Simulating Shuffle–Exchange Networks with P Systems

Rodica CETERCHI
Faculty of Mathematics and Computer Science
University of Bucharest
Academiei 14, RO-010014, Bucharest, Romania
E-mail: `rc@funinf.cs.unibuc.ro`

Mario J. PÉREZ-JIMÉNEZ
Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: `mario.perez@cs.us.es`

Abstract. We present in this paper a simulation with P systems of the parallel architecture known as *shuffle–exchange network*. This will lead us to consider a new version of P systems with communication, for which the communication graphs are not fixed, but have a dynamic evolution, and for which different communication rules can be associated to different communication graphs. The simulation of the shuffle–exchange network provides a first, simple example, of such a system.

1 Introduction

Any computer, whether sequential or parallel, operates by executing instructions on data. An algorithm (*stream instructions*) tells the machine what to do at each moment, and the input of an algorithm (*stream data*) is affected by the instructions of the algorithm.

A parallel machine consists of a large number of processors (each one have an arithmetic logic unit with registers and a private memory) that be able to solve problems in a cooperative way; that is, that machine is capable of executing several instructions at the same time unit. Basically there are two different kinds of parallel machines according to the possibility to execute, simultaneously, either the same instruction on different data sets (**SIMD**) or different instructions on different data sets (**MIMD**).

In [1] one deals with some sorting parallel algorithms, based on a comparison network model of computation, which incorporates features of parallelism, and a simulation of these algorithms through P systems with communication is presented.

In this paper we study the perfect shuffle **SIMD** computer where the processors interconnection is based on the *shuffle* and *exchange* functions (that is, by a *shuffle–exchange network*), and we analyze the possibility to simulate this parallel architecture by a cellular computing with membranes.

The paper is organized as follows. In Section 2 we first give some preliminary notions on the perfect shuffle **SIMD** architecture, and some interesting properties of this parallel model are presented. In Section 3 we introduce a variant of P systems with communication modelling perfect shuffle machines. Section 4 and Section 5 illustrate the above simulation with parallel algorithms to sum integer numbers and to find the maximum of a finite numbers of integer numbers. In the last section some conclusions and remarks of our study are presented.

2 Description of the Perfect Shuffle SIMD Architecture

A form of synchronous parallelism is called **SIMD** (**S**ingle–**I**nstruction–**M**ultiple–**D**ata). In the **SIMD** paradigm, there are a number of processors arranged in a special way and connected according to a singular architecture. Each processor has a private memory for storing instructions and data. The model is synchronous and during a given time unit some processors are active and execute the *same instruction*, but on a (eventually) *different data* set.

Several different methods of connecting processors in a parallel computer have been proposed. In this paper we will study the *perfect shuffle* **SIMD** *architecture* and we analyze the possibility to consider it as a membrane system.

A *shuffle–exchange network* consists of $k = 2^n$ nodes, labelled $0, 1, \ldots, k-1$, and two kinds of connections.

Exchange connections link pairs of nodes whose labels differ in the least significant bit. Thus the *exchange* function indicates that two adjacent nodes should be connected for data exchange.

The following lemma characterizes exchange connections.

Lemma 2.1 *Let $x_{n-1}x_{n-2}\ldots x_1x_0$ be the label of a node in a shuffle–exchange network, expressed in binary. Then this node is exchange connected with the node labelled by $x_{n-1}x_{n-2}\ldots x_1x_0^*$, where $x_0^* = 1 - x_0$.*

The *exchange function* $e : L \longrightarrow L$ indicates that pairs of nodes, j and $e(j)$, for all $j \in L$, should be connected for data exchange. The exchange function can be defined by $e(x_{n-1}x_{n-2}\ldots x_1x_0) = x_{n-1}x_{n-2}\ldots x_1x_0^*$, with $x_0^* = 1-x_0$, and $j = x_{n-1}x_{n-2}\ldots x_1x_0 \in L$ is the label of a node expressed in binary.

The set $\{(j, e(j)) \mid j \in L\}$ of exchange connections can be considered the set of edges of a graph, with set of vertices L (or, equivalently, the set of processors labeled by L). In the sequel we will call this graph the *exchange graph*, and denote it G_e.

Note that, from $x_0^* = 1-x_0$ follows ${x_0}^{**} = x_0$, and from this it follows that $e(e(j)) = j$ for all $j \in L$; thus, if $(j, e(j))$ is an exchange connection, then so is $(e(j), j)$, and thus the exchange graph is a non-oriented one.

