Massively Parallel Algorithm for Evolution Rules Application in Transition P System

Luis Fernández; Fernando Arroyo; Jorge A. Tejedor; Juan Castellanos

Grupo de Computación Natural
Universidad Politécnica de Madrid
{setillo; farroyo; jtejedor}@eui.upm.es; jcastellanos@fi.upm.es

Abstract. Within membrane computing research field, there are many papers about software simulations and a few about hardware implementations. In both cases, algorithm for implementing membrane systems in software or hardware that really take advantages for massively parallelism, inherent to these models, are very important. During last years, it can be found some of them in literature in which are implemented some parallel software implementation. The work presented here deal with a massively parallel algorithm for application of evolution rules in a membrane. The algorithm is developed thinking about how minimize the critical sections of evolution rules working with the membrane multiset of objects and how this minimization make possible to really gain in the implementation of real parallelism into programs code.

1 Introduction

P systems are a new computational model based on the membrane structure of living cells [5]. This model has become, during last years, a powerful framework for developing new ideas in theoretical computation. "P systems with simple ingredients (number of membranes, forms and sizes of rules, controls of using the rules) are Turing complete" [6]. Moreover, P systems are a class of distributed, massively parallel and non-deterministic systems. "As there do not exist, up to now, implementations in laboratories (neither in vitro or in vivo nor in any electronical 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]. "An overview of membrane computing software can be found in literature , or tentative for hardware implementations , or even in local networks is enough to understand how difficult is to implement membrane systems on digital devices" [6].

Păun says: "we avoid to plainly say that we have 'implementations' of P systems, because of the inherent non-determinism and the massive parallelism of the basic model, features which cannot be implemented, at least in principle, on the usual electronic computer -but which can be implemented on a dedicated, reconfigurable, hardware [...] or on a local network" [6]. Thereby, there exists many simulators in bibliography 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].

The goal of this work is to present a massively parallel algorithm for the application of evolution rules able to implement, that is not simulate, a transition P system. This algorithm does not determine the utility nor applicability, nor activity of evolution rules, nor the communication of the resulting multiset of objects produced by the application of the evolution rules to their targets. It determines the multiset of applied evolution rules on one evolution step and the resting multiset of objects in a determined membrane.

2 Related Work

In [6], Păun says: "there are several keywords which are genuinely proper to membrane computing and which are of interest for many applications: distribution, algoritmicity, [...], parallelism (a dream of computer science, a common sense in biology)". In this way, Ciobanu presents several related papers about parallel implementation of P systems [1] [2], in which "the rules are implemented as threads. At the initialization phase, one thread is created for each rule. Rule applications are performed in term of rounds" [2]. Again, the author recognize that: "since many rules are executing concurrently and they are sharing resources, a mutual exclusion algorithm is necessary to ensure integrity" [1]. So, "when more than one rule can be applied in the same conditions, the simulator randomly picks one among the candidates" [2]. Hence, processes will have pre-protocols and post-protocols for accessing to critical sections included into their code in order to work under mutual exclusion. Then, each evolution rule set associated to a membrane must access to the shared multiset of objects under mutual exclusion; but different sets of evolution rules associated to different membranes there are no competition among them because they are disjoint processes. Hence, some degree of parallelism is achieved spite of having a thread for each evolution rule. The implementation of evolution rules application will be concurrent inside membranes but not massively parallel.

In conclusion, the obtained gain is that having r evolution rules in the membrane, you have $r - 1$ less iterations in the parallel algorithm than in the sequential one; but, in spite of this, you also have to deal with the management of the mutual exclusion. This situation is detailed in [4] which presents one process architecture analysis, leading to different parallelism degrees. "At this point, there is to note an important aspect in processes synchronization, which is determinant for the degree of parallelism of the process architecture: the critical section granularity" [4]. Thereby, we have coarse-grained solution when almost the whole application process of one evolution rule is under the critical section and fined-grained solution otherwise. In the case of coarse-grained solution, concurrent execution of evolution rules inside a membrane offers a similar behavior to a sequential implementation. Hence, coarse-grained critical sections decrease the degree of parallelism to $M/R \times 100\%$ [4] where M is the number of membranes and R the number of evolution rules. Hence, "in the case in which driven evo-

lution rules architecture will be the appropriate, always parallel algorithms for application of evolution rules with fined-grained critical sections will be required. Otherwise, this architecture will never be the candidate software architecture." [4].

