes how the experiment must be run by the clients (the name of the binary, the input/output files and the command line arguments).

The BOINC client connects to the server and asks for work (WU). The client downloads the necessary files and starts the computations. Once the results are obtained, the client uploads them to the server. As BOINC relies on users, BOINC resources are not reliable. Therefore, BOINC provides a set of features (checkpointing, digital signature, etc.) to avoid users from cheating, (i.e. [2]).

A. How to use BOINC with a Scientific Project

A scientific project that wants to use BOINC has to set up a GNU/Linux server (Apache, MySQL, PHP) and build a binary for an OS. The binary must be coded in C/C++ or Fortran, thus, two possible methods can be employed to support BOINC:

- **Method 1.** *To port the code*. This is the most used method. Basically, a researcher has to adapt his application source code to support BOINC. The changes could be simple if the tool is coded in C/C++ or Fortran. In other cases the researcher will have to rewrite the whole code.
- **Method 2.** *The Wrapper*. The BOINC team provides a tool called *wrapper* which enables to run statically linked applications inside BOINC without needing to modify or port the application source code. Basically the wrapper embodies the target software in such a way that for the BOINC client does only exist one application: the wrapper.

In conclusion, an application which is not coded in C/C++/Fortran will use the **Method 2**. However, if the application is coded in C/C++/Fortran some minor changes to support BOINC will be needed (**Method 1**).

The first described method can be used when a scientist has to adapt the GP source code like Chávez et al. presented in [8]. In this case the only requirement is the programming language (C++). When a scientist wants to employ a standard GP tool, like for example ECJ[4], there are two options: (i) to port the source code or (ii) to employ the second method (**Method 2**). The porting step could be in some cases difficult or impossible due to time constraints or the complexity of the porting step. Nevertheless, this method can only be used when the applications are statically linked.

The following Section provides a new model that allows a GP environment (and any other tool) to run within BOINC when the previous two methods fail to provide sufficient support.

III. Virtualizing VGC

Very frequently, the complexity of the scientific software is critical. This complexity leads to not being able to port the code to BOINC middleware nor using the previous methods for running applications within the BOINC framework. Consider, for instance, using the Matlab GP toolbox together with other toolboxes or external applications. The complexity is not only related to the different languages employed by the middleware and the GP code, but also to the different

[3]Data obtained from http://boincstats.com under BOINC Combined stats

[4]See http://www.cs.gmu.edu/∼eclab/projects/ecj/

library versions and other software tools required to solve the problem.

For this reason, we propose to employ the virtualization technology (i.e. [5], [16]). Virtualization is a technology which abstracts the real hardware from a computer and creates a new virtual machine (VM). In this VM it is possible to load and run an OS and its applications.

Therefore, what we propose is to employ the virtualization technology when the previously described methods 1 and 2 (see Section II-A) cannot be used.

A. Extending BOINC with VMware Virtual software

As we have stated in Section I, we have chosen VMware as our virtualization technology. The main reasons are firstly that it runs on the same platforms as BOINC does, and secondly because it is free software. The supported platforms are very important, because the DGC tries to harness as much resources as available. Thus, the employed virtualization technology must run, at least, on the same platforms as the DGC middleware does.

As a result of using this new layer, BOINC becomes a *customizable execution environment*. This execution environment can run any scientific application independently of its sources (programming language, complexity, etc.). Thanks to the virtualization layer, BOINC will be able to run a virtual host which will match exactly the same environment as the scientists have for running their experiments. In other words, the scientist has not to take care of the underlying BOINC software, and prepare only an image of his executing environment for running it inside BOINC via VMware.

VMware runs VMs using a set of files which is called an image. Within these files VMware stores the configuration of the VM hardware plus the OS and applications. This set of files will be part of the BOINC project, and will be downloaded as part of the WU (see Section II). To sum up, thanks to this technology the scientist will be only in charge of "photocopying" his running system into a VMware image; an easy step compared to porting or modifying the source code for BOINC.

As we have presented, VMware is free software, but not open source. Hence this leads us to use the second method, described in Section II-A. This method employs the *wrapper* solution, which allows running legacy applications like VMware. However, VMware is not itself statically linked, so the *wrapper* solution is not sufficient. In order to run VMware or any similar legacy application, we have to install firstly the desired target software (the VMware Player software in this case) on the clients and then launch it from *a script or a program*. This new *program* will be in charge of setting up everything for the virtualization environment, and then launching it; for this reason we have named it *starter*. In other words, the *wrapper* solution (**Method 2**) will be used plus the new *starter* program which runs the virtual layer inside BOINC.

