Algorithm of Rules Application Based on Competitiveness of Evolution Rules

Jorge Aurelio Tejedor, Luis Fernández, Fernando Arroyo, Sandra Gómez

Natural Computing Group
Universidad Politécnica de Madrid
{jtejedor; setillo; farroyo; sgomez}@eui.upm.es

Summary. Transition P systems are a parallel computational model based on the notion of the cellular membrane system. The distributed architectures for P system implementation in digital device must deal jointly with the time for active rules application over multisets and the time for communication among membranes. Analysis of these architectures shows that it is very important to improve the time used in the rules application step since this allows for reducing the evolution step time and the number of processors needed in the system. This work introduces the concepts of competitiveness relationship among active rules and competitiveness graph. For this, it takes into account the fact that some active rules in a membrane can consume disjointed object sets. Based on these concepts this paper presents a new evolution rules application algorithm that improves throughput of active rules elimination algorithms (sequential and parallel).

1 Introduction

Computation with membranes was introduced by Gheorghe Păun in 1998 [10] through a definition of transition P systems. This new computational paradigm is based on the observation of biochemical processes. The region defined by a membrane contains chemical elements (multisets) which are subject to chemical reactions (evolution rules) to produce other elements. Transition P systems are hierarchical, as the region defined by a membrane may contain other membranes. Multisets generated by evolution rules can be moved towards adjacent membranes (parent and children). This multiset transfer feeds back into the system so that new products are consumed by further chemical reactions in the membranes.

These systems perform computations through transition between two consecutive configurations. Each transition or evolution step goes through two sequential steps: rules application and objects communication. First, the evolution rules are applied simultaneously to the multiset in each membrane. This process is performed by all membranes at the same time. Then, also simultaneously, all membranes communicate with their destinations.

The objective of this paper is to present an improvement of the algorithm of active rules elimination [12] used in the rules application step. To achieve this, the paper is structured as follows: first, related works are presented; then, the basic ideas of the active rules elimination algorithm are summarized, which is followed by a definition of the concept of competition between rules; next, the optimization of the algorithm is specified and, finally, conclusions are drawn.

2 Related works

No line of research for the implementation of P system in digital devices has managed to achieve the massively parallel nature of these systems. However, significant advances have been made. First, specifications have been made of faster algorithms for application of evolutions rules for both sequential devices [12] [3] and parallel ones [4] [5]. Moreover, the problem of network congestion in the multiset communications step described by Ciobanu [2] has been solved in the works of Tejedor [11] and Bravo [1]. In short, today a number of implementations can be made that achieve a certain degree of parallelism that depends on communications and the application of evolution rules.

Viable communications architectures for the P systems implementation proposed in [11] and [1] use a number of processors on the order of the square root of the number of membranes and place several membranes in each processor. Moreover, these solutions achieve an evolution pass time, at worst, on the order of the square root of the number of membranes and eliminate network congestion by means of access control to the common medium. However, these architectures need to know the time needed to perform rules application. This information is needed to be able to optimally distribute membranes among processors. In these P system implementation architectures, it is important to improve the time used in the rules application step since this allows for reducing the evolution step time and the number of processors needed in the system. Specifically, if rule application time is divided by the factor N, then the evolution time and the number of processors in the system is divided by the square root of N.

Analysis of the rules application algorithms published to date shows that only the algorithm of active rules elimination [12] together with its parallel version [5] meet the viable architectures of Tejedor [11] and Bravo [1]. These 2 algorithms enable prior determination of the maximum execution time, since this value depends on the number of rules rather than on the multiset cardinal to which they are applied, as in other algorithms reported. [2] [3]. In addition, these algorithms are the quickest in their category (sequential and parallel). This paper improves throughput of active rules elimination algorithms and takes into account the fact that some active rules in a membrane can consume disjointed object sets. This improvement is applicable to the algorithm in both its sequential and parallel versions.

3 Algorithm of active rules elimination [12]

The general idea of this algorithm is to eliminate, one by one, the rules from the set of active rules. Each step of rule elimination performs two consecutive actions:

1. Iteratively, any rule other than that which is to be eliminated is applied for a randomly selected number of times in an interval from 0 to the maximum applicability benchmark. This action ensures the non-determinism inherent to P systems.
2. The rule to be eliminated is applied a number of times which is equal to its maximum applicability benchmark, thus making applicable no longer and resulting in its disappearance from the set of active rules.