Shuffle connections link a node labelled by i ($i \neq k-1$) with a node labelled by j (in the direction from i to j!) if and only if $j \equiv 2i$ modulo $k-1$ (excepting the node labelled by $k-1$ that is connected to itself); that is,

$$j = \begin{cases} 2i & \text{for } 0 \leq i \leq k/2 - 1, \\ 2i + 1 - k & \text{for } k/2 \leq i \leq k-1. \end{cases}$$

The *shuffle function* $s : L \longrightarrow L$ expresses the fact that $(i, s(i))$ is a shuffle connection for all $i \in L$.

We have the following interesting property of the shuffle connection.

Lemma 2.2 *Let $x_{n-1}x_{n-2}\dots x_1x_0$ be the label of a node in a shuffle–exchange network, expressed in binary. Then this node is shuffle connected with the node labelled by $x_{n-2}x_{n-3}\dots x_0x_{n-1}$.*

That is, the *shuffle* function corresponds to perfectly shuffling a deck of n cards, performing cyclic shifting to the left of the bits in $x_{n-1}x_{n-2}\dots x_1x_0$ through one bit position.

The set $\{(i, s(i)) \mid i \in L\}$ of shuffle connections can be considered the set of edges of a graph, with set of vertices L (or, equivalently, the set of processors labeled by L). In the sequel we will call this graph the *shuffle graph*, and denote it G_s.

Note that shuffle connections are oriented, thus the shuffle graph is an oriented graph.

The shuffle–exchange network corresponding to eight nodes is depicted in Figure 1. The arrows (oriented edges) depict the *shuffle* connections, while the double-arrows (the non-oriented edges) depict the *exchange* connections.

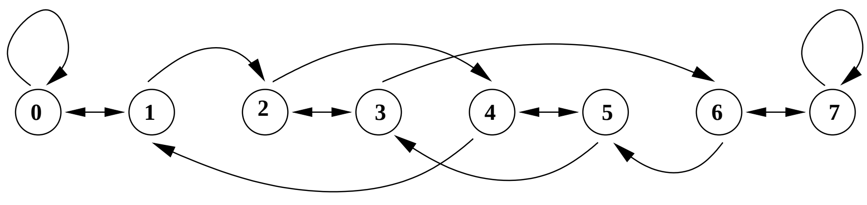

Figure 1: Shuffle–exchange network

Shuffle/exchange connections are used by the processors to perform shuffle/exchange *operations*, which consist of transferring values of internal variables “along” a given connection.

Consider a parallel architecture with processors organized by a shuffle–exchange network, where we use the binary representation of the labels of the nodes. Let the processors be loaded with k items of data $a_0, \dots, a_{k-1}$. After n applications of the shuffle operations, the data return to their original positions.

Now, we consider the data items originally in pairs of adjacent processors (by shuffle or exchange connections); that is, the corresponding binary representation differ only in a bit (for instance, in position $n - p$, where $1 \leq p \leq n$). Then, after n shuffle operations, these data items are located in adjacent processors.

3 Simulation of the Perfect Shuffle SIMD Architecture with P Systems

Let M be a perfect shuffle **SIMD** machine, with $k = 2^n$ processors, P_i (with $0 \leq i \leq k-1$). Then to each processors P_j we will associate a membrane, which we will still denote P_j. Thus, similarly to tissue-like P systems, we will have a collection of elementary membranes, no skin membrane, and the elementary membranes will be connected by certain graphs, at certain moments of their evolution in time, as we will explain in the sequel.

The *contents* of each processor P_j will be codified in its associated membrane with symbol objects. The alphabet of symbols used, V, will depend on the contents of the processors we are simulating (see the application to sum for instance). If each processor has to contain say r variables with positive integer values, then they can be in principle codified with an alphabet with r letters. In the case of the applications illustrated further, the sum and the maximum value, two variables are used by the perfect shuffle algorithms, and their simulation with P systems uses accordingly an alphabet with two symbols.

We have to model:

– *Patterns of communication* between processors. Recall that in a perfect shuffle architecture there exists basically two types of communication graphs:

  - the **shuffle** graph, which we will denote G_s: the set of vertices is composed of all membranes, the set of oriented edges is defined as in 2;
  - the **exchange** graph, which we will denote G_i: the set of vertices is composed of all membranes, the set of oriented edges is defined as in 2;

  Although the exchange graph G_i is not oriented, as opposed to the shuffle graph, G_s, we will consider all the edges of all graphs to have an orientation, and we will use the usual convention on graph theory: an edge $(r, s) \in G_i$ iff the edge $(s, r) \in G_i$.

