Master-Slave Distributed Architecture for Membrane Systems Implementation

Ginés Bravo[1], Luís Fernández[2], Fernando Arroyo[2], Jorge Tejedor[3]
Grupo de Computación Natural
[1]Centro de Cálculo
[2]Lenguajes, Proyectos y Sistemas Informáticos
[3]Organización y Estructura de la Información
Escuela Universitaria de Informática
Universidad Politécnica de Madrid
Carretera de Valencia, km 7, 28031, Madrid
Spain

Abstract: - P System computational power lies in its non-deterministic, distributed and massively parallel nature. So, it would be desirable for every implementation of a P System to achieve, as far as possible, these features. In this paper, we suggest a distributed architecture of processors called master-slave, in which communications are directed by a single processor, called 'master', and a series of processors called 'slaves' whose task is to apply evolution rules to the multisets they receive from the master. To prevent collision and network congestion, communications between master and slaves occur in an organized way. Furthermore, some membranes are allocated in each processor and, finally, proxies are used to communicate with membranes allocated in different processors. All this yields better parallelism in the system as a whole than in previously published studies. In addition to this, we present an analytic study that establishes a series of equations that allows us to accurately determine the optimum number of processors needed, the required time to execute an evolution step, the number of membranes to be located in each slave processor and the conditions that will determine when it is best to use this distributed solution or the ones that have previously been proposed, and even the sequential one.

*Keywords: A*rchitecture, Bottleneck, Communication, Master-Slave, P Systems

1 Introduction

The possibilities that natural computation offers and, in particular, the Transition P Systems, for the resolution of problems have led researchers to focus their work on hardware and software implementations of this new model of computation. Transition P systems were presented by Gheorghe Pãun in 1998 [1], who based his work on basic features of biological membranes. A membrane defines a region where a series of chemical elements (multisets) may undergo a series of chemical reactions (evolution rules) and produce other elements. Inside the region limited by a membrane there may be, at the same time, other membranes creating a complex, hierarchical structure that can be represented by a tree. Products generated by the chemical reactions may stay in the same region or travel to container region or to the regions contained by a membrane. As a result of this reaction, a membrane may dissolve itself - its chemical elements transfer to the container membrane - or inhibit itself (the membrane becomes impermeable and does not allow any object to pass).

Membranes systems are dynamic as the chemical reactions inside them produce elements that cross the frontiers of the membranes, travel to other regions and produce new reactions. This dynamic behavior can be sequenced in a series of evolution steps between one and another configuration system that will be determined by the membrane structure and multisets present within membranes. In the transition P systems formal model two phases are distinguished in each evolution step: rules application and communication. In the rules application phase, its rules are applied inside each membrane, in a exhaustive and non deterministic way, to the multisets in parallel. Once the previously described phase has concluded, the communication phase begins and the multisets generated travel towards the target membranes. These systems perform a computation through transition between two consecutive configurations, transforming themselves into computational devices with the same capacities as Turing machines.

The power of this computation model lies in the fact that the process is massively parallel in the rules application phase as well as in the object communication phase. The challenge for researchers is to achieve hardware or software implementations of P systems with a high degree of parallelism.

The aim of this paper is to achieve a distributed implementation of a P System whose step evolution time is as short as possible by increasing parallelism in both application and communication phase.

The paper is structured as follows: first, related works are enumerated and the proposed architectures analyzed; next, a communication architecture model is introduced stating its economical and computational cost as well as its viability; then, a more detailed analysis is offered of the model and, finally, conclusions are drawn.

2 Related Works

A large number of studies propose implementations of a P system in a single processor [2], and these are strictly sequential in nature.

On the contrary, few papers have addressed the possibility of implementation of a distributed cluster of processors. Syropoulos [4] and Ciobanu [3], in their distributed implementations of P systems, use Java Remote Method Invocation (RMI) and the Message Passing Interface (MPI) respectively, on a cluster of PCs connected by Ethernet. These authors do not provide a detailed analysis of the importance of the time used during the communication phase in the total time of P system evolution, although Ciobanu states that "the response time of the program has been acceptable. There are, however, executions that could take a rather long time due to unexpected network congestion" [3]. Specifically, implementation of the second phase of an evolution step, communication between membranes, has not received the same level of attention from the research community.