We assume that:

1. The object multiset to which active rules are applied is ω.
2. The active rules set is transformed in an indexed sequence R in which the order of rules is not relevant.
3. The object multiset resulting from application of active rules is ω'.
4. The multiset of applied rules that constitute the algorithm output is ω_R.
5. Operation $|R|$ determines the number of rules in the indexed sequence R.
6. Operation $\Delta_{R[Ind]}\lceil\omega'\rceil$ calculates the maximum applicability benckmark of the rule $R[Ind]$ over ω'.
7. The operation $input(R[Ind])\cdot K$ performs the scalar product of the antecedent of rules by a natural number.

and thus the algorithm is:

(1) $\omega' \leftarrow \omega$
(2) $\omega_R \leftarrow \emptyset_{M_{R(O,T)}}$
(3) **FOR** $Last = |R|$ **DOWNTO** 1
(4) **BEGIN**
(5) **FOR** $Ind = 1$ **TO** $Last - 1$ **DO**
(6) **BEGIN**
(7) $Max \leftarrow \Delta_{R[Ind]}\lceil\omega'\rceil$
(8) $K \leftarrow random(0, Max)$
(9) $\omega_R \leftarrow \omega_R + \left\{R[Ind]^K\right\}$
(10) $\omega' \leftarrow \omega' - input(R[Ind]) \cdot K$
(11) **END**
(12) $Max \leftarrow \Delta_{R[Last]}\lceil\omega'\rceil$
(13) $\omega_R \leftarrow \omega_R + \left\{R[Last]^{Max}\right\}$
(14) $\omega' \leftarrow \omega' - input(R[Last]) \cdot Max$
(15) **END**

Remember that if rule $R[i]$ is no longer applicable in the elimination step for $R[j]$, it is no longer necessary to perform the elimination step for $R[i]$, the algorithm is greatly improved, as shown in [12].

In each iteration of the algorithm of actives rules elimination, the maximum applicability benchmark of a rule is calculated and then the rule is applied. The number of iterations executed at worst is:

$$\#iterations = \sum_{i=1}^{q} i = \frac{q \cdot (q+1)}{2}$$

Let q be the cardinal of the indexed sequence of active rules. Thus, this algorithm allows one to know how long it takes to be executed in the worst case, with knowledge of the rules set of a membrane.

It is important to note that, in general, it is essential to perform the first action in each elimination step of a rule. This action is necessary to ensure that any possible result of the rules application to the multiset is produced by the algorithm. In case the action is not performed, the eliminated rule (applied as many times as the value of its maximum applicability benchmark) may consume the objects necessary so that any other rule can be applied. However, the latter does not always occur and the first action in each elimination step can be simplified. For the sake of illustration, let us assume that the antecedents of a set of active rules are shown in figure 1.

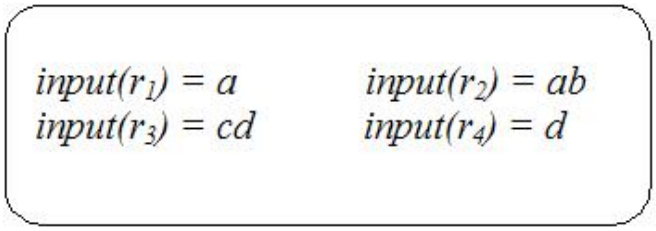

Fig. 1. Antecedents of a set of active rules

In this case, in the elimination step of the rule r_1 only the first action with the rule r_2 has to be taken, as r_1 and r_2 are the only rules with the object a in their antecedents. The same is the case with rules r_3 and r_4, as these two compete for the object d. Thus, taking into account the competition between rule antecedents, one can adjust the rule elimination algorithm to perform only 6 iterations in the worst case, rather than 10 (2 to eliminate r_1, 1 to eliminate r_2, 2 to eliminate r_3, 1 to eliminate r_4) as shown in figure 2.

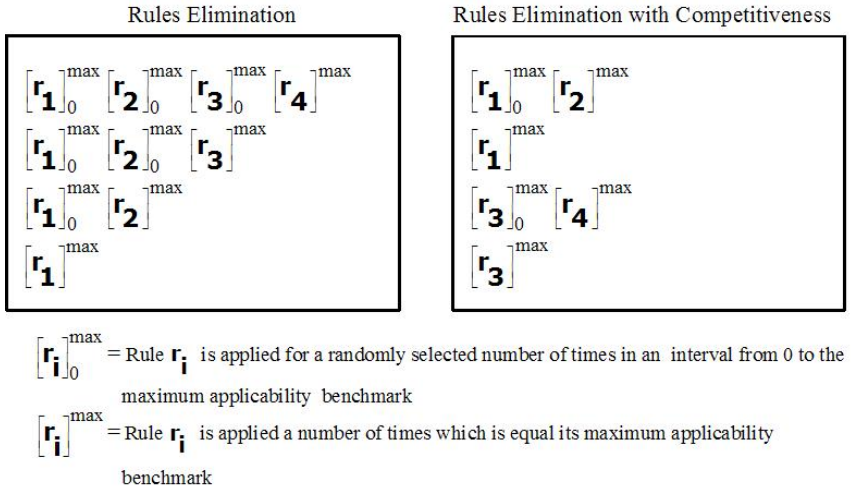

Fig. 2. Execution trace of Rules Elimination and Rules Elimination with competitiveness algorithms