– *Patterns of specific internal processing* in each processor; these will be modeled by symbol rewriting rules. In order to have unity of notation, we can associate such rules to the **identity** graph, G_{Id}: the set of vertices is composed of all membranes, the set of edges is defined as $\{(j, j) \mid 0 \le j \le k\}$.

The P systems which we consider in the sequel, for modelling the perfect shuffle architecture, depart from the classical P systems in (at least) two respects:

– The connections between individual membranes of a P system, μ, which was a tree-like structure of membranes (see [3]), and which in tissue-like P systems becomes a graph structure; now, in the algorithms we will simulate these connections change in time, and we can better think of them as a *sequence of graph structures.*

– The rules of a P system, usually associated to *membranes*, will now be associated to *communication graphs* between membranes.

  (a) We will simulate the internal computations performed by the processors (recall that in the case of the **SIMD** architecture, we have the same internal computations inside each processor), by the action of symbol or object rewriting rules, at work simultaneously inside each individual membrane. In order to have a unitary formalism, we will associate such rules to the identity graph, G_{Id}.

  (b) We will simulate the exchange of data performed by the processors with communication rules (symport/antiport rules) between membranes. The communication rules will be associated to graphs different from the identity graph, the graphs of different types of connections which link the processors in every particular architecture, and, for a given architecture, in every particular algorithm.

So, in our formalism, we will have pairs $[rules, graph]$ which we use to describe the behaviour of a P system which simulates the behaviour of a given algorithm, in a particular architecture.

We can impose the restriction of having a finite set of overall rules, say *Rules*, and a finite set of overall graphs, say *Graphs*, from which the members of the pairs $[rules, graph] \in P(Rules) \times P(Graphs)$ are picked, i.e., $rules \subseteq Rules$, $graph \subseteq Graphs$, and $P(A) = \{x \mid x \subseteq A\}$ represents the power set of A.

Furthermore, we will allow a flexible use of rules and graphs, in the sense that, for a given graph, G, we can use pairs $[R_1, G]$ and $[R_2, G]$, with $R_1 \neq R_2$ at different moments in time. This corresponds to the fact that the same connection graph between processors can be used (even by the same algorithm) at different moments in time to perform different data exchange, or, when simulating the internal computations, which again are different at different moments in time.

For the perfect shuffle architecture, we will have

$$Graphs = \{G_s, G_i, G_{Id}\}.$$

The set *Rules*, of all symbol/object rewriting rules which simulate internal computations performed by the individual processors, will depend on the particular algorithm, used to solve a particular problem, within the framework of a given architecture.

Since we model *deterministic algorithms*, each *iterative step* of such an algorithm will be modeled by *a finite sequence of pairs* $[rules, graph]$, each pair simulating the effect of "an instruction". The finite sequence of pairs $[rules, graph]$, corresponding to an iterative step of the algorithm, will be denoted in the sequel by R_μ, and will be applied repeatedly (but a finite number of times) to the elementary membranes simulating the processors and their initial configuration. Our systems will thus exhibit a *periodic* behaviour, and in their presentation we will use *the period* R_μ, which takes the place of both rules associated to membranes and communication graph(s).

A P system which simulates a particular algorithm in the perfect shuffle SIMD architecture will thus be a construct

$$\Pi = (V, P_0, \cdots, P_{k-1}, R_\mu),$$

where $P_0, \cdots, P_{k-1}$ are elementary membranes, V is an alphabet of symbols used to codify the contents of the membranes, and R_μ is a finite sequence of pairs $[rules, graph]$, such that: (i) if $graph = G_{Id}$ then its rules are rewriting rules; (ii) if $graph \neq G_{Id}$ then its rules are communication rules, and all the graphs in this category belong to the shuffle–exchange set, $\{G_s, G_e\}$, i.e., are either of type *shuffle* or of type *exchange.* We will call such a system a *P system with periodic dynamic communication.*

We illustrate in the next two sections such types of P systems.

4 A Sum Algorithm with Perfect Shuffle and its Simulation

The perfect shuffle architecture can be used to compute the sum of $k = 2^n$ integers $a_0, a_1, \cdots, a_{k-1}$. Processor P_i, labelled by i, possesses two local variables: s_i (initialized by a_i) and t_i (initialized by 0), for all i such that $0 \leq i \leq k-1$. In the variable t_i, processor P_i will put the data communicated for an adjacent processor.