The paper by Tejedor [5] reviews two models of communication software architectures and proposes an alternative. The first, called "parallel application/parallel communication," consists of an implementation that reflects the massively parallel nature of P systems: each processor has a membrane and it will have as many communication interfaces as children. Nowadays, this is unfeasible because current technology does not allow a processor to have as many communication interfaces as membranes are connected to it.

The second approach, called "parallel application/sequential communication", is more realistic because it is more feasible technologically: all processors are connected to a common bus through a communication interface governed by a protocol. However, this is also unfeasible because massive amounts of time are used in an evolution step, and this time grows lineally with the number of membranes, even causing network congestion.

For his part, Tejedor [5] proposes "distributed architecture with both application and communication phases partially parallel". To achieve this, he relies on the following pillars:

1. In each processor, K membranes are located that will evolve, at worst, sequentially. Where

$$K = {}^{M}\!/_{P}, \quad K \geq 1 \qquad (1)$$

   And M is the total number of membranes of the P System and P the number of processors of the distributed architecture. Physical interconnection between processors is through a common communications line. In this scenario, there are two sorts of communications:
   - Internal communications are those between membranes in the same processor. Communication times are negligible because they occur through use of shared memory techniques.
   - External communications are those between different processors because the membranes that need to communicate are in different processors.

   The benefit obtained is that the number of the external communications decreases.

2. Creation of proxy. Membranes in different processors do not communicate directly. Instead, they communicate through proxies in their respective processors. Proxies are used to communicate between processors. A proxy handles the communications between the membranes in a processor and the proxy of another processor. In the same way, information received from other proxies is redistributed to the membranes in its own processor.

   The benefit of using proxies in communication among membranes instead of direct communication is double. First, the N packets necessary to achieve communication of N membranes with the same parent are transformed into a single packet which length is the corresponding to a single multiset. Second, given that communication protocols penalize the transmission of small packets due to protocol overhead, communicating N messages of L length is slower than one message of (N * L) length.

3. Topology in a processor tree. The benefit of a topology in an interconnection tree among processors lies in how it minimizes the total number of external communications carried out, because proxies exchange information only with its immediate antecessor and direct successors and thus, the total number of external communications is 2(P-1).

4. Token passing in communication to prevent collisions and network congestion. For each processor, a communication order is established. As a result, no more than one proxy can attempt to transmit at any given time.

The analysis of this distributed architecture by [5] is as follows:

- This solution prevents communication collisions and reduces the number and length of the external communications.

- In this model, the minimal time is expressed in the formula:
$$T_{\min} = 2\sqrt{2\, M\, T_{apl}\, T_{com}} - 2\, T_{com} \tag{2}$$
Where, T_{apl} is the maximum time used by the slowest membrane in applying its rules, and T_{com} is the maximum time used by the slowest membrane for communication.

- The number of membranes housed per processor that makes the time minimal is:
$$K_{opt} = \sqrt{\frac{2\, M\, T_{com}}{T_{apl}}} \tag{3}$$

- The number of processors that make the time minimal is:
$$P_{opt} = \sqrt{\frac{M\, T_{apl}}{2\, T_{com}}} \tag{4}$$

- Also, this architecture is highly scaleable, at a moderate cost. The cost is moderate compared to previously proposed architectures because the latter required a total number of processors (*P*) which was equal to the number of membranes (*M*); however, this architecture needs only about $\sqrt{M}$.

- From a throughput perspective, the system is more balanced than previous ones to the extent that an operating percentage of 50% is maintained in processors and communications.

- Also, it offers better step evolution times than one-processor solution when the P System's number of membranes is:
$$M > 8\,\frac{T_{com}}{T_{apl}} \tag{5}$$

3 Master-Slave Distributed Architecture

This paper presents a new distributed architecture which, like previous architecture, maintains the parallelization of the application phase, but also seeks to parallelize the rule application phase in some processors with the communication phase in others. To do this, a series of processors will take on the role of slaves and one processor will act as the master. Moreover, all will be linked by a common communications medium. The functions of each are as follows:

- Each slave (P_S) houses *K* membranes, processes multisets and sends the master multisets whose destination is in a membrane in another slave.
- The master (P_M) redistributes the multisets to the proper slaves. The master contains no membranes.

