Parallel Software Architectures Analysis for Implementing P Systems

Luis Fernández
Natural Computing Group
Universidad Politécnica de Madrid
Madrid (Spain), Campus Sur. 28031

Fernando Arroyo
Natural Computing Group
Universidad Politécnica de Madrid
Madrid (Spain), Campus Sur. 28031

Iván García
Natural Computing Group
Universidad Politécnica de Madrid
Madrid (Spain), Campus Sur. 28031

Abraham Rodriguez
Natural Computing Group
Universidad Politécnica de Madrid
Madrid (Spain), Campus Sur. 28031

Abstract

This paper presents software processes architectures for implementing P systems on parallel hardware architectures. In order to analysis three different software architectures present in literature, two parameters are established: parallelisms degree and computational overload. This parameters will allow us to compare the three different studied software architectures. Finally, this analysis determines the candidate software architecture depending on available hardware architecture and the P system structure.

1 Introduction

P systems are a new computational model based on the membrane structure of living cells [1]. This model has become, during last years, a powerful framework for developing new ideas in theoretical computation. P systems with simple ingredients are Turing complete. In particular, P systems are a class of distributed, massively parallel and nondeterministic systems. "As there do not exist, up to now, implementations in laboratories (neither in vitro or in vivo nor in any electronic medium), it seems natural to look for software tools that can be used as assistants that are able to simulate computations of P systems" [2].

There are many simulators, but "the next generation of simulators may be oriented to solve (at least partially) the problems of storage of information and massive parallelism by using parallel language programming or by using multiprocessor computers" [2]. "Several authors have implemented the first versions of simulators based on parallel and distributed architectures, which is close to the membrane computing paradigm" [2]. From them, we emphasize the following ones: Ciobanu presents an implementation based on a computer cluster that "consists of 64 dual processor nodes. [...] The rules are implemented as threads" [3], this implies that a one to one relationship is established between processes and evolution rules, we refer his parallel software architecture as *evolution rules oriented*; Syropoulos presents a simulation that "is characterized as distributed in the sense that a number of objects -modeling membranes- execute code on different machines" [4], we will refer this as parallel software architecture as *membrane oriented*; finally, Tejedor presents in [5] a new parallel software architecture in which only exists one process per processor establishing a one to one relationship between processes and processors, we refer to it as *processor oriented*.

The aim of this work is to determine the appropriate parallel software architecture for a given P system and a hardware architecture. So, it pretends to determine the set of process and their relationships (parallel software architecture, PSA) that are appropriate to be executed over a set of connected processors (hardware architecture). In order to do it, it is necessary to establish the evaluation parameters for the analysis of parallel software architectures independently of software and hardware technologies.

This paper is structured as following: first, they are established the evaluation parameters for the analysis of a given parallel software architecture; the three following points evaluate the different parallel soft-

ware architectures in bibliography; next, it is presented a comparative of the three previous architectures through the established evolution parameters; finally, we present our conclusions.

2 Evaluation Parameters

Evaluation of parallel software architecture must be independent of particular hardware technology in which it is implemented on. This abstraction process is only dependent on the number of processors available on the hardware architecture; we will refer to this number as attribute P from now on. As first parameter for evaluating a parallel software architecture, we define the parallelism degree, PD_{PSA} as follows:

$$PD_{PSA} = \frac{P}{R} \cdot 100 \tag{1}$$

where R is the number of evolution rules. The range of values for PD_{PSA} belongs to $(0 - 100]$.

On the other hand, each process will be executed sequentially on its own processor. But, in case of there is less processors than processes, multiprogramming technique gets concurrency of several processes assigned to a same processor but it does not get real parallelism. However "while the general motivation for demand-driven concurrency is laudable, the implementations [...] may not produce optimal results. To understand why, we must consider the subtleties of process creation and scheduling" [6]. Consequently, in multiprogramming cases, it is required a computational overload of the operating system to manage processes. Process management costs, $\Delta_{PM}(N, P)$, is increased when the number of processes, N, increase keeping the same number of processors, P. Hence,