4 Definition of competitiveness between rules

Definition 1: *Competitiveness graph.*
Let R be a set of active rules

$$R = \{r_1, r_2, ..., r_q\} \ with \ q > 0$$

Let C be a binary relationship defined over the set R

$$\forall x, y \in R, \ x \neq y \quad x \, C \, y \ \Leftrightarrow \ input(x) \cap input(y) \neq \emptyset$$

This binary relationship can be represented by a non-directed graph $CG = (R, C)$ called a competitiveness graph, where the rules are related to each other if, and only if its antecedents have an object in common. For example, given the rules inputs shown in figure 3, the competitiveness graph generated by these rules taking into account the relationship C will be as shown in figure 4.

Definition 2: *Subgraph resulting from elimination of a rule.* Let competitiveness graph be $CG = (R, C)$ and rule $x \in R$. The subgraph resulting from elimination of rule x is defined as:

$$CSG = (R - \{x\}, C \cap R - \{x\} \times R - \{x\})$$

Definition 3: *Induced subgraph.* Let a competitiveness graph be $CG = (R, C)$ and $R' \subseteq R$. The competitiveness subgraph induced by the subset R' is the graph:

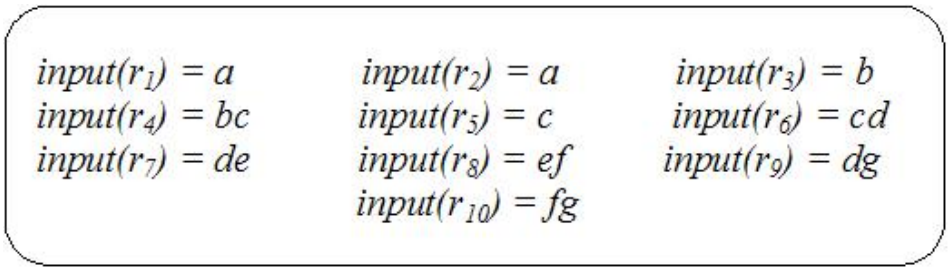

Fig. 3. Antecedents of a set of active rules

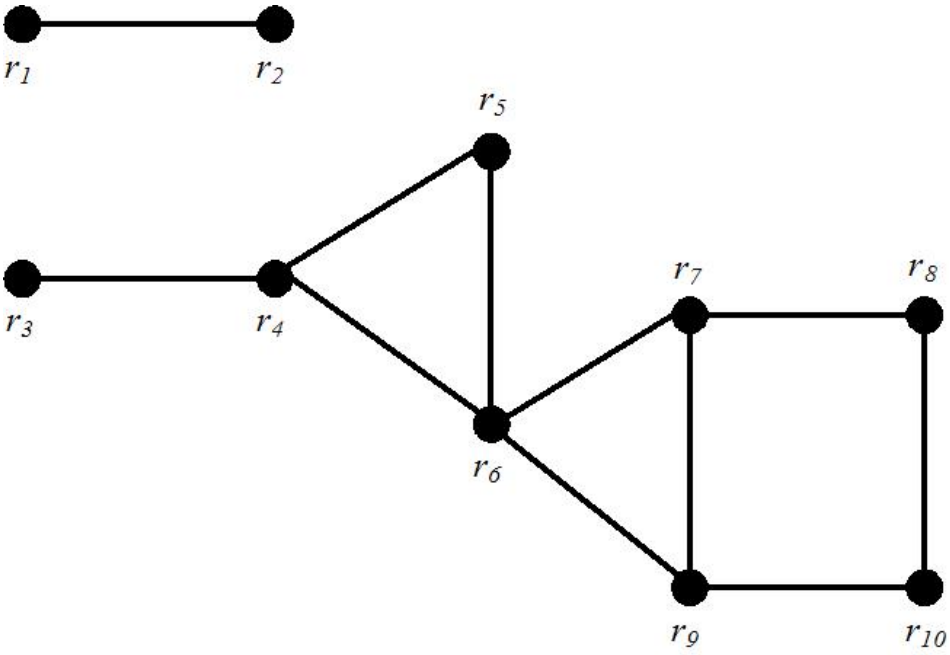

Fig. 4. Competitiveness graph

$$CSG = (R', C \cap R' \times R')$$

Definition 4: *Competitiveness chain.* For a competitiveness graph $CG = (R, C)$, a competitiveness chain is defined as an ordered sequence of rules pertaining to R

$$s_1,\ s_2,\ \ldots\ , s_n \quad s_i \in R,$$

satisfying:

$$s_i\, C\, s_{i+1} \quad \forall i \in \{1,\ \ldots,\ n-1\}$$

By definition, there is always a competitiveness chain composed of a single rule.