The following procedure compute the sum $a_0 + a_1 + \cdots + a_{k-1}$.

```
procedure sum_PS($a_0, a_1, \cdots, a_{k-1}$)
begin
   for $i \leftarrow 1$ to log $k$ do
      for all $j$ where $0 \leq j \leq k-1$ doPS-simPsys
         shuffle ($s_j$)
         $t_j \leftarrow s_j$
         exchange ($t_j$)
```

```
            $s_j \leftarrow s_j + t_j$
        endfor
    endfor
end
```

Here `shuffle` (s_j) means that processor P_r links by shuffle processor P_j, then P_r sends the value of s_r and processor P_j put this value in the variable s_j. The operation `exchange`(t_j) means that processors P_r and P_j connected by an exchange-edge interchange the corresponding values of their variables t.

The result is collected in the processor P_0. In this algorithm there are $\log(k)$ iterations and every iteration takes constant time, so this parallel algorithm works in $\Theta(\log(k)$ time.

Example: Take $n = 3$, thus $k = 8$, and consider the eight positive integers $a_0 = 2$, $a_1 = 3$, $a_2 = 1$, $a_3 = 1$, $a_4 = 4$, $a_5 = 5$, $a_6 = 3$, $a_7 = 2$, each integer a_i represented as the variable a_i of the processor P_i, for $0 \leq i \leq k-1$.

Figure 2 shows that, after three iterations, the sum $a_0 + a_1 + \cdots + a_7 = 21$ is obtained in the a_i variable of each processor P_i, $0 \leq i \leq 7$, and also shows the contents of each processor in terms of the values of the a_i variables.

Simulation of the Example with P systems: Consider that to each processor P_i, $0 \leq i \leq 7$, there corresponds a membrane, still denoted P_i, $0 \leq i \leq 7$. Let the positive integer a_i be codified as a_i occurrences of a symbol a, and the positive integer b_i be codified as b_i occurrences of a symbol b, different from a.

- *The initial configuration:* for every $0 \leq i \leq 7$, every membrane P_i contains a^{a_i}, and no b's.
- *The shuffle graph:* is denoted by G_s and contains the following set of arcs
  $$\{(0,0),(7,7),(1,2),(2,4),(4,1),(3,6),(6,5),(5,3)\}.$$
  To the shuffle graph G_s we associate the unique symport rule: (a, out).
- *The exchange graph:* is denoted by G_e and contains the following set of arcs
  $$\{(0,1),(1,0),(2,3),(3,2),(4,5),(5,4),(6,7),(7,6)\}.$$
  To the exchange graph G_i we associate the unique symport rule: (b, out).
- *The identity graph:* is denoted by G_{Id} and contains the following set of arcs
  $$\{(i,i) \mid 0 \leq i \leq 7\}.$$
  We will use the identity graph for *internal* computations, which in this case will be rewritings.

Consider the following sequence of pairs $[rule, graph]$:

$$[(a, out), G_s], [a \to ab, G_{Id}], [(b, out), G_e], [b \to a, G_{Id}],$$

and apply it to the initial configuration of our eight membranes.

– action of $[(a, out), G_s]$: rule (a, out) is active in all membranes; "out" is relative to the graph G_s: if an edge $(i, j) \in G_s$, then all objects a of P_i get transported into