From the point of view of the BOINC side, the VMware image files are treated as if they were other input files of the BOINC project[5]. Hence, the BOINC clients will download not only the standard BOINC files for this VMware project (the *starter* and the *wrapper*) but also the image files. Once the BOINC client has downloaded all the necessary files it will start the execution of the whole system. First, the wrapper is initiated and then, through the wrapper, the starter is executed. These steps will launch the VMware software that runs the virtual environment, which in our case means the GP application.

An important legal problem arises by using a virtualization technology inside BOINC. The legality of using BOINC+VMware for running several copies of a licensed product is out of the scope of this paper and research. Therefore, our aim is to present an original technology – BOINC+VMware– to run DGC customizable execution environments.

IV. Experiments and Results

The experiments presented below are aimed at showing that VGC is a useful computing platform for running problems independently of the employed software tool. For accomplishing this purpose, we are going to use different GP problems. It is important to state that we are not interested in analyzing the quality of obtained results, but rather in achieving a good *Computing Power* (CP) for free thanks to the VGC technology. Thus, we provide two test cases: a standard GP framework and a complex GP environment to show that any of them can benefit from this approach.

The achieved performance by a BOINC project is obtained from the following equation presented by Anderson and Fedack [3]:

$$\begin{aligned} CP = X_{arrival} * X_{life} * X_{ncpus} * X_{flops} * X_{eff} \\ *X_{onfrac} * X_{active} * X_{redundancy} * X_{share} \end{aligned} \tag{1}$$

For all the following experiments, $X_{redundancy}$ is equal to 1 because we didn't use the redundancy facility provided by BOINC. X_{share} is also equal to 1 because none of the clients shared its resources with other BOINC projects. $X_{arrival}$ and X_{life} are very important variables due to they measure the *host churn* see Fig. 2 (the volunteer computing project's pool of hosts is dynamic). The rest of the variables measure specific hardware features (i.e. [3]).

The following subsections presents two different scenarios where two different GP tools are used. We show how it is possible to obtain computing power free by no modifying the source code and by employing the DGC BOINC software.

A. A Standard GP Framework

The first experiment considers the case of using a widely adopted GP framework, such as ECJ[6]. The special features of ECJ, a complex JAVA EC framework, allows its use with BOINC altogether thanks to our proposed method, given

[5]The size of the images are not a big deal in the download process, thanks to the modern Internet connections provided by ISPs and the new BitTorrent stack inside BOINC [7].

[6]For further information see http://cs.gmu.edu/~eclab/projects/ecj/

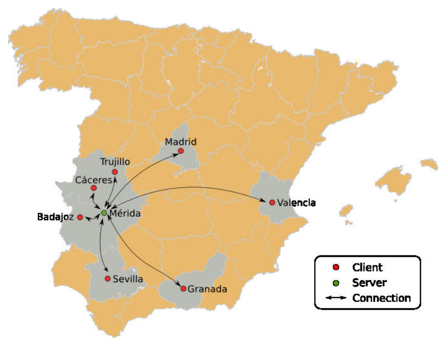

Fig. 1. Distributed infrastructure

TABLE I
OBTAINED RESULTS FOR THE MULTIPLEXER FUNCTION

Bits	Active Hosts	Time	Runs	Time per Run	CP
11	27 of 45	5.35d	828	134.75s	80GFLOPS
20	11 of 145	48d	60	23449.3s	11.08GFLOPS

that it includes its own Virtual Machine, the Java VM. By employing the Java Virtual Machine, it is not necessary to use the VMware VM. Consequently, the ECJ framework runs "natively" on the BOINC clients.

Therefore ECJ is a good first test case given that it is widely used and allows the deployment of a GP tool with BOINC by means of a VM. The second experiment will consider the worst test case: another GP tool which doesn't include the VM, featuring higher complexity in terms of a porting process. The chosen problem to be solved by ECJ was the GP benchmark Boolean Multiplexer function, using the same GP parameters as described in [13]. The goal of the problem is to create a multiplexer function for k address bits. In general, the input to the boolean multiplexer function consist of k address bits a_i and 2^k data bits d_i, which has the form $a_{k-1} \cdots a_1 a_0 d_2^{k-1} \cdots d_1 d_0$ with length equal to $k + 2^k$. Therefore, the search space for this function is equal to 2^{k+2^k}.

This problem has been run in several geographically distributed laboratory clients belonging to the University of Extremadura (Cáceres, Badajoz and Mérida). This scenario was a testbed for the next experiments, due to the environment (network and laboratories) was controlled.