Definition 5: *Accessible rule relationship.* For a competitiveness graph $CG = (R, C)$, the accessible rule relationship (A) is defined as:

$$x, y \in R \quad x\, A\, y \;\Leftrightarrow\; \exists \text{ a competitiveness chain } \; s_1, ..., s_n | s_1 = x \wedge s_n = y$$

This is an equivalence relation which divides the rule set R into equivalence classes.

Definition 6: *Connected component of competitiveness graph.* Let competitiveness graph be $CG = (R, C)$, let the accessible rule relationship be A and let E be an equivalence class produced by A. The connected component of CG is defined as the graph induced by the vertices pertaining to the equivalence class E.

Definition 7: *Connected competitiveness graph.* Let a competitiveness graph be $CG = (R, C)$ and consider the its rule accessibility relation. We call connected CG if and only if it has a connected component.

Definition 8: *Articulation.* For a competitiveness graph $CG = (R, C)$ and a rule $x \in R$, it is said that x is an articulation of CG if and only if the subgraph resulting from the elimination of rule x has more connected components than CG.

5 Algorithm based on rules competitiveness

Based on the rules competitiveness relationship of a membrane's rules one can improve the algorithm of elimination of active rules. To do this, an analysis must be made of the evolution rules of each membrane prior to P system evolution. The analysis will determine the order of active rules elimination and what rules set are used in the first action of each elimination step of a given rule. The following optimizations can be made of the algorithm of rule elimination:

5.1 1^{st} optimization

The idea of this optimization is based on the fact that in the elimination step of a rule, the first action of the algorithm must be applied to the rules in the same connected component of the competitiveness graph. This can be done because the antecedents of rules in different connected components do not compete for common objects of the multiset.

The analysis prior to the execution of each P system calculates the competitiveness graph of each membrane. Then the connected components of the graph are calculated. The algorithm of active rule elimination will be applied independently to the rules of each of the connected components, with no need for any change in its codification.

In the worse case of the example of figure 4, the sequential version of this algorithm will need to perform 3 iterations in the connected component consisting of the rules $\{r_1,\ r_2\}$ and 36 iterations in the connected component consisting of

the rules $\{r_3,\ r_4,\ r_5,\ r_6,\ r_7,\ r_8,\ r_9,\ r_{10}\}$. Therefore, this example has gone from 55 iterations in the worst case of the algorithm of active rules elimination to 39 iterations (figure 5), that is, it has been reduced by 71% compared to the active rules elimination algorithm.

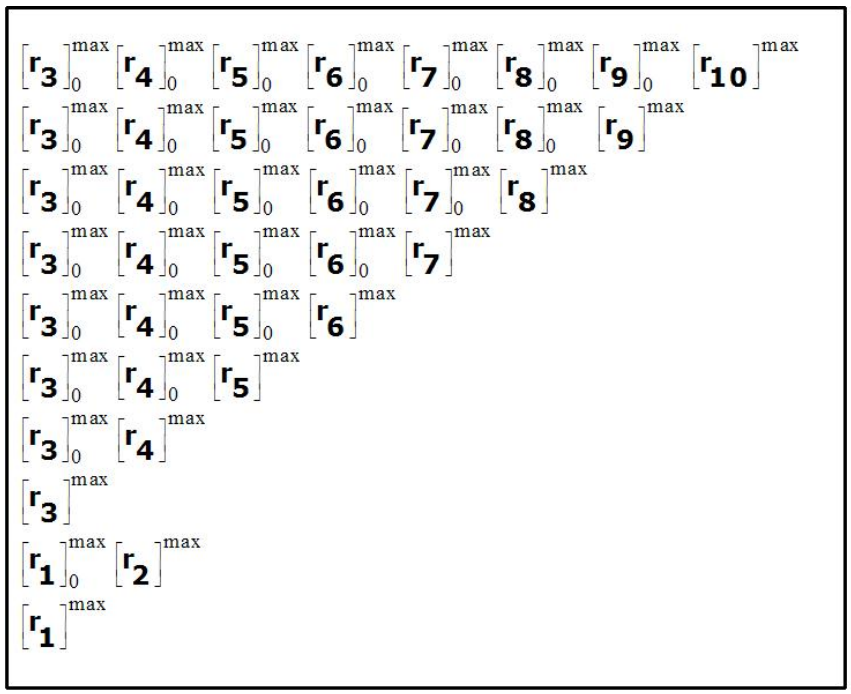

Fig. 5. Execution trace of sequential 1^{st} optimization

Making a parallel version of the algorithm is quite simple. One need only apply the algorithm of active rules elimination in parallel to the rules of each connected component on the competitiveness graph. The parallel version would require only 36 iterations ($maximum(36, 3)$) in the worst case, as shown in figure 6), that is, it has been reduced by 65% compared to the active rules elimination algorithm.

