2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology

Parallel simulation of NEPs on clusters

Carmen Navarrete Navarrete, Marina de la Cruz Echeandía,
Eloy Anguiano Rey, Alfonso Ortega de la Puente
Departamento de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Madrid, Spain
carmen.navarrete@uam.es

José Miguel Rojas Silas
Departamento de Lenguajes y Sistemas Informáticos
e Ingeniería de Software
Universidad Politécnica de Madrid
Madrid, Spain
josemiguel.rojas@upm.es

***Abstract*—[1] This paper compares two different approaches, followed by our research group, to efficiently run NEPs on parallel platforms, as general and transparent as possible. The vague results of jNEP (our multithreaded Java simulator for multicore desktop computers) suggests the use of massively parallel platforms (clusters of computers). The good results obtained show the scalability and viability of this last approach.**

***Keywords*-Natural computing, NEPs, simulation, Parallel computing, clusters**

I. Motivation

A great deal of research effort is currently being made in the so called "natural computing" realm. "Natural computing" is mainly focused on the definition, formal description, analysis, simulation and programming of new models of computation (usually with the same expressive power as Turing Machines) inspired by Nature. Their bio-inspired nature makes these models specially suitable for the simulation of complex systems.

These models could be used as new architectures for computers, different from von Neumann's machine.

This paper focuses on NEPs (Networks of Evolutionary Processors), one of the best known natural computers for which a great research effort is currently being made.

Several attempts have been made to build hardware devices to support some bio-inspired models, but nothing has been done for NEPs.

One of the most interesting features of these bio-inspired computers is their intrinsic parallelism. We can design algorithms for them that could improve the exponential performance of their *classic* versions. Nevertheless, when the models have to be simulated on conventional computers, the total amount of space needed to simulate the model and to actually run the algorithm usually becomes exponential. This may be one of the main reasons why natural computers are not widely used to solve real problems. Most of the simulators are not able to handle the size of non trivial problems. Grid, cloud computation and clusters offer an interesting and promising option to overcome the drawbacks of both solutions:"specific" hardware and simulators run on von Neumann's machines.

This paper introduces a new approach to the parallel simulation of NEPs on clusters of computers. Our approach includes a general middleware that we plan to use to simulate (also in parallel) other natural computing models.

[1]This work was partially supported by the R&D program of the Community of Madrid (S2009/TIC-1650, project "e-Madrid").

II. Introduction

A. Introduction to NEPs

NEPs [2] is a new computing mechanism directly inspired by the behaviour of cell populations. A NEP can be defined as a graph whose nodes are processors which perform very simple operations over strings sending them again to other nodes. The operations included in the basic model are well defined as follows: erasing and adding a symbol, replacing a symbol with other, and splicing two strings. Every node has filters that block some strings from being sent and/or received. As computing devices, NEPs alternate evolutionary and communicating steps, until a predefined stopping condition is fulfilled. All the processors change their contents at the same time in each evolutionary step. In the communicating steps, the strings that pass the corresponding filters are written on the graph and the strings, that pass their input filters to build their contents for next steps, are taken from the graph. Further details and formal treatment can be found in [2].

B. Introduction to parallel computing

Parallel computing is a form of computation in which many calculations are carried out simultaneously. It uses multiple processing elements simultaneously to solve a problem. This is accomplished by breaking the problem into independent parts (each one is called subdomain or partition) so that each processing element can execute its part of the algorithm simultaneously with the others.

Perhaps the most popular general approach to parallel algorithms is the master/slave type of organization. In these

978-0-7695-4513-4/11 $26.00 © 2011 IEEE
DOI 10.1109/WI-IAT.2011.131

multiple-tier applications, a single node (or more) functions to organize and disseminate the relatively separate tasks of the overall composite problem, and (optionally) to collect and/or reassemble the individual results into a single integrated answer or product. The class of nodes actually receiving and processing the smaller component tasks represent another specialized tier of this hierarchical approach. More than two tiers of organization are also possible. A single tier of "slaves", all simultaneously running serial code with absolutely no inter-communication, can be viewed as a specialized form of this approach. But two levels of organization, often with a single "master" node, is the most common configuration. Strategies for providing and optimizing load-balancing across multiple slave nodes within heterogeneous parallel environments is of general significance across a wide array of problems.