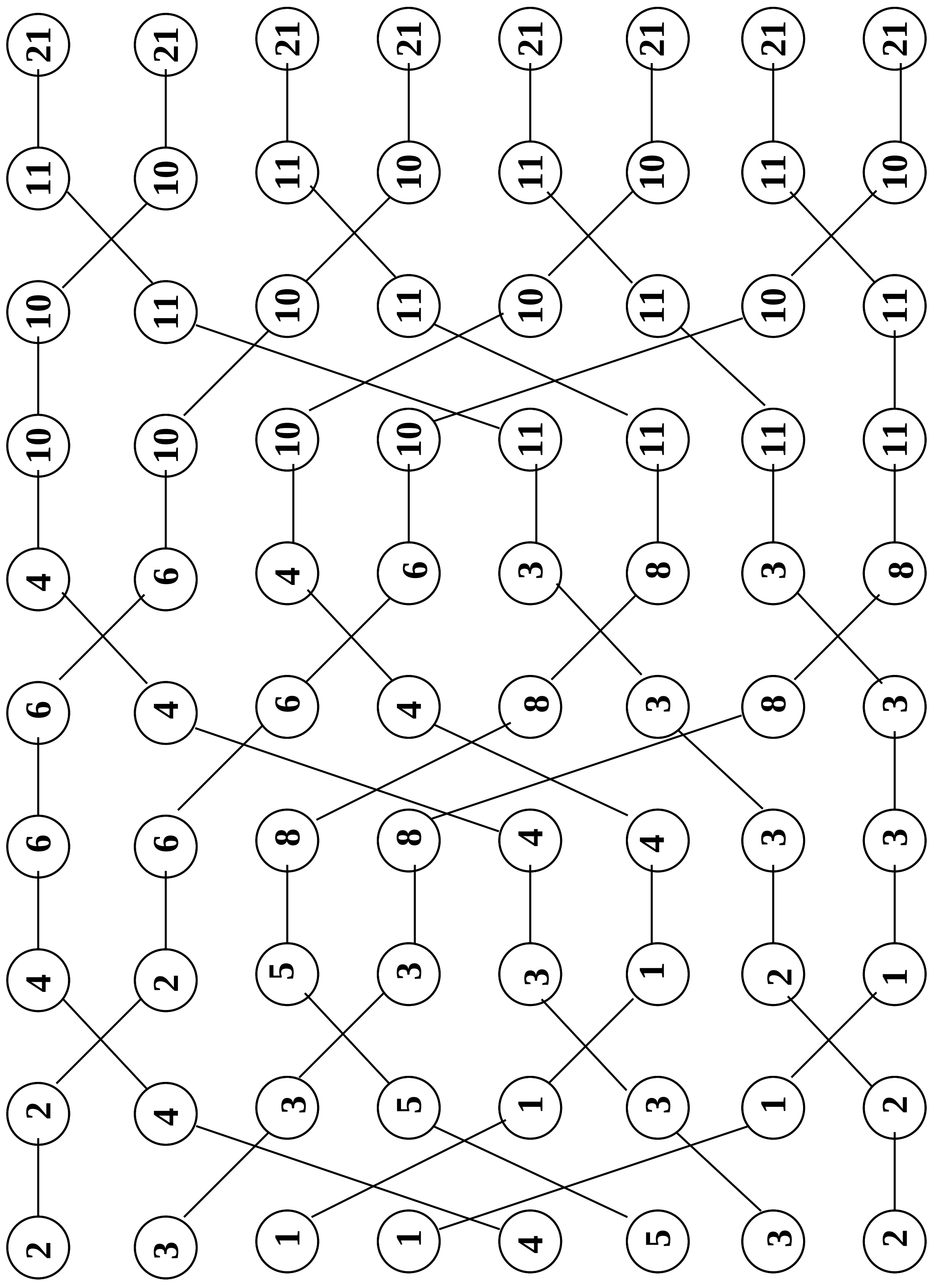

Figure 2: Sum Perfect–Shuffle

P_j "using" this edge. In our particular example, this means: membranes P_0 and P_7 remains with the same content of a's, since each is connected only to itself; edges $\{(1,2),(2,4),(4,1)\}$ form a cycle, thus the a's of P_1 get into P_2, those of P_2 into P_4, and those of P_4 into P_1; similarly for the edges $\{(3,6),(6,5),(5,3)\}$. Each membrane will hold after this application the following integers (represented with a's):

$$P_0 : a^{a_0},\ P_1 : a^{a_4},\ P_2 : a^{a_1},\ P_3 : a^{a_5},\ P_4 : a^{a_2},\ P_5 : a^{a_6},\ P_6 : a^{a_3},\ P_7 : a^{a_7}.$$

For our particular example, we get

$$P_0 : a^2,\ P_1 : a^4,\ P_2 : a^3,\ P_3 : a^5,\ P_4 : a^1,\ P_5 : a^3,\ P_6 : a^1,\ P_7 : a^2.$$

– action of $[a \to ab, G_{Id}]$: this is a rewriting step taking place in each membrane, using the rule $a \to ab$; the graph G_{Id} plays no role, there is no communication taking place. The effect of the rewriting rule is to produce in each membrane the same number of b's as the number of a's it is already holding. For our example, after this application, we obtain the following content:

$$P_0 : a^2b^2,\ P_1 : a^4b^4,\ P_2 : a^3b^3,\ P_3 : a^5b^5,\ P_4 : a^1b^1,\ P_5 : a^3b^3,\ P_6 : a^1b^1,\ P_7 : a^2b^2.$$