5.2 2^{nd} optimization

This optimization is applied in each connected component of the competitiveness graph. If the competitiveness graph of a membrane has articulations the algorithm can be used to eliminate these rules first and cause the appearance of new connected components. Thus, if rule r_6 is eliminated in our example (figure 4) the connected component splits in two: the one composed of $\{r_3,\ r_4,\ r_5\}$ and the one composed of $\{r_7,\ r_8,\ r_9,\ r_{10}\}$.

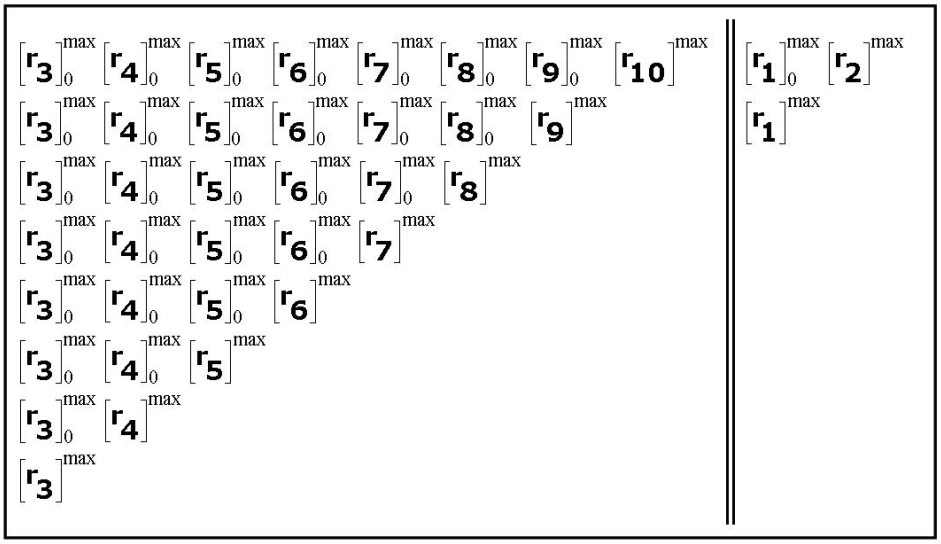

Fig. 6. Execution trace of parallel 1^{st} optimization

When a connected component has no articulations, elimination of more than one rule can break it into more than one connected component. Continuing with the example we have proposed, if we first remove from connected component $\{r_7,\ r_8,\ r_9,\ r_{10}\}$ rules r_7 and r_{10} in two elimination steps, it then splits into two connected components consisting of the rules r_8 and r_9, respectively.

To perform this optimization, a slight change must be made in the sequential algorithm of active rules elimination. Now, each step of elimination of a rule must eliminate a specific rule and is carried out in a sequence of determinate rules. Moreover, there is a certain partial order in the elimination steps of a rule. Whereas order is irrelevant in previous versions of the active rules elimination algorithm, it is decisive in this version. The set of rules used and the rule being eliminated in each elimination step is calculated for each membrane in the analysis prior to the evolution of the P system; as a result, the calculation does not penalize the execution time of the algorithm.

Figure 7 shows the order in which evolution rules are eliminated and the set of rules used in each elimination step for the example in figure 4. The number of iterations of this algorithm in the worst case is 25, that is, it has been reduced by 45% compared to the active rules elimination algorithm.

The parallel version of the algorithm involves applying the sequential version to each of the connected components that are either in the original competitiveness graph or that are generated as a result of the elimination of a rule.

The execution trace of the parallel algorithm used with the set of rules of the example in figure 4 is shown in figure 8. It may be noted that the number of iterations in the worst case is 16 $(maximum(8,2)+maximum(3,4,1)+maximum(1,1,3)+$