Because communication and synchronization between the different subtasks and nodes of the cluster are typically one of the greatest obstacles to getting a good parallel program performance, parallel computer programs and algorithms are more difficult to implement than sequential ones. But only a good management of the communication and synchronization will not achieve the best performance of the parallel algorithm; the loadbalancing and the domain decomposition technics have also much to do there.

The most noteworthy idea of the parallel computing is to decompose the problem into subproblems that are easier to solve; that is, the *Divide and conquer* philosophy. But, it is needed to consider that depending on the solution taken to decompose the problem into at least as many domains as processes, we will obtain a better or worse performance and therefore, a better use of the resources of the cluster [9]; choosing an inappropriate domain decomposition will affect the speed-up of the parallel solution, but the domain decomposition depends both on the problem that we want to execute in the cluster as well as on the symmetries of it. Thus, the idea is to develop a generic platform for the execution of existing sequential codes, so that the parameters that optimize the application performance in the cluster (as network and data topologies or domain decomposition...) will be dynamically obtained and simultaneously with the execution of the algorithm. In general, the domains decomposition algorithm must take into account the problem properties and symmetries and must change them if the speed-up decreases.

Although we have used this framework to run the NEP application in parallel, in a real cluster, the framework is not limited to this kind of application and it can be run to solve with several applications.

III. Methodology

In order to test the performance of clusters of computers when running in parallel NEPs, we have designed a family of graphs to solve several instances of the Hamiltonian Path problem (HPP). [4] shows how the HPP can be solved by means of NEPs with a lineal (temporal) performance. Although to reach this bound is not our goal, this proof will give us useful hints with respect to improve the performance of the simulation of NEPs on non parallel hardware platforms.

A. Hamiltonian path problem solution by NEPs

This well-known NP-complete problem searches an undirected graph for a Hamiltonian path, that is, the one that visits each vertex exactly once.

It is possible to solve this problem by means of the following NEP: The NEP graph is very similar to the one studied: an extra node is added to ease the definition of the stopping condition. Let n be the number of nodes of the graph under consideration 1. Let $\{v_j, 0 \leq j \leq n\}$ the set of processors of the NEP. The set $\{i, 0, 1, ..., n\}$ is used as the alphabet. Symbol i is the initial content of the initial node (v_0). Each node (except the final one) adds its number to the string received from the network. Input and output filters are defined to allow the communication of all the strings that have not yet visited the node. The input filter of the final node excludes any string which is not a solution. It is easy to image a regular expression for the set of solutions (those words with the proper length, the proper initial and final node and where each node appears only once). The NEP basic model allows defining filters by means of regular expressions.

B. Family of graphs

Our goal is to check the cluster performance when solving the HPP for graphs of increasing difficulty. We have used a familiy of graphs with n nodes. 0 is the label of the initial node. Each node is connected with the four closest nodes. That is, the node i is connected with the set of nodes $\{i-2, i-1, i+1, i+2\}$ There is a special case. When defining the NEP to solve this instance, we have to add the output node. The highest label is given to this node $(n+1)$. The output node is only connected with the final node of the graph under consideration (n). The other connections of the output node are removed.

Figure 1 shows this circumstance and the graph for $n = 6$.

This paper compares two approaches that our research group has followed to run NEPs on parallel platforms: 1) a multithreaded simulator for desktop computers (possibly parallel) 2) a massively parallel architecture (clusters of computers)

C. Multithread platform architecture

jNEP (a complete description can be found in [3]) is a multithreaded Java simulator for NEPs. That is, it could be run in parallel if the underlying system were able to distribute the threads among different processors. In this paper, we have performed a set of experiments in a multicore desktop computer with these characteristics.

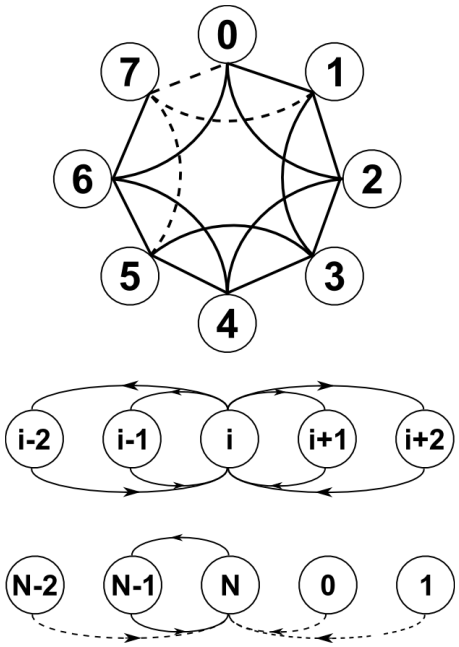

Figure 1. Example of a NEP with n=6 and the extra one to collect the strings.