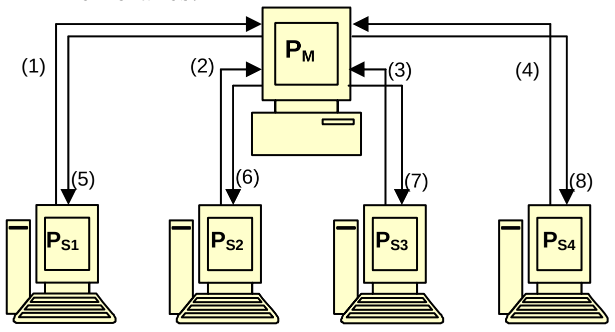

Figure 1. Master-Slave Distributed Model: One Master and Four Slaves

The innovative aspect of this model is that when a slave processor receives multisets from the master, it begins to apply rules autonomously (and parallel) from the other slave processors. Thus, while communication is occurring – master with the slaves – some slave processors are applying rules. That is, the external communication time overlaps with the application time, which is not the case in [5]. The timeline in figure 2 shows when the two phases of an evolution step occur in different processors:

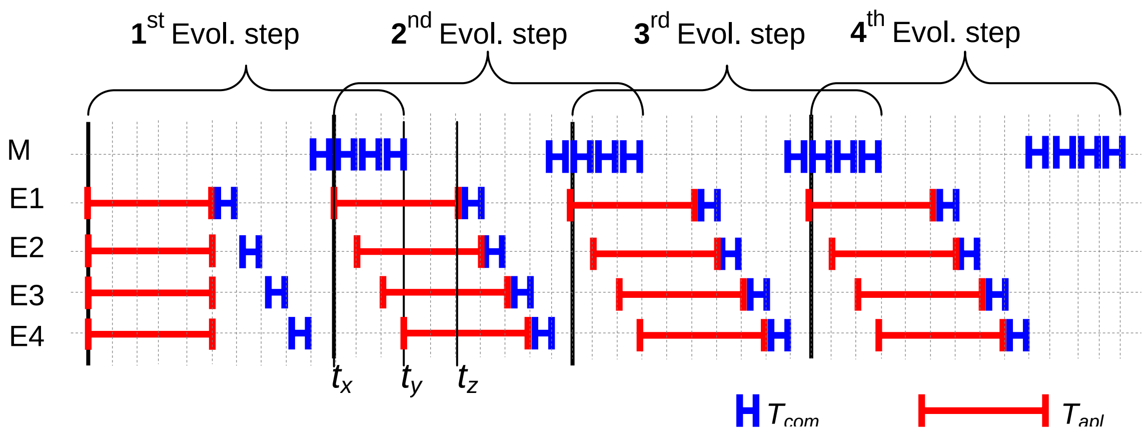

Figure 2. Timeline for Master-Slave Distributed Model with One Master and Four Slaves

As always, the time of an evolution step is from the moment processors apply their rules until results reach the destination membranes. In this model, the first evolution step, slaves already have their multisets and begin to apply their rules immediately. Now, in the result communication phase of this first evolution step, at the t_x moment when the master sends the firs slave the multisets to be processed, this slave begins to apply rules in what is now the second evolution step.

This overlapping of evolution steps is repeated for the remainder. From that moment onward, overlapping occurs because of the parallelism between the communication phase and application of rules; that is, because while the master is sending multisets of an evolution step, there are slave processors applying rules in the next evolution step. This parallelism occurs from the moment the master communicates with the first slave until it sends the multiset to the last slave (interval t_x-t_y).

Finally, there is also parallelism in the rule application phase, for it can be seen that in the time interval delimited by t_y-t_z, all slave processors are applying rules simultaneously.

3.1 Bases of New Model

As in the Tejedor model [5]:

1. There are K membranes in each slave processor in order to increase the number of internal communications - whose communication time is negligible – and thereby reduce the total number of external communications.
2. There is a proxy in each of the processors so that communication between their membranes (master-slave and vice-versa) is through proxies.