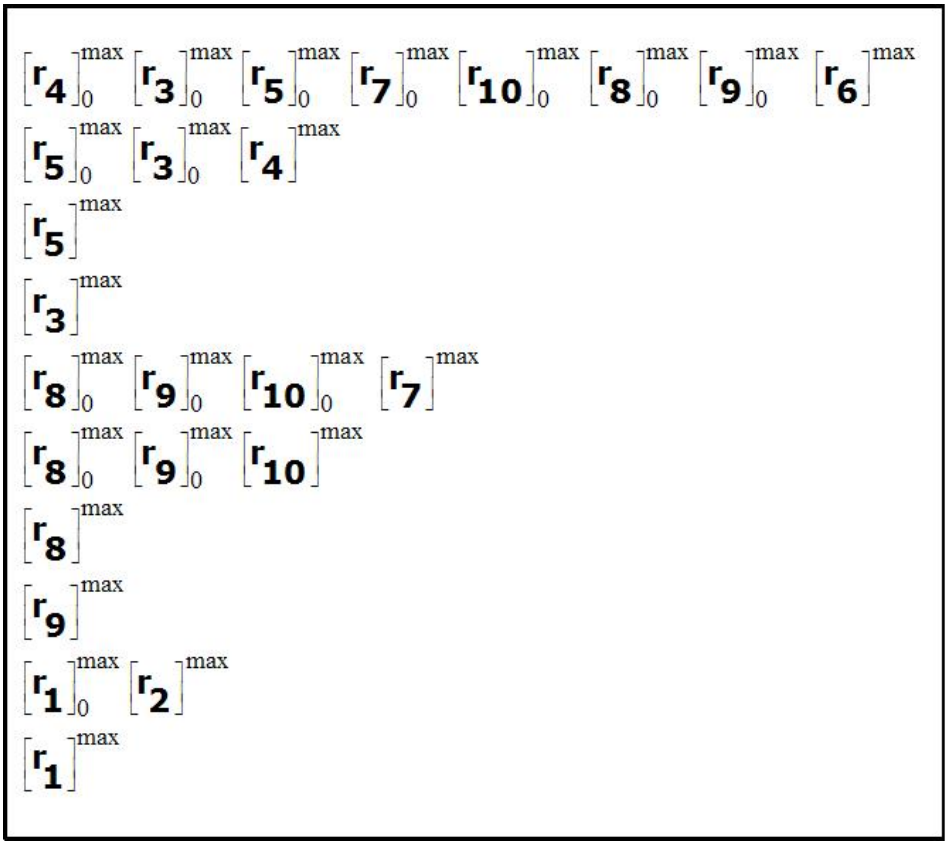

Fig. 7. Execution trace of sequential 2^{nd} optimization

$maximum(1,1)$) using 5 processes. Hence, the number of iterations is reduced by 29% compared to the active rules elimination algorithm.

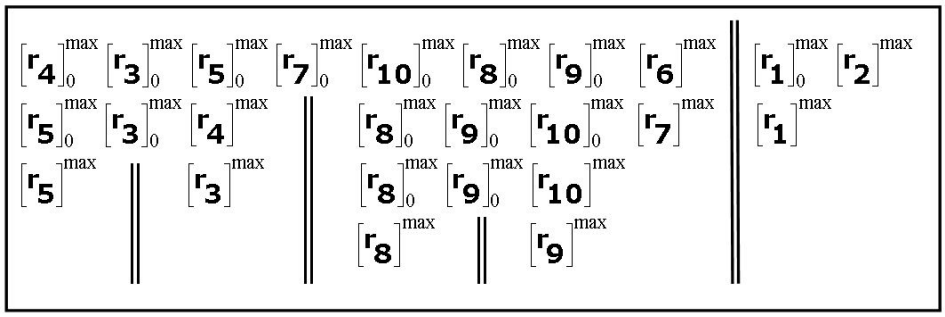

Fig. 8. Execution trace of parallel 2^{nd} optimization

Table 1 shows the number of iterations performed by the algorithms in the worst case:

- Actives rules elimination (ARE)[12]
- Sequential competitive rules with 2^{nd} optimization(SCR)
- Delimited massively parallel (DMP)[5]
- Parallel competitive rules with 2^{nd} optimization (PCR)

Applied to P systems defined in:

- Reference#1:*Computing with Membranes* [8]
- Reference#2:*A Guide to Membrane Computing* [9]
- Reference#3:*P Systems Running on a Cluster of Computers* [2]

	Sequential			**Parallel**		
	ARE	SCR	SCR/ARE	DMP	PCR	PCR/DMP
Ref#1 pag 10	6	4	66%	5	3	60%
Ref#1 pag 13	6	4	66%	5	3	60%
Ref#1 pag 15	3	3	100%	3	3	100%
Ref#2 pag 9	6	4	66%	5	3	60%
Ref#2 pag 14	10	6	60%	8	3	37%
Ref#3 pag 137	10	5	50%	8	3	37%
Ref#3 pag 138	6	4	66%	5	3	60%

