Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

journal homepage: www.elsevier.com/locate/swevo

Regular Paper

A distributed neuro-genetic programming tool

Marco Russo

Department of Physics and Astronomy, University of Catania, Viale Andrea Doria 6, 95125 Catania, Italy

ARTICLE INFO

Article history:
Received 12 December 2014
Received in revised form
10 June 2015
Accepted 30 October 2015
Available online 23 November 2015

Keywords:
Genetic programming
Neural networks
Distributed computing
Parallel programming

ABSTRACT

This paper describes the performance of the Brain Project, a distributed software tool for the formal modeling of numerical data using a hybrid neural-genetic programming technique. One of the most interesting characteristics of the Brain Project is its distributed implementation. Unlike many other parallel and/or distributed solutions the only requirement of the Brain Project is that the collaborating personal computers must be 64-bit Linux machines connected to Internet via the transmission control protocol/internet protocol. The performance of the Brain Project is clearly enhanced with the very simple parallelization scheme illustrated in the paper. Although the Brain Project presents many innovative solutions for the genetic programming research, this paper focuses mainly on its behavior in the distributed environment.

1. Introduction

Evolutionary Computing (EC), Neural Networks (NNs), and Fuzzy Logic are well-known techniques that are frequently used for Multi-Input–Multi-Output (MIMO) system modeling. These techniques often require a learning phase where the input data set is presented with the aim of identifying the parameters of the final model. This phase is typically very intensive, so, to reduce the learning time, hardware solutions and/or parallel/distributed ones can sometimes be adopted [13,15,18,23,14,12].

Genetic Programming (GP) falls within the EC field [17]. One particular GP line of research foresees the evolution of tree structures that represent formal mathematical expressions. The Brain Project (BP) is just a hybrid Neuro-GP where we have tree structures for the MIMO modeling of experimental data.[1] Of course this kind of GP can also be used in many other domains, for example, for the numerical approximation of formulae [27].

EC is well suited to be parallelized and/or implemented in a distributed environment. Some works report superlinear speedups [23,2,5]. This result is reached, because the speedup is evaluated starting from a unique population In this case, once again in 1943, Wright [25] demonstrated that the problem is resolved more quickly in terms of total computational effort when rearranged on several multiple semi-isolated subpopulations. We can say that the difference between a single population of individuals and the same number of individuals divided into a number of subpopulations maintains the probability of premature convergence and/or super-individual hegemony. In the case of very difficult problems, when we have single population it is quite easy that the population reaches a premature convergence. In the case of multiple subpopulations, the probability that all the subpopulations simultaneously reach a premature convergence with practically all the equal individuals is almost unrealistic. So if a low probability of migration is permitted, one population that has blocked its evolution can begin to improve the individuals again when other different individuals are injected from another subpopulation. This is also confirmed by previous studies [20] where it was demonstrated that migration in the island is an essential model.

Other studies analyze not the speedup reachable comparing the computational effort of only one population and those obtained with many subpopulations, but what happens when, given a number of subpopulations, we increase their number. Unfortunately the results regard either simplified math models or simpler realizations than the BP [19]. The speedup reached in these cases is almost linear only for a small number of islands.

Further studies with more complex implementations [13] do not reach linear speedup.

In this paper despite the complexity of the BP, we have analyzed its behavior with a number of islands that increases up to 160 divided onto up to 5 PCs.

We started to develop the BP in 2008. The current version is number 7.227 and consists of about 15,000 C code lines. It is composed of a set of highly optimised, object-oriented modules for vector processing. Its internal data structures were designed

E-mail address: marco.russo@ct.infn.it
URL: http://superpippo.ct.infn.it/~marco

[1] If you have any data to model formally and you wish to share it for research purposes, do not hesitate to get in touch with us.

http://dx.doi.org/10.1016/j.swevo.2015.10.009
2210-6502/© 2015 Elsevier B.V. All rights reserved.

considering CPUs as Single-Instruction–Multiple-Data (SIMD) processors.

At present, we are using the BP project into two different fields. The former [24] regards short-term solar plant power forecasting without the use of meteorological forecasts or cloud inspection. This work has been founded by “Enel Ingegneria e Ricerca S.p.A” and deals with systems having up to 74-inputs–4-outputs, about 20 thousand patterns and a function complexity of several hundreds nodes. The latter regards the implementation of the nuclear matter equation of state, which is the main ingredient for the study of the neutron star equilibrium configuration. Special attention must be paid when strange baryonic matter is included [8]. In this case we are dealing with 4-input–1-output systems, roughly 1 thousand learning patterns and a function complexity in the order of several hundred nodes.