– action of $[(b, out), G_e]$: our membranes are now connected by the exchange graph G_e; the rule in each membrane is the symport rule (b, out); since G_e contains pairs of edges $\{(j, j+1),(j+1, j)\}$ for all $j = 0, 2, 4, 6$, then for all such pairs the contents of b's of P_j and P_{j+1} are swapped. For our example, after this application, we obtain the following content:

$$P_0 : a^2b^4,\ P_1 : a^4b^2,\ P_2 : a^3b^5,\ P_3 : a^5b^3,\ P_4 : a^1b^3,\ P_5 : a^3b^1,\ P_6 : a^1b^2,\ P_7 : a^2b^1.$$

– action of $[b \to a, G_{Id}]$: this is again a rewriting step, taking place in each membrane, using the rule $b \to a$; the graph G_{Id} plays no role, there is no communication taking place. Rewriting b's to a's in a membrane has the effect of "adding" the integers represented by the a's and by the b's, the result being written with a's (and we can think that the b's counter is set to zero, since b's are dissapearing): $a^xb^y \to a^{x+y}$. For our example, after this application, we obtain the following content:

$$P_0 : a^2a^4 = a^6,\ P_1 : a^4a^2 = a^6,\ P_2 : a^3a^5 = a^8,\ P_3 : a^5a^3 = a^8,\ P_4 : a^1a^3 = a^4,$$

$$P_5 : a^3a^1 = a^4,\ P_6 : a^1a^2 = a^3,\ P_7 : a^2a^1 = a^3.$$

Note that this is precisely the integer content of the processors we are simulating, after the first iteration step.

Next, we will apply again the quadruple sequence

$$[(a, out), G_s], [a \to ab, G_{Id}], [(b, out), G_e], [b \to a, G_{Id}],$$

of pairs $[rule, graph]$, to the above configuration of our eight membranes. After working out the details, as above, we obtain that our eight membranes will have the following content of integers represented with a's:

$$P_0 : a^{10},\ P_1 : a^{11},\ P_2 : a^{10},\ P_3 : a^{11},\ P_4 : a^{10},\ P_5 : a^{11},\ P_6 : a^{10},\ P_7 : a^{11},$$

and this is precisely the integer content of the processors we are simulating, after the second iteration step.

Applying again the quadruple sequence

$$[(a, out), G_s], [a \to ab, G_{Id}], [(b, out), G_e], [b \to a, G_{Id}],$$

to the previously obtained configuration, will have the effect of the third iteration on the processors, namely obtaining the "final" configuration,

$$P_0 : a^{21},\ P_1 : a^{21},\ P_2 : a^{21},\ P_3 : a^{21},\ P_4 : a^{21},\ P_5 : a^{21},\ P_6 : a^{21},\ P_7 : a^{21},$$

which contains the sum, namely 21, of the initially given integers.

In a completely similar manner to the example above, we can simulate the **PS** algorithm for computing the sum of arbitrary $k = 2^n$ positive integers, using k membranes, labeled from 0 to $k - 1$, starting from an initial configuration which codifies integer a_j as a^{a_j} in each membrane P_j, and applying $n = log_2(k)$ times the sequence

$$[(a, out), G_s], [a \to ab, G_{Id}], [(b, out), G_e], [b \to a, G_{Id}].$$

Theorem 4.1 *For $k = 2^n$, the P system with periodic dynamic communication*

$$\Pi = (\{a, b\}, P_0, \cdots, P_{k-1}, R_\mu = \{[(a, out), G_s], [a \to ab, G_{Id}], [(b, out), G_e], [b \to a, G_{Id}]\})$$

computes the sum of k positive integers $\{a_0, \cdots, a_{k-1}\}$, starting from the initial configuration of objects

$$\{P_j : a^{a_j} \mid 0 \leq j \leq k - 1\},$$

and applying the finite sequence

$$([(a, out), G_s], [a \to ab, G_{Id}], [(b, out), G_e], [b \to a, G_{Id}])^n.$$

The sum $S = \Sigma_{j=0}{}^{k-1} a_j$ is obtained in each membrane, i.e., as the configuration

$$\{P_j : a^S \mid 0 \leq j \leq k - 1\}.$$

It is obvious that, in order to add k positive integers, with $2^{n-1} < k \leq 2^n$, (i.e., k not necessarily a power of 2) the same system with 2^n membranes can be used, letting the extra membranes hold the value zero (i.e. no occurrences of a).

Until now we have dealt with adding only positive integers. Our model needs only a small refinement in order to deal with both positive and negative integers. Let, as before,

- a be the symbol to codify the positive integer content of the a_j field of every membrane P_j;
- b be the symbol to codify the positive integer content of the b_j field of every membrane P_j.