Tab. I shows a summary of the main results for the 11 multiplexer deployment ($k = 3$). The experiment employed the same GP parameters as Koza used [13]. The experiment was running only a few days, 5.35, so X_{life} is measure only from the first connection to the last communication of hosts that had not communicated in at least one day. Hence, the obtained CP is equal to 80 GFLOPS. The obtained CP is large because we have a controlled environment where the computers were active most of the time. Furthermore, the needed time per run was small (134.75s), so the clients always finished their tasks. We increased the complexity of the problem, and therefore the running time, with the 20 multiplexer function ($k = 4$). The GP parameters were the same as for the 11 multiplexer,

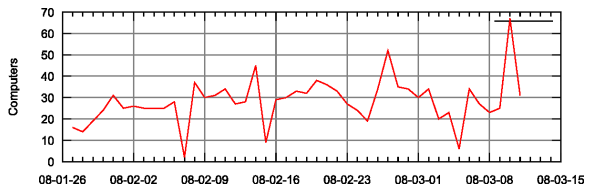

Fig. 2. Host churn – 20 multiplexer function

except for the size of the population: 2000 individuals. Tab. I shows a summary of the relevant data for the 20 multiplexer. As said before, our interest is not in solving the problem, but in setting up a time consuming experiment for testing the VGC model. For this new experiment, volunteer computers (145 in total) from other Spanish Universities or institutions such as: CICA in Sevilla, University of Extremadura (Cáceres, Badajoz, Mérida), Granada, Valencia, UNED in Madrid, and Ceta-Ciemat in Trujillo collaborated with the project. Thus, the computing resources are more heterogeneous and realistic now, see Fig. 1.

X_{life} was adapted again, but in this case taking into account connections from the first communication to last communication for hosts that had not communicated in at least 17 days. Even when we obtained a lower CP because only 11 from the 145 hosts were actively computing for the project –typical VGC behavior– the CP obtained is proportional to active hosts, which would allow to greatly increase it with a larger pool of users collaborating with the project. Fig. 2 shows how the dynamic VGC pool of hosts varies during the active period of this BOINC project. Fig. 3 shows the free available RAM memory and hard disk space.

B. A Complex GP environment

The second experiment considers a more pessimistic scenario –which is frequently the case– with a complex and not statically linked GP tool that makes impossible to employ Method 1 or 2 (see Section II-A). In order to make the experiment to resemble more realistically a standard scientific research project, we are deploying here a real life and time-consuming Computer Vision problem that has already been solved in a sequential fashion (Interest Point detectors (IPGP), see [18] bronze medal at Hummies 2007). This GP framework uses the Matlab environment and several image tool-boxes, which implies a much more complex system, being therefore more difficult to deploy it over a BOINC infrastructure. Hence, we used our new proposed virtualization layer within BOINC. In other words, the researchers create an image of their running environment and set up a BOINC project with it (see Section III). The last requirement for running this new virtualized BOINC, is that all the clients which want to collaborate with the project have to install the VMware Player software in advance. Once this last step is done in all the PCs, the clients can start working with the given project. For testing this experiment we set up 10 Ms Windows volunteer computers. The virtual image was built using a GNU/Linux

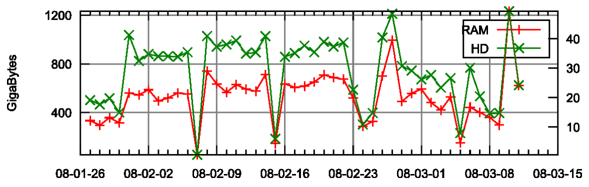

Fig. 3. Available free memory and hard disk space

TABLE II
ACHIEVED COMPUTING POWER

Env.	Type	GP Tool	Problems	Hosts	CP
Labs.	Standard	ECJ	11 Mult.	45	80GFLOPS
	Complex	Matlab	IPGP	10	25.67GFLOPS
Real	Standard	ECJ	20 Mult.	145	11.08GFLOPS

x86 operating system (for further details see [14]). Thus, a GNU/Linux scientific environment runs directly inside Ms Windows thanks to the Virtual-BOINC approach. Moreover, the computer vision problem was not modified at all. The 10 Windows PCs produced 12 solutions during 48 hours. The consumed time by each solution was in average of 18 hours. The total time consumed for producing 12 solutions by a sequential run was 215 hours. Therefore, and using the equation 1 we achieved a CP of 25.67 GFLOPS for free. Tab. II shows a summary of the achieved CP for all the experiments.