Table 1. Number of iterations performed by the algorithms in the worst case

5.3 3^{rd} optimization

This last optimization is based on an analysis of the execution trace of the 2^{nd} optimization. It can occasionally be observed that the elimination step of one rule r_j also eliminates one or more additional rules r_i. This can occur either because r_i is applied a number of times that coincides with the maximum applicability benchmark, or the rules applied prior to r_i consume the objects it needed to continue being active. This can be used mainly in three ways to improve the execution time of the algorithm:

1. There is no need to execute the elimination step of the rule r_i eliminated in a previous step. Bearing in mind the execution trace in figure 7, if the elimination step of rule r_6 also eliminates rule r_4, then it would no longer be necessary to execute the elimination step of r_4, thus allowing execution of the algorithm to save 3 iterations.
2. Rule r_i is not to be applied in the elimination steps of subsequent rules. Bearing in mind the execution trace in figure 7, if the elimination step of rule r_6 also eliminates rule r_8, it is therefore unnecessary in elimination steps of rules r_7

and r_{10} to try to apply r_8, thus allowing execution of the algorithm to save 2 iterations.
3. Elimination of rule r_i causes a change in the composition and order of the subsequent elimination steps. Keeping in mind the execution trace in figure 7, if the elimination step of rule r_6 also eliminates rule r_8, then it is beneficial to execution of the algorithm for r_9 to be the next rule eliminated. This is the case because once r_6, r_8 and r_9 have been eliminated, r_7 and r_{10} can be eliminated in a single iteration in their elimination step since they do not share objects. Here, 3 iterations would be saved.

To implement this optimization, a determination is necessary of what rules continue to be active whenever an elimination step is performed, and this information is used to calculate the next optimal elimination step to be taken. Logically, calculation of the next optimal elimination step would severely penalize the execution time of the algorithm. Hence, a different solution must be sought. This solution involves making an analysis prior to the execution of each P system, in which we can calculate all the possible active rule sets and assign them the next optimal step of rule elimination. All this information would be reflected in a director graph of the algorithm, the definition of which is as follows:

Definition 9: *Director graph of algorithm of rule application.* Let R be a set of active rules. The director graph of the algorithm of rule application is composed of a triad $DG = (Q, A, T)$ where:

1. Q is the node set of the graph, composed of a subset of parts of R, that is: $\forall q \in Q, , q \in \mathcal{P}(R)$
2. A is a correspondence whose initial set is Q and whose final set is a set of sequences of rules composed of rules of the origin element of Q. Thus, each set of active rules has one or more sequences of rules. Each sequence of rules indicates the order in which elimination step rules are applied. So a state can have several elimination steps associated in the analysis prior the evolution of each P system.
   $A : Q \to E$ *where* E *is the set of possible sequences with elements in* Q

1. T is a set of transitions. Each transition is composed of a triad $\langle q_i, A(q_i), q_f \rangle$ where q_i, $q_f \in Q$ are the initial and final state, respectively, of the transition and $A(q_i)$ are the elimination step (s) of rules associated to state q_i, which, after being executed, means that active rules are those of state q_f.

Execution of the sequential algorithm of application of competitive rules will involve making a loop that ends when it reaches a state with no active rules. In each iteration, there are 3 steps:

1. The elimination steps associated to the state are executed.
2. Active rules are calculated.
3. The state represented by active rules is transited.

Execution of the parallel algorithm of application of competitive rules will be similar to the sequential one. The difference is that execution of several elimination steps associated to a state is performed in parallel way.

In the worse case, 3^{rd} optimization performs the same iterations as the 2^{nd} optimization. However, the experimental data obtained with the execution of the algorithm with 3^{rd} optimization are better than the 2^{nd} one, as show figure 9. In this figure, the X-axis values represent the relationship between the cardinal of the multiset and the cardinal of the sum of inputs of active rules. The Y-axis values are percentage relationship between the number of iterations with active rules elimination algorithm and sequential competitive rules with 3^{rd} optimization algorithm.

