DELIMITED MASSIVELY PARALLEL ALGORITHM BASED ON RULES ELIMINATION FOR APPLICATION OF ACTIVE RULES IN TRANSITION P SYSTEMS

F. Javier Gil, Luis Fernández, Fernando Arroyo, Jorge Tejedor

Abstract: In the field of Transition P systems implementation, it has been determined that it is very important to determine in advance how long takes evolution rules application in membranes. Moreover, to have time estimations of rules application in membranes makes possible to take important decisions related to hardware / software architectures design.

The work presented here introduces an algorithm for applying active evolution rules in Transition P systems, which is based on active rules elimination. The algorithm complies the requisites of being nondeterministic, massively parallel, and what is more important, it is time delimited because it is only dependant on the number of membrane evolution rules.

Keywords: Natural computing, Membrane computing, Transition P systems, rules application algorithms

ACM Classification Keywords: D.1.m Miscellaneous – Natural Computing

Introduction

Transition P systems are a distributed parallel computational model introduced by Gheorghe Păun based on basic features of biological membranes and the observation of biochemical processes [Păun, 1998]. In this model, membrane contains objects multisets, which evolve according to given evolution rules. Applying the later ones in a nondeterministic maximally parallel way the system changes from a configuration to another one making a computation. This model has become, during last years, an influential framework for developing new ideas and investigations in theoretical computation. “P systems with simple ingredients (number of membranes, forms and sizes of rules, controls of using the rules) are Turing complete” [Păun, 2005]. 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 electronically medium), it seems natural to look for software tools that can be used as assistants that are able to simulate computations of P systems” [Ciobanu, 2006]. “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” [Păun, 2005].

In addition, Gheorghe Păun says that: “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”. Thereby, there exists many P systems 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” [Ciobanu, 2006].

This work presents a time delimited massively parallel algorithm based on rules elimination for application of active rules in transition P systems. After this introduction, other related works appear, where the problem that is

tried to solve is exposed. Later the massively parallel algorithm of application of rules appears developed, including the synchronization between processes and the analysis of its efficiency.

Related Work

J. Tejedor proposes a software architecture for attacking the bottleneck communication in P systems denominated "partially parallel evolution with partially parallel communications model" where several membranes are located in each processor, proxies are used to communicate with membranes located in different processors and a policy of access control to the network communications is mandatory [Tejedor, 2006]. This obtains a certain parallelism yet in the system and an acceptable operation in the communications. In addition, it establishes a set of equations that they allow to determine in the architecture the optimum number of processors needed, the required time to execute an evolution step, the number of membranes to be located in each processor and the conditions to determine when it is best to use the distributed solution or the sequential one. Additionally it concludes that if the maximum application time used by the slowest membrane in applying its rules improves *N* times, the number of membranes that would be executed in a processor would be multiplied by $\sqrt{N}$, the number of required processors would be divided by the same factor, and the time required to perform an evolution step would improve approximately with the same $\sqrt{N}$ factor.

Therefore, to design software architectures it is precise to know the necessary time to execute an evolution step. For that reason, algorithms for evolution rules application that they can be executed in a delimited time are required, independently of the object multiset cardinality inside the membranes. Nevertheless, this information cannot be obtained with the present algorithms since its execution time depends on the cardinality of the objects multiset on which the evolution rules are applied.

In addition, Ciobanu presents several related papers about parallel implementation of P systems [Ciobanu 2002, 2004, 2006], 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" [Ciobanu, 2006]. Again, the author recognizes that: "since many rules are executing concurrently and they are sharing resources, a mutual exclusion algorithm is necessary to ensure integrity" [Ciobanu, 2004]. So, "when more than one rule can be applied in the same conditions, the simulator randomly picks one among the candidates" [Ciobanu, 2006]. 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.

On the other hand, L. Fernández proposes a massively parallel algorithm for evolution rules application [Fernández, 2006]. In this solution a process by each rule is generated and exist one more controller process that simulates the membrane containing the objects multiset. In a loop, each rule process proposes simultaneously a object multiset to be consumed and the membrane process determines if it is possible to apply the proposed multiset, until the proposal is correct. The algorithm execution finishes when there is no active rule. This last solution contains a high degree of parallelism, but its execution time is not delimited. Therefore, this algorithm is not appropriate to be used in the previously commented [Tejedor, 2006] software architecture.