In conclusion, the BOINC model obtains for free computing power when time-consuming real life problems are solved by means of GP. Moreover, thanks to the proposed virtualization layer it is possible to run any GP tool, independently of its complexity or programming language, within BOINC providing customizable execution environments free as no modifications are needed. The large number of BOINC users makes the described proposal as a very promising source of CP for GP, as shown, but also for any other EA.

V. CONCLUSIONS

We have presented how it is possible to increase the CP free by means of DGC for GP. We have presented two common scenarios which show how it is feasible to employ a standard tool or a complex system within BOINC. Moreover, we have presented a new methodology, BOINC plus the virtualization layer, which permits to run any available GP tool –statically linked or not– inside BOINC without any modification in the source code. In other words, we have shown how to harness computing resources for a research project for free because: (i) it is not necessary to buy expensive supercomputers or clusters thanks to the DGC and (ii) it is not necessary to modify any source code line. We have also stated the extra benefits of having a controlled environment to obtain the best CP for a GP project. Therefore, this technology can be also used in controlled environments like computer laboratories and try to engage additionally volunteers from the large pool of BOINC hosts. Although tests have been presented for GP, DGC plus virtualization can provide resources for any application which requires large amounts of CP.

ACKNOWLEDGMENTS

This work was supported by Cátedra CETA-CIEMAT Universidad de Extremadura, Regional Gridex project PRI06A223 Junta de Extremadura and National Nohnes project TIN2007-68083-C02-01 Spanish Ministry of Science and Education.

REFERENCES

[1] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. Seti@home: an experiment in public-resource computing. *Commun. ACM*, 45(11):56–61, 2002.

[2] D.P. Anderson. Boinc: a system for public-resource computing and storage. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pages 4–10, 2004.

[3] D.P. Anderson and G. Fedak. The Computational and Storage Potential of Volunteer Computing. *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, 2006.

[4] David Andre and John R. Koza. Parallel genetic programming: a scalable implementation using the transputer network architecture. pages 317–337, 1996.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, 2003.

[6] Forrest H III Bennet, James Shipman John R. Koza, and Oscar Stiffelman. Building a parallel computer system for $18,000 that performs a half peta-flop per day. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1484–1490, Orlando, Florida, USA, July 1999.

[7] F. Costa, L. Silva, I. Kelley, and G. Fedak. Optimizing the data distribution layer of boinc with bittorrent. *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–8, April 2008.

[8] Francisco Chávez de la O, Jose Luis Guisado, Daniel Lombraña, and Francisco Fernández. Una herramienta de programación genética paralela que aprovecha recursos públicos de computación. In *V Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados*, volume 1, pages 167–173, Tenerife, Spain, February 2007.

[9] L. Vanneschi F. Fernández, M. Tomassini. An empirical study of multipopulation genetic programming. *Genetic Programming and Evolvable Machines*, 2003.

[10] G. Fedak, C. Germain, V. Neri, and F. Cappello. XtremWeb: A Generic Global Computing System. *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGRID'01)*, 2001.

[11] Francisco Fernández and Erick Cantú-Paz. Special issue parallel bioinspired algorithms. *Journal of Parallel and Distributed Computing*, 66(8), 2006.

[12] C. Kesselman and I. Foster. The Grid: Blueprint for a New Computing Infrastructure. *Morgan Kaufmann*, 1999.

[13] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, USA, 1992.

[14] Daniel Lombraña, Francisco Fernández, Leonardo Trujillo, Gustavo Olague, and Ben Segal. Customizable execution environments with virtual desktop grid computing. *Parallel and Distributed Computing and Systems, PDCS*, 2007.

[15] J. Basney M. Litzkow, T. Tannenbaum and M. Livny. Checkpoint and migration of unix processes in the condor distributed processing system. Technical report, University of Wisconsin, 1997.

[16] Jason Nieh and Ozgur Can Leonard. Examining VMware. *j-DDJ*, 25(8):70, 72–74, 76, August 2000.

[17] M.E. Samples, J.M. Daida, M. Byom, and M. Pizzimenti. Parameter sweeps for exploring GP parameters. *Proceedings of the 2005 workshops on Genetic and evolutionary computation*, pages 212–219, 2005.

[18] Leonardo Trujillo and Gustavo Olague. Synthesis of interest point detectors through genetic programming. In Mike Cattolico, editor, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2006, Seattle, Washington, USA, July 8-12, 2006*, volume 1, pages 887–894. ACM, 2006.