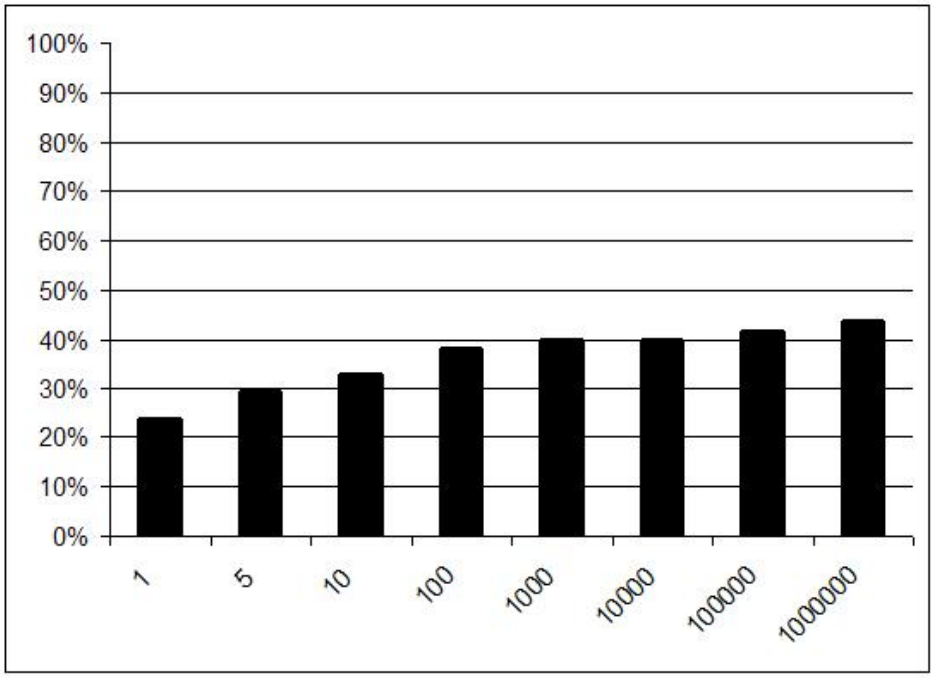

Fig. 9. Comparation between Active Rules Elimination Algorithm and Sequential Competitive Rules with 3^{rd} optimization

6 Conclusions

This paper introduces the concept of a competitiveness relationship among active rules. Based on this concept, a new way of parallelism has been opened towards the

massively parallel character needed in rules application in P systems. Moreover, the sequential version of this algorithm has a lower number of operations in its execution than in other sequential algorithms published to date.

Both the sequential and the parallel versions of the algorithm perform a limited number of operations, thus allowing for prior knowledge of the execution time. This characteristic makes both versions of the proposed algorithm appropriate for use in viable distributed architectures of implementation of P systems. This is said because architectures require determining the distribution of the number of membranes to be located in each processor of the architecture in order to obtain minimal evolution step times with the use of minimal resources.

References

1. G. Bravo, L. Fernández, F. Arroyo, J. A. Tejedor, *Master-Slave Distributed Architecture for Membrane Systems Implementation*, 8th wseas International Conference on EVOLUTIONARY COMPUTING (EC '07) (accepted)
2. G. Ciobanu, W. Guo , *P Systems Running on a Cluster of Computers*, Workshop on Membrane Computing LNCS 2933, 2004 pp.123-139
3. L. Fernández, F. Arroyo, J. Castellanos, J. Tejedor, I. García , *New Algorithms for Application of Evolution Rules based on Applicability Benchmarks*, BioInformatics & Computational Biology, 2006 pp. 94-100
4. L. Fernández, F. Arroyo, J. Tejedor, J. Castellanos, *Massively Parallel Algorithm for Evolution Rules Application in Transition P Systems*, Pre-Proceedings of Workshop on Membrane Computing (WMC7) Leiden, Netherlands, 2006, pp. 337-343
5. F. J. Gil, L. Fernndez, F. Arroyo, J. A. Tejedor, *Delimited Massively Parallel Algorithm based on Rules Elimination for Application of Active Rules in Transition P Systems*, i.TECH 2007 Fourth International Conference Information Research and Applications (accepted)
6. A. Gutiérrez, L. Fernández, F. Arroyo, V. Martínez, *Design of a hardware architecture based on microcontrollers for the implementation of membrane system*, Proceedings on 8th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC-2006). Timisoara (Rumania) September, 2006, pp. 39-42.
7. V. Martínez, L. Fernández, F. Arroyo, A. Gutiérrez, *HW Implementation of a Bounded Algorithm for Application of Rules in a Transition P-System*, Proceedings on 8th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC-2006). Timisoara, Romania, 2006, pp. 32-38.
8. Gh. Păun, *Computing with Membranes*, Journal of Computer and System Sciences, 61(2000), and Turku Center of Computer Science-TUCS Report n 208, 1998.
9. Gh. Păun, G. Rozenberg, *A Guide to Membrane Computing*, Theoretical Computer Science, vol 287, 2000. pp.73-100.
10. Gh. Păun *Membrane Computing. An Introduction*, Springer-Verlag, 2002
11. J. A. Tejedor, L. Fernández, F. Arroyo, G. Bravo, *An Architecture for Attacking the Bottleneck Communication in P Systems*, The Twelfth International Symposium on Artificial Life and Robotics, 2007, pp. 500-505
12. J. A. Tejedor, L. Fernández, F. Arroyo, A. Gutiérrez, *Algorithm of Active Rules Elimination for Application of Evolution Rules*, 8th wseas International Conference on EVOLUTIONARY COMPUTING (EC '07) (accepted)