$$\Delta_{PM}(N_a, P) < \Delta_{PM}(N_b, P) \Leftrightarrow N_a < N_b \tag{2}$$

Moreover, concurrent and/or parallel processes interrelate between each other by two types of behaviour: cooperating and/or competing. These relationships require synchronization and communication interactions between processes. At the end, these interactions become conditional and barrier synchronizations and mutual exclusions when there are shared resources. These interactions fall into a new computational overload. In particular, costs of conditional synchronization, $\Delta_{CS}(N)$, and barrier synchronization, $\Delta_{BS}(N)$, are linear dependent on the number of processes while costs of mutual exclusion $\Delta_{ME}(N)$ do not depend on the number of processes. Therefore, we get following equations:

$$\Delta_{CS}(N_a) = k.\Delta_{CS}(N_b) \; where \; N_a = k.N_b \tag{3}$$
$$\Delta_{BS}(N_a) = k.\Delta_{BS}(N_b) \; where \; N_a = k.N_b \tag{4}$$

Then, our second evaluation parameter for the parallel software architecture is defined as computational overload, $\Delta_{PSA}(N, P)$, being N the number of processes and P the number of processors, this parameter is expressed by.

$$\Delta_{PSA}(N, P) = \Delta_{PM}(N, P) \; + \; \Delta_{CS}(N) \; + \; \Delta_{ME}(N) \; + \; \Delta_{BS}(N) \tag{5}$$

Let C be the sequential execution cost of a P system on a monoprocessor hardware architecture. Then it could be define the performance of a parallel software architecture with N processes on a hardware architecture with P processors by the following equation:

$$\frac{C + \Delta_{PSA}(N, P)}{P} \tag{6}$$

Next three sections are devoted to the analysis of evaluation parameters over the three parallel software architectures mentioned above.

3 PSA Evolution Rules Oriented

First architecture establish one process per evolution rule of the P system. Therefore, the number of processes is R and each one of them is responsible for applying one evolution rule over the multiset of its membrane. So, parallelism is given when each one of the R processes progresses over the membrane multiset.

This parallel software architecture reaches a parallelism degree equal to 100% when $P = R$. But when $P < R$, it does not exists a total parallelism because, necessary, it is made use of multiprogramming techniques. Thereby, its parallelism degree decays to $P/R \times 100$ %. Moreover, computational costs of the P system evolution is overloaded by multiprogramming of R processes over the P processors, that we will refer as $\Delta_{PM}(R, P)$. If $P << R$, this overload degrades the system because of $\Delta_{PM}(R, P) \rightarrow \infty$ as these architectures "do not bound concurrency are a risk in an environment that presents a heavy load. Concurrency can increase until operating system

becomes swamped with processes" [6].

On the other hand, algorithms for evolution rules application over a membrane multiset gathered in literature [2] always answer to the same scheme: "Since many rules are executing concurrently and they are sharing resources, a mutual exclusion algorithm is necessary to ensure integrity" [3]. So, "when more than one rule can be applied in the same conditions, the simulator randomly picks one among the candidates" [2]. Therefore, processes include pre-protocols and post-protocols for critical sections of their code that necessary must work under mutual exclusion. We will refer to this overloaded computational cost as $\Delta_{ME}(R)$.

At this point, it has to be highlighted one aspect of synchronization between processes that is determinant for the parallelism degree of this architecture: granularity of critical sections referred above. Thus, we have coarse-grained critical sections when majority of the process of evolution rule application over its multiset is located in a critical section and fined-grained critical sections otherwise.

For the case of coarse-grained critical sections, concurrent execution of subsets of processes of rules of a same membrane offers a sequential behavior. Therefore, repercussion of coarse-grain critical sections in processes implicates in a direct way that parallelism degree is reduced to a $M/R \times 100$ %, where M is the number of membranes.

Last, we would like to highlight the following synchronization between processes and the communication phases in the P system evolution:

- A barrier synchronization that ensures that every process ends its corresponding evolution rule application before that it begins the communication of current evolution transfer.
- A conditional synchronization that ensures that every process waits to apply its evolution rule until it ends the communication phase of the previous evolution step ends.