Finally, in [Tejedor, 2007] is exposed an algorithm for application of evolution rules based on active rules elimination. In this algorithm, in each loop iteration all the rules -except the last one- are sequentially applied a random number of times. Next, the last active rule is applied the greater possible number of times, reason why it

became inactive. This algorithm reaches a certain degree of parallelism, since one rule can be simultaneously several times applied in a single step. In this algorithm, the execution time depends on the number of rules, not of the objects multiset cardinality. In the experimental tests, this algorithm has obtained better execution times than the previously published sequential algorithms. This sequential solution is, of course, a minimal parallelism solution.

Delimited Massively Parallel Algorithm based on Rules Elimination for Application of Active Rules in Transition P Systems

Here we present a time delimited massively parallel algorithm for application of active rules. The initial input is a set of active evolution rules for the corresponding membrane -the rules are applicable and useful- and the initial membrane multiset of objects. The final results are the complete multiset of applied evolution rules and the obtained multiset of objects after rules application. In order to achieve this, we propose one process for each rule and one more controller process that simulate the membrane containing the multiset of objects.

The general idea is that each rule -except the last one- randomly proposes, in an independent manner, a multiset to be consumed from the membrane multiset (the obtained algorithm is nondeterministic due to this random proposal). If the addition of all the proposed multiset by rules is smaller than the membrane multiset, then the proposed multiset is subtracted from the membrane multiset. Next, the last active rule determines and applies its maximal applicability benchmark over the membrane multiset, subtracting the correspondent multiset from the membrane multiset. At this point, rules that are not applicable over the new membrane multiset finish their process execution –obviously, including the last active rule-. The resting active rules come back to the starting point, and again, propose a new multiset to be consumed. This process is repeated until none rule is applicable over the membrane multiset.

This idea can be divided into seven phases:

Phase 1 *Membrane initialization*. A global probability for proposing multiset to be consumed by rules is initialized. One rule is able to consume with a determined probability; but if a rule is not allowed to consume then it will not propose multiset to be consumed until the next loop iteration.

This phase is performed only by the controller process, while rules are waiting to second phase.

Phase 2 *Evolution rules initialization*. Each rule -except the last active 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 propositions*. Considering the global probability established by the membrane, 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, except the last one.

Phase 4 *Sum of Multiset Proposals*. The addition of the 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 way for adding multiset 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 *Proposal management and last active rule maximal application*. Membrane analyzes the proposed multiset by the rules. If the proposed multiset is valid (not empty and included in the membrane multiset), then the membrane subtract from its own multiset the proposed multiset from phase 4. Next, the membrane process determines and applies the maximal applicability benchmark of the last active rule over the membrane multiset, subtracting the correspondent multiset from the membrane multiset. Moreover, the membrane process indicates to the rule processes the executed operation. Finally, it initializes the information about its active evolution rules for the next loop in the algorithm.

This phase is performed only by the membrane process while rules wait until the phase 6.

Phase 6 *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 7.

Phase 7 *Checking membrane halt*. Membrane checks if there exists some active rule for the next loop and, in this case, it returns to establish the global probability to propose multisets by the rules. If so, it come back to phase 5 waiting the proposal management, otherwise it finishes the execution.

This phase is performed only by the membrane and the rules wait for coming back to phase 3 -if they are active for next loop- of for finish their execution.

Next we will deal with two different aspects for the exposed general idea: the phases synchronization and finally efficiency analysis.

Synchronization Design

Accordingly whit the previous explanation, these phases shared between the two different processes types, evolution rules and membrane, as it can be observed in tables 1 and 2.

```
(1)  Phase 1: Membrane initialization
(2)  REPEAT
(3)    Phase 5: Proposal Management &
                Last active rule maximal application
(4)    Phase 7: Checking membrane halt
(5)  UNTIL End
```

Table 1: Process Type Membrane

```
(1)  Phase 2: Rules initialization
(2)  REPEAT
(3)    Phase 3: Multiset proposition
(4)    Phase 4: Sum of Multiset Proposals
(5)    Phase 6: Checking rules halt
(6)  UNTIL End
```

Table 2: Process Type Evolution Rule

Both processes types are not disjoint and they must preserve the following synchronizations (Fig. 1 presents the activity diagram showing the needed synchronization in the different phases for the process membrane and two evolution rules processes):