On the other hand, in [3] is presented another sequential algorithm based on maximal applicability benchmark (maximal number of times that the antecedent of the rule $input(r)$ is included into the multiset m) for application of active evolution rules inside a membrane . The complexity for this algorithm, in the worst case, is $log_2\ N$ where N is the weight of the multiset of objects. "The following algorithm is based on considering the maximal applicability benchmark of evolution rules over a multiset of objects. Process is as follows: once an evolution rule has been selected in a non deterministic manner, the rule is applied a random number of times between 1 and the maximal applicability benchmark, per iteration. It is expected that this higher consume of objects will accelerate the end of execution" [3]. Hence, at each algorithm loop several applications of the same evolution rule are performed. This sequential solution is, of course, a minimal parallelism solution.

3 Massively Parallel Algorithm for Evolution Rules Application

Here is presented a solution for massively parallel application of evolution rules. The initial input is a set of active evolution rules for the membrane -the rules are applicable and useful. The desired results are the complete multiset of applied evolution rules and the resulting multiset of objects after rules application. In order to achieve this, we propose one process per rule and one more controller process that simulate the membrane containing the multiset of objects.

The general idea is that each rule proposes, in an independent manner, a multiset to be consumed from the membrane multiset. Whether the addition of all the proposed multiset by rules is bigger than the membrane multiset then rules must to propose a different multiset to be consumed, otherwise, the resulting multiset obtained from addition of the proposed rules multiset is subtracted from the membrane multiset. At this point, rules that are not applicable over the new membrane multiset finishing their process execution till next evolution step. The resulting active rules come back to the starting process point, and again, propose a new multiset to be consumed by. This process is repeated until no rule is applicable over the membrane multiset. This idea can be divided into 8 phases:

Phase 1. *Membrane initialization.* A probability for proposing multiset to be consumed by rules is initialized. This phase is performed only by the controller process, while rules are waiting to phase 2.

Phase 2. *Evolution rules initialization.* Each rule determines its applicability benchmark to its maximal applicability benchmark over the membrane multiset. On the other hand, every rule is settled to the state in which rules can

propose. This phase is performed in parallel by every rule. The controller process -membrane- waits until phase 5.

Phase 3. *Multiset proposition.* Each rule proposes in a randomly manner one multiset of objects to be consumed from the membrane multiset. The proposed rule multiset can be the empty multiset or the scalar product of its antecedent by a natural number chosen in a random manner in between 1 and its applicability benchmark. This phase is performed in parallel for every evolution rule.

Phase 4. *Proposed multiset addition.* The addition of proposed multisets by rules is performed two by two by neighborhood with respect to their number. For example, rule number 1 with rule number 2, rule number 3 with rule number 4, and so on. After finishing this step, the resulting multisets are added two by two again. For example, rule number 1 with rule number 3. And so on until reaching one single multiset. This process develops a binary tree of additions performed in parallel at each level of the tree. This phase is performed in parallel for every rule.

Phase 5. *Collision Management.* Membrane analyzes the two different possibilities in which the proposed multiset cannot consume symbols from the membrane multiset. They are:

A. *By excess.* If the proposed multiset is not included in the membrane multiset. In this case, the proposed multiset is not valid and then evolution rules must go into phase 3 in order to propose new multiset to be consumed.

B. *By defect.* When the proposed multiset is the empty multiset, then the membrane process will be randomly choosen and, after that a natural number in between 1 and the maximal applicability benchmark for the selected rule is generated. In this case, the proposed multiset will be the scalar product of the antecedent of the rules by the chosen natural number.