D. Parallel platform architecture

We can consider this platform as a framework that works on both sides, master and slaves, and allows to execute sequential code in a cluster, taking the advantages of the workload and dynamic domain decomposition concepts, without rewriting the NEP code to adapt it this parallel platform. This framework is implemented in ANSI C++ language, uses the MPI-II ([10]) extensions and its design follows the master-slave model.

The problem that will be solved on the cluster, can be modeled as a weighted and directed graph G_a, denoted by $G_a(T, D, \omega)$; T denotes a set of vertices of the graph that represents the tasks to be done; D represents a finite set of edges of the graph; each vertex has a computation weight ω that represents the amount of computations required by the a task to accomplish one step of the algorithm. The existence of an edge between vertex A and vertex B means that, to calculate the value of A at a certain instant of the execution, we need the value of B at the previous step of the algorithm. We say that A has a data dependency with B.

Both master and slave processes work with graph stuctures so, the master translates the information given by the user into a graph and descomposes this one into several domains that are sent to the slaves. The slaves receive the domains and translate them again into a graph. To transfer the information through the net, both processes must be able to serialize and deserialize the information of the graph, that is, binarize the user defined data structure that allocates the data of each vertex of the graph.

Time(s)	Processes				
n	2	3	5	9	17
NEP12	5	4	4	5	-
NEP16	6	6	5	4	6
NEP20	17	14	13	9	9
NEP24	103	90	83	79	74

Mem.(Mb)	Processes				
n	2	3	5	9	17
NEP12	-	-	-	-	-
NEP16	4	9.8	5.2	3.9	11
NEP20	106-7	135.5	19.3	9.8	21.3
NEP24	842.9	930.4	1445.5	200.1	153.6

Table I
EXECUTION TIME VS. NUMBER OF PROCESSES TABLE AND MEMORY CONSUMPTION VS. NUMBER OF PROCESSES.

The kernel method allows the user to execute its specific algorithm on the slave process. The user does not have to take care about the communication and synchronization with the master process and it neither knows how many slaves have joined the simulation/resolution nor the loadbalancing and partition algorithm used.

By the use of plugins, the behaviour of the cluster (cluster configuration and loadbalancing policies) and of the problem (problem configuration and domain decomposition method) can be modeled. The system provides several plugins but also the user can define its own plugins.

IV. RESULTS

We have performed two sets of experiments: on a conventional multicore architecture and on a massively parallel platform. Our goal is to compare the performance of both systems, giving them the same resources.

For the first set of experiments we have used a multithreaded multicore platform (a desktop computer running a Linux kernel 2.6.26, with 16Gb of memory and $4 \star 6$ cores Intel(R) Xeon(R) CPU E7450 2.40GHz) running a Java multithreaded simulator for NEPs, developed by our research group. The jNEP platform was able to solve graphs up to 8 nodes whereas the biggest graph solved by our parallel framework had 24 nodes.

The results for the second set have been obtained by executing a sequential NEP kernel in a parallel environment ([11] HLRB II, 9728 cores, 4Gb memory per core) using the described framework. The simulation has been executed with NEPs of several number of nodes, from $n = 16$ (more or less $4x10^3$ valid strings) up to $n = 24$ ($5x10^5$ strings). From $n = 28$ and higher values, the assigned resources reached the limit. To observe the framework behaviour the number of slaves has been changed, from 2^0 (equivalent to single processor) to 2^4. It is not possible to have 2^5 or more slaves, because this exceeds the number of vertex of the NEP. That is the reason for our limited testbench.

From the point of view of the execution time, we can observe that the performance of the algorithm has not been

worsened by the use of the framework (see table I and fig. 2). It can be also observed that the execution time decreases until a certain value, that depends on the number of processors and on the dimension of the problem, is reached. Once this point is exceeded, if the number of processors still grows, the execution time will start growing again, just because the master spends more time on the management of the communication, processes and domains than the slaves on the real calculus of the problem.