The following pillars are specific to the model proposed herein, and which seek to improve the architectures described above:

3. Order in Communications. To eliminate collisions and congestion of the common communication medium, there is an order of communication with the slaves. For this reason, each of the slave processors will receive an order number (or position) so that communications in the two directions (master → slave and slave → master) are orderly. Specifically, the communication policy consists of the following:

   - The master: distributes work (multisets) to different slaves. When it finishes delivering work, it awaits results from slaves
   - The slaves: begin to apply evolution rules to a multiset as soon as they receive them from the master. When they complete application of evolution rules, they have to communicate their results to the master. However, to prevent collisions with other slaves, they wait their turn to do so. Since slaves need not end in order, and because they may have to wait to communicate with the master, communications are in broadcast mode, so the rest will know when it is their turn to broadcast.

4. Flat communications architecture: slave processors do not communicate among themselves, but only with the master; for its part, the master processor communicates only with slave processors. In this scenario, communication occurs at a single level - from many to one and from one to many - that is, there are no hierarchies or connection trees between processors. The physical interconnection between master and slave is through a common communication line.

4 Detailed Analysis of Master-Slave Model

In this model, given that each slave processor has K membranes, the total time of the rule application phase is $K * T_{apl}$. Thus, the time (T) to execute a complete evolution step – i.e., delivery of multisets to slaves, rule application time and return of results - is as follows:

$$T = K\,T_{apl} + T_{com}(P+1) \tag{6}$$

One of the most important parameters which will determine if the architecture is valid is the total number of slave processors (P_{opt}) needed to make the evolution step time minimal. Thus, we set the time (6) according to the number of processors and achieve:

$$P_{opt} = \sqrt{\frac{M\,T_{apl}}{T_{com}}} \tag{7}$$

Hence, by replacing in (6) the optimal number of slave processors (7), the minimal time (T_{min}) needed for an evolution step is:

$$T_{\min} = 2\sqrt{M\,T_{apl}\,T_{com}} + T_{com} \tag{8}$$

For its part, the optimal number of membranes (K_{opt}) with regards to the optimal number of slave processors (P_{opt}) which gives a minimal time (T_{min}) is:

$$K_{opt} = \sqrt{\frac{M\,T_{com}}{T_{apl}}} \tag{9}$$

The expressions that determine the throughput of the processor (Th_{proc}) and of the communication line (Th_{com}) are the following:

$$Th_{proc} = \frac{\sqrt{M\,T_{apl}\,T_{com}}}{2\sqrt{M\,T_{apl}\,T_{com}} + T_{com}} \tag{10}$$

$$Th_{com} = \frac{2\sqrt{M\,T_{apl}T_{com}}}{2\sqrt{M\,T_{apl}\,T_{com}} + T_{com}} \tag{11}$$

If we ignore the value of T_{com}, expressions (10) and (11) are reduced to $Th_{proc}\approx 0.5$ and $Th_{com}\approx 1$, then the system achieved is balanced, in terms of processor performance, and the communication line is fully occupied.

Finally, the cost of the system is approximately double that of the model “architecture with application and communication partially parallel” because double the number of processors is required for the same P System.

4.1 Possible Time Improvements

Given that occupation of the communication line, i.e., throughput, is 100%, improving times of an evolution step would require dealing with the other factor involved, namely the rule application time. In fact, it is feasible for software engineers to make the rules of K membranes in a processor apply more quickly by developing quicker sequential algorithms and making them execute in parallel. If T_{apl} can be made to be N times faster, and if we apply it to the equations K_{opt} (9), P_{opt} (7) and T_{min} (8), we will see that both the number of membranes executed in a processor and the time required to execute an evolution step would improve by approximately a $\sqrt{N}$ factor, while the number of processors required would be divided by the same $\sqrt{N}$ factor

4.2 Advantages of Distributed Model Over One-Processor Model

The time required to execute an evolution step in an architecture of one processor, and therefore no external communications, is:

$$T = M\,T_{apl} \tag{12}$$

As this paper has presented a new distributed master-slave architecture that offers evolution step times determined by (8), it would be interesting to know, for a given P system, at what number of membranes would it be better to use the latter or one based on a single processor. That is, in what conditions is (8) lower than (12):

$$2\sqrt{M\,T_{apl}\,T_{com}} + T_{com} \langle\ M\,T_{apl} \tag{13}$$

In resolving the inequality we get the expression below, which indicates that when the number of membranes of the P system is greater than that value – which is a constant – the solution with a number of processors is superior.