This phase is performed only by the membrane process while rules wait until the phase 7 or come back to phase 3.

Phase 6. *Symbols consume.* Membrane subtract form its own multiset the proposed multiset from phase 4 or from phase 5-B. Moreover, it indicates to evolution rules that the proposed multiset is a valid multiset. Finally, it initializes its active evolution rules data structure for the next loop in the algorithm. This phase is performed only by the membrane process.

Phase 7. *Checking rules halt.* Each one of the evolution rules accumulates the number of proposed application over the membrane multiset. Moreover, it computes its maximal applicability benchmark over the new resting membrane multiset for the next iteration and, if it is bigger than 0, they pass to the state in which rules can propose and indicate it into the active evolution rules data structure. Otherwise, they finish their execution. This phase is performed in parallel by every evolution rule except into the access to the active evolution rules data structure. Membrane is waiting until phase 8.

Phase 8. *Checking membrane halt.* Membrane checks if there exists some active rule for the next loop. If so, it come back to phase 5, otherwise it finishes the

```
PROCESS TYPE MEMBRANE
(1)  Phase 1. Membrane initialization
(2)  REPEAT
(3)    REPEAT
(4)      Phase 5. Collision management
(5)        - By excess.
(6)        - By defect.
(7)    UNTIL NOT Collision;
(8)  Phase 6. Symbols consume
(9)  Phase 8. Checking membrane halt
(10) UNTIL End

PROCESS TYPE RULES
(1)  Phase 2. Rules initialization
(2)  REPEAT
(3)    REPEAT
(4)      Phase 3. Multisets proposition
(5)      Phase 4. Proposed multisets addition
(6)    UNTIL NOT Collision
(7)    Phase 7. Checking rules halt
(8)  UNTIL End
```

Table 1. Processes pseudo code

execution. This phase is performed only by the membrane and rules wait for coming back to phase 2 -if they are active for next loop- of for finishing their execution.

3.1 Synchronization design

Table 3.1 show in pseudo code the two different types of processes presented here.

Both processes types are not disjoint and they must preserve the following synchronizations:

A. Every evolution rule must wait for initialization until membrane initialization finishes.
B. Each evolution rule must wait for their neighbor evolution rules to finish their respective additions of proposed multisets by their neighbor evolution rules.
C. Membrane must wait to start management collision until evolution rules finish to accumulate the proposed multisets.
D. Every evolution rule must wait to start proposing new multiset until membrane finishes collision management
E. Every evolution rule must wait to start checking halting condition until membrane finishes multisets subtraction.

F. Every evolution rule must wait for the mutual exclusion to access into the active evolution rule data structure and it can perform its register for the next loop iteration.
G. Membrane must wait to checking halting condition until evolution rules finish their corresponding checking for halting conditions.
H. Every evolution rule must wait to start to determine, it they finish their execution or come back to propose a new multiset, until membrane halt checking finishes.

3.2 Efficiency

The algorithm proposed here is measured in terms of the number of performed operation over multisets. The complexity order for this algorithm, in the worst case, is $log_2 N \times log_2 R$, where N is the weight of the multiset and R is the number of rules in the membrane. It is important to note that the theoretical complexity order is worst than the theoretical complexity order for the sequential one. However, in practice, at each iteration of the loop, the new algorithm consumes a higher number of symbols from the membrane multiset than the sequential one. Moreover, the proposed algorithm consumes between 1 and R more symbols than the sequential algorithm. Experimental data holds up this hypothesis. Below the comparative results between two algorithms for evolution rules application: the parallel one with course-grained solution presented in [1] versus the massively parallel with fined-grained solution presented in this work. In order to obtain the presented results a test set has been randomly generated (1000 different evolution rules sets, applied to 1.000.000 multisets). Comparative results presented here are the relation between the number of multiset operation performed by both compared algorithms. First obtained result, attending to the average of all possible situations, is that best behavior is shown by the massively parallel algorithm reducing to 76.38% with respect to the other algorithm. Figure 1 shows the obtained results from the average of the relation between both algorithms related to the number of evolution rules without taken into account the two less relevant parameters. This figure exhibits the penalty for the proposed algorithm when the numbers of evolution rules are 1 and 2. However, in any other case the benefits of the proposed algorithm are increasing. In the set of tests bounded to 10 evolution rules, a reduction of 47% in the number of multisets operation is achieved in the massively parallel algorithm versus the parallel one.