So, it has to be considered a new computational overload that ensures these synchronizations for the R processes, that we will refer $\Delta_{CS}(R)$ and $\Delta_{BS}(R)$ respectively.

Figure 1 summarizes the parallelism degrees presented in this point for parallel software architecture evolution rule oriented. Thus, it brings in as determinant parameters: number of processors and critical section of process granularity. Moreover, on every case, it is attached the computational added costs to the evolution of the P system.

4 PSA Membranes Oriented

In this architecture it is established one process per membrane of the P system. Therefore, the number of processes is M and each one of these processes is responsible for applying sequentially every evolution rule of a membrane over the multiset of its region. So, parallelism is reached when each one of the M processes progresses over the multiset and the evolution rules of its membrane.

This architecture is bounded by a degree of parallelism equal to $M/R \times 100$ % when $P = M$. But, when $P < M$, it is necessary to use multiprogramation techniques , in a similar way to the architecture presented above. In similar manner, its parallelism degree decay to $P/R \times 100$ %. Moreover, computational costs of the P system evolution are overloaded with the multiprogramming of M processes over the P processors, that we will refer as $\Delta_{PM}(M, P)$. In the case of de $P << M$, this overload degrade the system due to $\Delta_{PM}(M, P) \to \infty$.

On the other hand, sequential algorithms for the application of evolution rules does not use concurrent access to shared information because only one process applies every evolution rule over the multiset sequentially.

Finally, synchronizations between processes and communication phases in the evolution of the P system are the same than in the previous architecture: barrier synchronization respect to the communication of the current evolution step and conditional synchronization respect to the communication of previous evolution step. So, it is also necessary to consider the computational overload that ensures these synchronizations for the M processes, that we will refer as $\Delta_{CS}(M)$ and $\Delta_{BS}(M)$ respectively.

Figure 1 summarizes the parallelism degrees presented in this point of membrane oriented parallel software architecture. So, it brings in just one parameter: the number of processors. Moreover, on every case, it is attached the computational added costs to the evolution of the P system.

Granularity of critical section	Number of Processors	PD_{PS4}	Δ_{PS4}	
Parallel Software Architecture Evolution Rules Oriented				
Fine Grain	$P = R$	100 %	$\Delta_{CS}(R) + \Delta_{MB}(R) + \Delta_{BS}(R)$ $\Delta_{PM}(R,P) \to 0$	(A)
	$P < R$	P / R x100 %	$\Delta_{CS}(R) + \Delta_{MB}(R) + \Delta_{BS}(R) + \Delta_{PM}(R,P)$	(B)
	$P << R$	0 %	∞ $\Delta_{PM}(R,P) \to \infty$	(C)
Gross Grain	$P = R$	M / R x100 %	$\Delta_{CS}(R) + \Delta_{MB}(R) + \Delta_{BS}(R)$ $\Delta_{PM}(R,P) \to 0$	(D)
	$P < R$	min $(P,M) / R$ x 100 %	$\Delta_{CS}(R) + \Delta_{MB}(R) + \Delta_{BS}(R) + \Delta_{PM}(R,P)$	(E)
	$P << R$	0 %	∞ $\Delta_{PM}(R,P) \to \infty$	(F)
Parallel Software Architecture Membranes Oriented				
	$P = M$	M / R x 100 %	$\Delta_{CS}(M) + \Delta_{BS}(M)$ $\Delta_{PM}(M,P) \to 0$	(G)
	$P < M$	P / R x 100 %	$\Delta_{CS}(M) + \Delta_{BS}(M) + \Delta_{PM}(M,P)$	(H)
	$P << M$	0 %	∞ $\Delta_{PM}(M,P) \to \infty$	(I)
Parallel Software Architecture Processors Oriented				
		P / R x 100 %	$\Delta_{CS}(P) + \Delta_{BS}(P)$	(J)

Figure 1: Parallel Software Architecture Analysis

5 PSA Processors Oriented

Last proposed architecture establishes a process per available processor in hardware architecture. Therefore, every one of these processors is responsible for applying sequentially every evolution rule over the multisets of a subset of membranes. These subsets of membranes assigned to processes define a wrapper over the set of membranes. Hence, parallelism is reached when each one of the P processors progresses over the multisets and the evolution rules of the membranes of each subset.