$$M \rangle 5.8 \frac{T_{com}}{T_{apl}} \tag{14}$$

5 Comparative Analysis

As the model proposed by Tejedor [5] offers reasonable times and costs, we shall compare it with the model proposed herein in order to ascertain which can be more efficient:

A. By comparing expressions of the minimal times (9) and (2), and given that T_{com} is negligible compared to the high number of membranes (M), we obtain a proportion of $1/\sqrt{2}$, that is, that this new model reduces the minimal time by 30%. This model achieves better times for the following reasons:

   a) In [5], the communication phase can begin only when all the processors are done applying their rules. However, in the model proposed herein, the application of the evolution step 'n' is parallelized with the communication stage of the previous step 'n-1'. Hence, there is a parallelism between rules application and external communications: while there is processing, there is communication.

   b) By comparing the expressions of the optimal number of processors (7) and (2) we reach the formal conclusion that this model has double the number of processors. This increase in the number of processors means that in the rule application phase there is a higher degree of parallelism.

   c) As a consequence, if there are more processors for the same number of membranes (M), there will be fewer membranes per processor. Indeed, by comparing the expressions (9) and (3), it can be seen that there are half the number of membranes per processor in this new model, which means that the application time in processors will be shorter.

B. To determine the number of membranes above which the times (2) in the Tejedor model [5] are longer than in the model herein (8), we obtain the expression:

$$M \rangle 13.11 \frac{T_{com}}{T_{apl}} \tag{15}$$

By comparing the values in (5), (14) and (15), which indicates in terms of minimal times, above what number of membranes (M) which architecture is preferable, we obtain the following chart:

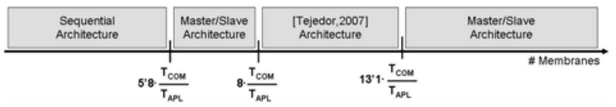
Sequential Architecture
Master/Slave Architecture
[Tejedor,2007] Architecture
Master/Slave Architecture
# Membranes
$5'8 \cdot \frac{T_{COM}}{T_{APL}}$
$8 \cdot \frac{T_{COM}}{T_{APL}}$
$13'1 \cdot \frac{T_{COM}}{T_{APL}}$

Figure 3. Optimal architecture according to number of membranes

C. Finally, but just as important, the simplicity of this method of communication between the master processor and the slaves makes it suitable for implementation with low-cost micro-controllers [6] for two reasons:

   a) Precisely because it is a flat architecture, that is, not hierarchical, it comfortably satisfies the interconnection specifications (synchronization and ICC communication protocol) through a common data bus used in these micro-controllers.

   b) Requirements for processor resources and/or memory are lower.

6 Conclusion

The software architecture proposed in this paper is based on a master processor that distributes work to a series of slave processors that apply rules. Moreover, it places several membranes in each processor, uses proxies for communication among processors, flattens the communication architecture and, finally, establishes an order in communications of nodes.

All these mechanisms offer improvements of previously described architecture in several important respects: minimal times of an evolution step, higher degree of parallelism between rule application phases and external communications, a lack of congestion in the communication medium – even with a high number of membranes -, independence of the topology of the P system, possible implementation of architectures based on micro-controllers and, finally, though of no less importance, the architecture is highly scalable and has moderate costs.

References:

[1] Gh. Pãun, *Computing with membranes*, Journal of Computer and System Sciences, 61, 1 (2000), 108-143

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

[3] G.Ciobanu, W.Guo. *P Systems Running on a Cluster of Computers*, Workshop on Membrane Computing (Gh. Pãun, G. Rozenberg, A. Salomaa Eds.), LNCS 2933, Springer, 123-139, 2004.

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

[5]A. Tejedor, L. Fernandez, F. Arroyo, G. Bravo, *An architecture for attacking the bottleneck communication in P systems*. M. Sugisaka, H. Tanaka (eds.), Proceedings of the 12th Int. Symposium on Artificial Life and Robotics, Jan 25-27, 2007, Beppu, Oita, Japan, 500-505.

[6] A Gutierrez, L Fernández, F Arroyo, V Martínez. *Design of a hardware architecture based on microcontroller for theimplementation of membrane systems*. Proceedings of the 8th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing. Timisoara (Romania), September 2006, 39-42.