4 Conclusions

This paper presents two different versions of a massively parallel algorithm for evolution rules application in transition P systems. Both algorithms can apply simultaneously several rules several times in the same membrane. Moreover, if every needed condition is accomplished all the rules can be simultaneously applied in the whole P system. Both versions obtain empirical results better than other algorithms than others present in literature, in particular those which do

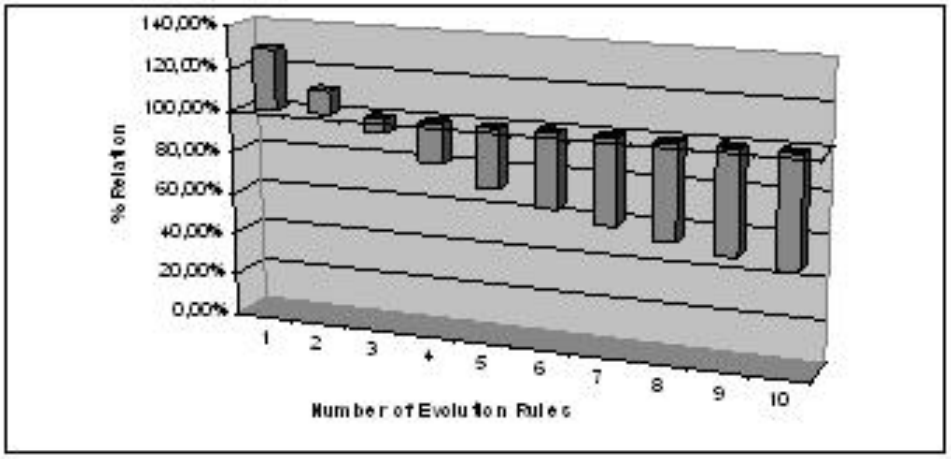

Fig. 1. Average relation between algorithms related to the number of evolution rules

not offer the massively parallel character. This kind of algorithms give a chance to real implementation of P systems in front of simulations and partial approximations presented in others works. Hence, in devoted hardware architectures to membrane systems, in which it is possible to have as many processors as evolution rules plus one more for the membrane, it can be possible to have a real 100% parallelism degree for implementing membrane systems.

References

1. 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.
2. G. Ciobanu, M. Pérez-Jiménez, Gh. Păun, "Applications of Membrane Computing". *Natural Computing Series*, Springer Verlag, (October, 2006).
3. L. Fernández, F. Arroyo, J. Castellanos, J.A. Tejedor, I. García, "New Algorithms for Application of Evolution Rules based on Applicability Benchmarks", *BIOCOMP06 International Conference on Bioinformatics and Computational Biology*, Las Vegas, (June, 2006) (accepted).
4. L. Fernández, F. Arroyo, I. García, J. Tejedor, "Parallel Software Architectures Analysis for Implementing Transition P System". *8th Interantional Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, Timisoara (September, 2006) (submitted).
5. Gh. Păun, "Computing with Membranes", *Journal of Computer and System Sciences*, 61(2000), and Turku Center of Computer Science-TUCS Report n 208, (1998).
6. Gh. Păun, "Membrane computing. Basic ideas, results, applications", *Pre-Proceedings of First International Workshop on Theory and Application of P Systems*, Timisoara, (September , 2005), 1-8.