One of the most salient features of the BP is its distributed implementation. As regards this aspect the only requirement is the Transmission Control Protocol/Internet Protocol (TCP/IP). Neither the Message Passing Interface nor the Parallel Virtual Machine nor any other overhead is needed [23,4]. A computer with a 64-bit Linux operating system connected to the Internet is also necessary, even if does not have a public IP.

The parallelization architecture of the BP follows a Master–Slave scheme [12]. Essentially, each time we have some data to model (we call this a Learning Task (LT)), we run one instance of the master software on the Personal Computer (PC) master (that has a public IP address). After a few seconds, all the active software slaves, that are in sleep mode on all the PC hosts collaborating with our research, wake up and start to work on the LT. When the LT finishes all the slaves return to sleep. Even if not described in detail in this paper the BP is able to handle multiple LTs.

Here we will show that the solutions behind our approach permit very complex modeling-problems to be dealt with. But the primary result obtained in these cases is that we can use many hosts thus reducing the learning time significantly. More precisely we can say that when we increase the complexity of the LT we can use more hosts and attain a computation time that reduces linearly with the total computation power we have at our disposal. In other words, we can reach ideal speedups if we have enough complex problems, taking into account the total computation power of all the available hosts. Even if in this paper we will show our results using only five hosts, we have tested the BP using several tens of hosts kindly offered by the no-profit research consortium COMETA [1] that performs research in the field of High Performance Computing. Our software slaves are able to run on any 64-bit Linux system and are able to adapt their CPU and RAM requirements dynamically to the actual available load and the memory subsystem. For this reason in this paper, even if we have many PCs at our disposal, we decided to use only the five equal, most powerful PCs to run our software slaves.

After a brief illustration of the way BP was developed, this work makes an in-depth analysis of its behavior in the distributed environment.

Section 2 provides a brief illustration of the software and hardware components of the BP. Then, Section 3 clarifies some key points regarding the genetic and neural choices adopted. Afterwards, in Section 4 an explanation is given for the learning procedure implemented. Subsequently, Section 5 reports an accurate study of the BP performance when both the number of PCs and the learning time were changed. Section 6 presents an interesting comparison of speeds. Finally, the last section reports our conclusions.

2. The main software and hardware components of the Brain Project

2.1. The three BP software components

The BP is composed of the following one script and two executables:

- A bash script called **demone**.[2] Only one **demone** runs for each remote PC.
- An executable named **brain_server** which is launched one or more times when there are data to model. Each different modeling experiment will be called Learning Task. We have one **brain_server** for each LT running at the same time.
- Another executable named **brain_client** running remotely. Many **brain_clients** normally run on the same PC.

To make the paper more readable we will call a running instance of **brain_client** simply client. Similarly, with the term server we will indicate a running instance of **brain_server** software. One generic PC where there is a running client will be called Client PC (CPC) or simply host, while in the case of the server we talk of the Server PC (SPC). Obviously nothing can avoid CPC and SPC matching for certain clients. At the moment, the SPC hostname is **superpippo.ct.infn.it**.

Fig. 1 shows an example of all three BP software components. In this figure we can see four CPCs where for each only one **demone** and a different number of clients are running. Furthermore, there is only one SPC in which the 3 LTs, called gravity, photovoltaic, and coulomb are running contemporarily. One server is depicted for each LT. We underline that each client contributes to only one LT and that on each host different LTs can run concurrently.

2.1.1. The script ***demone***

This is just a bash script that performs several different tasks.

- First, it discovers if there is a more recent version of itself on the SPC. If so, it downloads and after re-executes itself.
- The script also downloads, again from the SPC, the correct client version according to the CPC instruction set.
- After, the **demone** launches several instances of the client in background, at very low priority.
- One other important task of the **demone** is to limit the number of instances when an excess of host computer memory usage occurs. The script always gives more priority to all the other CPC running processes.

All these operations are executed continuously. So, for example, if we compile a newer version of the client, then the **demone** kills, downloads and re-executes all of them.

2.1.2. The server

The **brain_server** is launched when we have any new LT. For each running server one Configuration File (CF) is foreseen. It gives instructions for performing the LT.

Each server performs several tasks. We now briefly list some of them.

[2] If you want to collaborate as a volunteer offering your idle CPU cycles you can download the **demone** and execute it in background. For example, you can first write: **wget superpippo.ct.infn.it/ marco/demone**. After, you set the script execution bit: **chmod +x demone**. Lastly, you execute it: **nohup**./**demone** > /**dev**/**null2** > /**dev**/**null** &. To kill it: **pkill −9 demone**.