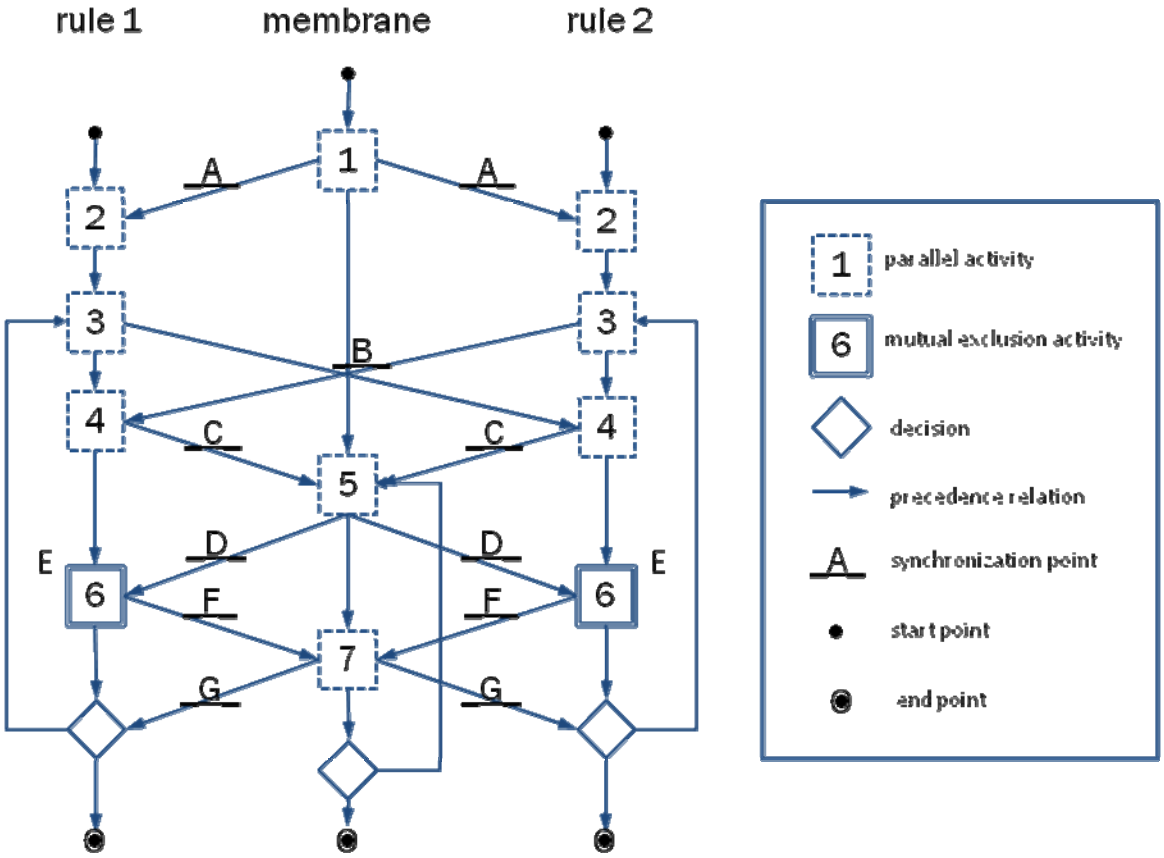

Figure 1: The activity diagram showing the needed synchronization in the different phases for the membrane and two evolution rules

A. Every evolution rule must wait for initialization until membrane initialization finishes.

B. Each evolution rule must wait for their neighbor evolution rules finish their respective additions of proposed multisets by their neighbor evolution rules.

C. Membrane must wait to start management collision until evolution rules finish accumulating the proposed multisets.

D. Every evolution rule must wait to start checking halting condition until membrane finishes multisets subtraction.

E. 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.

F. Membrane must wait to checking halting condition until evolution rules finish their corresponding checking for halting conditions.

G. 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.

Efficiency Analysis

Analyzing the membrane process it can be observed that the proposed algorithm executes, at the most, so many times as rules exist initially (we will denominate it by R), since in each iteration at least one rule is eliminated in the worse case. The rest of operations executed by the process membrane can be considered like basic operations. Moreover, the operations executed by the rules processes are simple, except the sum of proposals made in phase 4. This sum is performed two by two by neighborhood, reason why the obtained complexity order is $log_2 R_i$, being R_i the number of active rules minus one in each loop iteration. Consequently, the complexity order of the proposed algorithm is:

$$\sum_{i=R-1}^{2} log_2 i = log_2(R - 1) + log_2(R - 2) + \ldots + log_2 2 = log_2(R - 1)!$$

Consequently, we can conclude that the complexity order of the proposed algorithm -in the worse case- is $log_2(R - 1)$!, but better results can be expected experimentally than the ones obtained theoretically, because exists the possibility that in a same loop iteration disappears more than a rule.