This architecture is bounded by a parallelism degree equal to $P/R \times 100$ %. On the other hand, it has to be highlighted that it does not exist computational overload for processes multiprogramming, because it exists a one to one ratio a priori established. "In fact, because a single-process implementation requires less switching between process contexts, it may be able to handle a slightly higher load than an implementation that uses multiple processes" [6]. In an analogous manner to the previous architecture, given the sequential character of the processes of evolution rules application, neither exist computational overload for the mutual exclusion.

Last, synchronizations between processes and communication phases are the same as previous architectures. So, it is also necessary to consider the computational overload that ensures these synchronizations for the P processes, that we will refer as $\Delta_{CS}(P)$ and $\Delta_{BS}(P)$ respectively.

Figure 1 summarizes the parallelisms degree presented in this point of processors oriented parallel software architecture. Moreover, on every case, it is attached the computational added costs to the evolution of the P system.

6 PSA Comparative

In this point, we show a comparative of the three parallel software architectures presented above. In order to this, it is presented a detailed study of parallel software architectures behavior respect to the different states that a P system evolution can be found. So, considering the relation of P with M and R as the determinant condition, following cases are contrasted: a) $R \leq P$; b) $M \leq P < R$; c) $P < M$. Subsequently, obtained general conclusions are presented:

A. $R \leq P$. On evolution rules oriented architecture it is obtained the same computational overload in (A) and (D) of figure 1, but with a lesser parallelism degree with coarse-grained critical sections. This fact, lead us to avoid evolution rules oriented architecture with coarse-grained critical sections.

   On the other hand, in membranes oriented and processors oriented architectures, it is obtained the same parallelism degree and computational overload in (G) and (H) of figure 1. This is due to, being $R \leq P$ and $M < R$, processors oriented

architecture makes responsible for a subset of an only one membrane to each process. Which is an analogous situation to the membrane oriented architecture.

From (3), (4) and (6) equations and (A) and (G) equations of figure 1, it is obtained the necessary condition (7) for assuring that processors oriented architecture with fined-grained critical section has better performance than membranes and processors oriented architectures.

$$C > \frac{M}{R-M} \cdot \Delta_{ME}(R) \tag{7}$$

In case (7) is not fulfilled, membranes and processors oriented architectures will give a better performance despite of their lesser parallelism degree.

B. $M \leq P < R$. In this second case, we find same situations than the point above: the convenience of fined-grained critical sections over coarse-gross granularity in evolution rules oriented architecture. Equivalent results for membranes and processors oriented architectures have been obtained.

From (3), (4) and (6) equations and equations (B) and (G) from figure 1, it is obtained the new necessary condition (8) for assuring that processors oriented architecture with fined-grained critical sections has better performance than membranes and processors oriented architectures.

$$C > \frac{M}{P-M} \cdot (\Delta_{PM}(R,P) \ + \ \Delta_{ME}(R)) + \frac{R-P}{P-M} \cdot (\Delta_{CS}(M) \ + \ \Delta_{BS}(M)) \tag{8}$$

It occurs the same that in previous case, if (8) is not fulfilled, parallel software architectures membranes and processors oriented will give a better performance despite of their lesser parallelism degree.

C. $P < M$. In this third case, it can be noticed that every parallel software architecture offers the same parallelism degree. But, considering that $P < M < R$ and so it is $\Delta_{CS}(P) < \Delta_{CS}(M) < \Delta_{CS}(R)$ and that $\Delta_{BS}(P) < \Delta_{BS}(M) < \Delta_{BS}(R)$, lesser computational overload is always obtained with processors oriented architecture.

Moreover, in the most restrictive case in which $P << M < R$, computational overload of $\Delta_{PM}(M,P)$ and $\Delta_{PM}(R,P)$ degrade the system.