From the point of view of the memory consumption, we can observe a similar behaviour (see table I); we can find an optimal value of memory that depends on the number of processes and on the number of nodes of the NEP. Once this point is reached, if the number of slaves is incremented, the amount of memory needed to solve the HPP will also increase. As long as more slaves join the simulation, the number of domains will grow lineally and therefore, to fulfill the data dependencies between domains, the information will be more and more replicated among the cluster, i.e we need more memory to allocate the network buffers. On the other hand, increasing the number of domains implies and increment of the number of frames sent by the slaves to the master process. In summary, increasing the number of slaves leads to increase the number of dataframes and the size of each one.

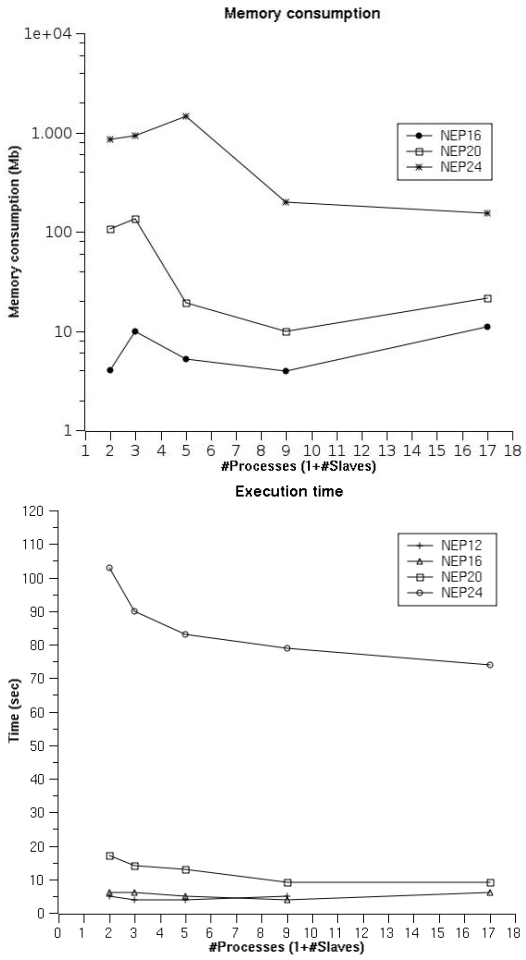

Figure 2. a) Execution time running a secuential NEP algorithm in parallel using the explained framework. The performance of the application has improved running the NEP algorithm in comparison with the single slave execution. b) Semilog plot for the memory consumption running the NEP algorithm under the framework.

REFERENCES

[1] J. Castellanos, C. Martín-Vide, V. Mitrana, and J.M. Sempere. Solving NP-complete problems with networks of evolutionary processors. In *Connectionist Models of Neurons, Learning Processes and Artificial Intelligence : 6th International Work-Conference on Artificial and Natural Neural Networks, IWANN 2001 Granada, Spain, June 13-15, 2001, Proceedings, Part I*, pages 621–, 2001.

[2] J. Castellanos, C. Martín-Vide, V. Mitrana, and J. M. Sempere. Networks of evolutionary processors. *Acta Informatica*, 39(6-7):517–529, 2003.

[3] del Rosal, E. del, Nuñez, R., Castañeda, C., Ortega, A., 2008. "Simulating NEPs in a cluster with jNEP" *International Journal of Computers, Communications & Control.* Supplementary Issue: Proccedings of ICCCC 2008. Vol. III, 480-485

[4] del Rosal, E., Rojas, J.M., Núñez, R., Castañeda, C. and Ortega, A. On the solution of NP-complete problems by means of JNEP run on computers. In Proceedings of *International Conference on Agents and Artificial Intelligence (ICAART 2009)*, pages 605-612, Porto, Portugal, 19-21 January 2009. INSTICC Press.

[5] Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco (1999)

[6] Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman, C.: Grid Information Services for Distributed Resource Sharing. In: 10th IEEE International Symposium on High Performance Distributed Computing, pp. 181–184. IEEE Press, New York (2001)

[7] Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid: an Open Grid Services Architecture for Distributed Systems Integration. Technical report, Global Grid Forum (2002)

[8] National Center for Biotechnology Information, http://www.ncbi.nlm.nih.gov

[9] A. Lastovetsky and R. Reddy, Data Partitioning with a Realistic Performance Model of Networks of Heterogeneous Computers, Scientific Programming, vol 2, pp. 93–112 (2005)

[10] Message Passing Interface Forum, MPI: A Message-Passing Interface Standard, University of Tennessee, UT-CS-94-230 (1994)

[11] Hochleistungsrechenszentrum Bayern, http://www.lrz.de