Future Work

In first phase of the presented algorithm -membrane initialization- a global probability for the rule processes is determined. This value determines the probability of proposing an objects multiset by a rule. Assigning a value of $1/R$ to this probability very good results in the made tests have been obtained. At the moment we are working in the process of determination of this value, trying of obtaining a better efficiency.

In addition, since one has studied in this work, the presented algorithm eliminates an active rule in each loop iteration. Evidently, the order in that the rules are applied influences in the final results obtained. Therefore, to improve the algorithm efficiency it seems interesting to study as it would have to be the last applied rule, studying the relations between the antecedents of the rules available.

Conclusions

This paper introduces an algorithm of active rules application based on rules elimination in transition P systems. The two most important characteristics of this algorithm are:

- The presented algorithm is massively parallel
- The execution time of the algorithm is time delimited, because it only depends on the number of rules of the membrane. The number of rules of the membrane is a well-known static information studying the P system

We think that the presented algorithm can represent an important contribution in particular for the problem of the application of rules in membranes, because it presents high productivity and it allows estimate the necessary time to execute an evolution step. Additionally, this last one allows to make important decisions related to the implementation of P systems, like the related ones to the software architecture.

Bibliography

[Ciobanu, 2002] G. Ciobanu, D. Paraschiv, "Membrane Software. A P System Simulator". Pre-Proceedings of Workshop on Membrane Computing, Curtea de Arges, Romania, August 2001, Technical Report 17/01 of Research Group on Mathematical Linguistics, Rovira i Virgili University, Tarragona, Spain, 2001, 45-50 and Fundamenta Informaticae, vol 49, 1-3, 61-66, 2002.

[Ciobanu, 2004] 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, 123-150, Springer Verlag, 2004.

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

[Fernández 2006] L. Fernández, F. Arroyo, J. Tejedor, J. Castellanos. "Massively Parallel Algorithm for Evolution Rules Application in Transition P System". Seventh Workshop on Membrane Computing, WMC7, Leiden (The Netherlands). July, 2006

[Păun, 1998] G. Păun. "Computing with Membranes". In: Journal of Computer and System Sciences, 61(2000), and Turku Center of Computer Science-TUCS Report nº 208, 1998.

[Păun, 2005] G. Păun. "Membrane computing. Basic ideas, results, applications". In: Pre-Proceedings of First International Workshop on Theory and Application of P Systems, Timisoara (Romania), pp. 1-8, September, 2005.

[Tejedor, 2006] J. Tejedor, L. Fernández, F. Arroyo, G. Bravo. "An Architecture for Attacking the Bottleneck Communications in P systems". In: Artificial Life and Robotics (AROB 07). Beppu (Japan), January 2007.

[Tejedor, 2007] J. Tejedor, L. Fernández, F. Arroyo, A. Gutiérrez. "Algorithm of Active Rules Elimination for Evolution Rules Application" (submited). In 8th WSEAS Int. Conf. on Automation and Information, Vancouver (Canada), June 2007.

Authors' Information

F. Javier Gil Rubio – Dpto. de Organización y Estructura de la Información, E.U. de Informática. Natural Computing Group, Universidad Politécnica de Madrid, Spain; e-mail: jgil@eui.upm.es

Luis Fernández Muñoz - Dpto. de Lenguajes, Proyectos y Sistemas Informáticos, E.U. de Informática. Natural Computing Group, Universidad Politécnica de Madrid, Spain; e-mail: setillo@eui.upm.es

Fernando Arroyo Montoro - Dpto. de Lenguajes, Proyectos y Sistemas Informáticos, E.U. de Informática. Natural Computing Group, Universidad Politécnica de Madrid, Spain; e-mail: farroyo@eui.upm.es

Jorge A. Tejedor Cerbel - Dpto. de Organización y Estructura de la Información, E.U. de Informática. Natural Computing Group, Universidad Politécnica de Madrid, Spain; e-mail: jtejedor@eui.upm.es