Let $|x|$ denote the positive part of the integer x. Consider two supplementary symbols, $\bar{a}$ and $\bar{b}$, to represent respectively the negative units. In other words, let

- $\bar{a}$ be the symbol to codify the negative integer content of the a_j field of every membrane P_j; if a_j is negative, it will be codified as $\bar{a}^{|a_j|}$;

- $\bar{b}$ be the symbol to codify the negative integer content of the b_j field of every membrane P_j; if b_j is negative, it will be codified as $\bar{b}^{|b_j|}$.

The graphs G_s, G_e, and G_{Id} are defined as before. We only add more rules to each of them, to ensure also the communication of negative units, and the proper addition. The corresponding four pairs of $[rules, graph]$ to be applied in sequence are:

- $[\{(a, out), (\bar{a}, out)\}, G_s]$
- $[\{a \to ab, \bar{a} \to \bar{a}\bar{b}\}, G_{Id}]$
- $[\{(b, out), (\bar{b}, out)\}, G_e]$
- $[\{b \to a, \bar{b} \to \bar{a}, a\bar{a} \to \lambda\}, G_{Id}]$

Suppose a membrane holds a positive integer x as a^x, and that, after the action of the exchange graph, receives a negative integer y, represented as $\bar{b}^{|y|}$. At the next step, a rewriting one, governed by $[\{b \to a, \bar{b} \to \bar{a}, a\bar{a} \to \lambda\}, G_{Id}]$, the rule $\bar{b} \to \bar{a}$ will rewrite all occurrences of $\bar{b}$ as $\bar{a}$, thus we get $a^x\bar{a}^{|y|}$, and the rule $a\bar{a} \to \lambda$ will ensure the proper addition of the positive and the negative integer. We have thus the result:

Theorem 4.2 *For $k = 2^n$, the P system with periodic dynamic communication*

$$\Pi = (\{a, b, \bar{a}, \bar{b}\}, P_0, \cdots, P_{k-1}, \bar{R}_\mu),$$

where

$$\bar{R}_\mu = \{[\{(a, out), (\bar{a}, out)\}, G_s], [\{a \to ab, \bar{a} \to \bar{a}\bar{b}\}G_{Id}],$$
$$[\{(b, out), (\bar{b}, out)\}, G_e], [\{b \to a, \bar{b} \to \bar{a}, a\bar{a} \to \lambda\}, G_{Id}]\}$$

computes the sum of k integers $\{a_0, \cdots, a_{k-1}\}$, starting from the initial configuration of objects

$$\{P_j : a^{a_j} \mid 0 \le j \le k-1, \text{ such that } a_j \ge 0\}, \{P_j : \bar{a}^{|a_j|} \mid 0 \le j \le k-1, \text{ such that } a_j < 0\}$$

and applying the finite sequence $\bar{R}_\mu$ n times.

The sum $S = \Sigma_{j=0}{}^{k-1} a_j$ is obtained in each membrane, i.e. as the configuration

$$\{P_j : a^S \mid 0 \le j \le k-1\} \text{ if } S \ge 0, \text{ or } \{P_j : \bar{a}^{|S|} \mid 0 \le j \le k-1\} \text{ if } S < 0.$$

5 Finding the Maximum with Perfect Shuffle and its Simulation

The perfect shuffle architecture can be also used to compute the maximum of $k = 2^n$ integers $a_0, a_1, \cdots, a_{k-1}$. Processor P_i, labelled by i, possesses two local variables: s_i (initialized by a_i) and t_i (initialized by 0, for all i such that $0 \le i \le k-1$). In the variable t_i, processor P_i will put the data communicated from an adjacent processor. We suppose that in each processor P_i, a function which returns the maximum value of the local variables s_i and t_i can be implemented.

The following procedure computes the maximum of $a_0, a_1, \cdots, a_{k-1}$.

```
procedure max_PS($a_0, a_1, \cdots, a_{k-1}$)
```

```
begin
   for $i \leftarrow 1$ to log $k$ do
      for all $j$ where $0 \le j \le k-1$ do
         shuffle $(s_j)$
         $t_j \leftarrow s_j$
         exchange $(t_j)$
         $s_j \leftarrow max(s_j, t_j)$
      endfor
   endfor
end
```

where `shuffle` (s_j) and `exchange`(t_j) have the same meaning as in the sum algorithm of Section 4. Note actually that the two algorithms differ only in the internal processing occurring at the fourth step of each iteration, namely, instead of making the sum $s_j + t_j$ and storing it in variable s_j, the present algorithm finds the maximum of the two variables, $max(s_j, t_j)$, and stores the result again in variable s_j.