Detailed study permits us to conclude that:

- The parallel software architecture membranes oriented always has a worse behavior than processors oriented. In particular, it is observed that membrane oriented is a particular case of processors oriented architecture where the membranes subset assigned to each processor is always equal to one membrane. Therefore, flexibility of processors oriented architecture allows suiting to different conditions always balancing or improving to the other more restrictive architecture.
- In cases in which evolution rules oriented architecture is suitable, it always need parallel algorithms of rules application with fined-grained critical sections. Otherwise, it never raises as a parallel software architecture candidate.
- Most appropriate parallel software architecture for a given hardware architecture, it is not always the one with a better parallelism degree. For cases where $P > M$, it has to be fulfilled (7) or (8) conditions.
- Moreover, for these cases, for a given hardware architecture there will not be only one suitable parallel software architecture. It also depends on the number of evolution rules and membranes of the P system. Therefore, combination of a hardware architecture and a particular P system will determine the parallel software architecture suitable for its implementation.

It is important to remarks the goodness of parallelism opposite to totally sequential implementation. "In particular, one must consider the cost of concurrency as well as its benefits" [6]. So, in order to guarantee that parallelism offers better results, it must be ensured that computational overload costs is taken by a bigger number of processors and that it reduces the evolution time of the P system with respect to be taken by just one processor. Therefore, it must be fulfilled:

$$\frac{C + \Delta_{PSA}(n,P)}{P} < C \tag{9}$$

In particular, the processors oriented parallel software architecture must fulfill:

$$C > \frac{\Delta_{CS}(P) + \Delta_{BS}(P)}{P - 1} \quad (10)$$

While, the evolution rules oriented parallel software architecture must fulfill:

$$C > \frac{\Delta_{CS}(R) + \Delta_{ME}(R) + \Delta_{BS}(R) + \Delta_{PM}(R, P)}{R - 1} \quad (11)$$

7 Conclusions

In this work, we present mechanisms for evaluating parallel software architectures for the evolution of a P system over a given hardware architecture. We propose the parallelism degree and the computational overload for processes management and synchronization as evaluation parameters.

With these parameters and the appropriate set of equations based on them, we have analyzed evolution rules, membranes and processors parallel software architectures. Each one of these architectures proposes different number of processes depending and the P system. But, obtained results in this study shows that the best performance is not parallel software architecture independent, neither always offered by the higher parallelism degree.

In particular, we show that membranes oriented architecture never improves the other two studied parallel software architectures. Moreover, evolution rules oriented architecture requires the design of processes with fined-grained critical sections to offer better results than processors oriented one. Finally, we present here a set of equations establishing the conditions for determining the appropriate parallel software architecture, and the conditions for the parallelism to overcome a totally sequential implementation.

Analysis of these equations permits to conclude that the candidate parallel software architecture is dependent on the number of processors of the hardware architecture and on the P system structure.

References

[1] Gh. Păun, "Computing with Membranes", *Journal of Computer and System Sciences*, 61(2000).

[2] G. Ciobanu, M. Pérez-Jiménez, Gh. Păun, "Applications of Membrane Computing". *Natural Computing Series*, Springer Verlag, (October, 2006).

[3] G. Ciobanu, G. Wenyuan, "A P system running on a cluster of computers", *Proceedings of Membrane Computing. International Workshop*, Tarragona (Spain). Lecture Notes in Computer Science, vol 2933, Springer Verlag (2004), 123-150.

[4] A. Syropoulos, E.G. Mamatas, P.C. Allilomes, K.T. Sotiriades, "A distributed simulation of P systems". *Preproceedings of theWorkshop on Membrane Computing* (A.Alhazov, C.Martin-Vide and Gh.Paun, eds); Tarragona, vol July 17-22 (2003), 455-460.

[5] J.A. Tejedor, G. Bravo, L. Fernández, F. Arroyo, "An Architecture for Attacking the Bottleneck Communication in P System". *AROB 12th. International Symposium on Artificial Life and Robotics*, Oita (january, 2006) (accepted).

[6] D.E. Cormer, D.L. Stevens, "Internet networking with TCP/IP. Vol 3"'. *Prentice Hall, International Editions* (1993).