In this algorithm also there are $\log(k)$ iterations, and every iteration takes constant time, so this parallel algorithm works also in $\Theta(\log(k))$ time. The result can be considered collected in P_0, it will hold the maximum of given k numbers after $\log(k)$ iterations.
Remark: In general, the perfect shuffle **SIMD** architecture provides an interesting tool to solve problems with $k = 2^n$ inputs an a single output, that involves only an associative binary operation (the *sum* in the example of the section 4, and the *maximum* in the example of the section 5).

Example: Take as before, $n = 3$, thus $k = 8$, and consider the eight positive integers $a_0 = 2$, $a_1 = 3$, $a_2 = 1$, $a_3 = 1$, $a_4 = 4$, $a_5 = 5$, $a_6 = 3$, $a_7 = 2$, each integer a_i represented as the variable a_i of the processor P_i, for $0 \le i \le k-1$.

After three iterations, the maximum of $a_0, a_1, \cdots, a_7$, i.e. value 5, is obtained in the a_i variable of each processor P_i, for $0 \le i \le 7$.

Simulation of the Example with P systems: This will be very similar to the simulation of the sum example. As in the previous section, consider that to each processor P_i, $0 \le i \le 7$, there corresponds a membrane, still denoted P_i, $0 \le i \le 7$. Let the positive integer a_i be codified as a_i occurrences of a symbol a, and the positive integer b_i be codified as b_i occurrences of a symbol b, different from a.

- *The initial configuration:* for every $0 \le i \le 7$, every membrane P_i contains a^{a_i}, and no b's.
- *The shuffle graph:* is denoted by G_s and contains the following set of arcs
  $$\{(0,0), (7,7), (1,2), (2,4), (4,1), (3,6), (6,5), (5,3)\}$$
  To the shuffle graph G_s we associate the unique symport rule: (a, out).
- *The exchange graph:* is denoted by G_e and contains the following set of arcs
  $$\{(0,1), (1,0), (2,3), (3,2), (4,5), (5,4), (6,7), (7,6)\}$$
  To the exchange graph G_e we associate the unique symport rule: (b, out).
- *The identity graph:* is denoted by G_{Id} and contains the following set of arcs

$\{(i,i) \mid 0 \leq i \leq 7\}$.

We will use the identity graph for *internal* computations, which in this case will be rewritings.

Consider the following sequence of pairs $[rule, graph]$:

$$R_\mu{}^{max} = ([(a, out), G_s], [a \to ab, G_{Id}], [(b, out), G_e], [\{ab \to a > b \to a\}, G_{Id}]).$$

Note that it differs from the sequence R_μ, considered in the previous Section for the sum algorithm, only in the fourth pair, $[\{ab \to a > b \to a\}, G_{Id}]$. The role of this pair is to simulate the computation of the maximum in each processor/membrane, and putting the result in the variable codified with the a symbol. Note that it uses two rewriting rules, ordered by a priority relation, $\{ab \to a > b \to a\}$.

Consider a membrane holding the values of two positive integers, x and y, and using symbols a and b for codifying them, for instance x is represented as a^x, and y is represented as b^y. We have precisely $min(x, y)$ pairs ab in our membrane, to which rewriting rule $ab \to a$ is applicable, so they will be rewritten as a's. If there are b's left, i.e., if $y > x$, then for every such b the second rewriting rule, $b \to a$, is applicable, thus precisely $y - x$ occurrences of b are rewritten as a's. If there are no b's left, i.e. $y \leq x$, the second rule is not applicable (but we still have $x - y$ occurrences of a left untouched in the membrane). In either situation, after application of the rules, the membrane holds $a^{max(x,y)}$ and no b's.

We have a result similar to Theorem 4.1.

Theorem 5.1 *For $k = 2^n$, the P system with periodic dynamic (periodic) communication graph*

$$\Pi = (\{a, b\}, P_0, \cdots, P_{k-1}, R_\mu{}^{max}),$$

where

$$R_\mu{}^{max} = ([(a, out), G_s], [a \to ab, G_{Id}], [(b, out), G_e], [\{ab \to a > b \to a\}, G_{Id}]),$$

computes the maximum of k positive integers $\{a_0, \cdots, a_{k-1}\}$, starting from the initial configuration of objects

$$\{P_j : a^{a_j} \mid 0 \leq j \leq k-1\},$$

and applying the finite sequence $R_\mu{}^{max}$ n times, i.e.,

$$([(a, out), G_s], [a \to ab, G_{Id}], [(b, out), G_e], [\{ab \to a > b \to a\}, G_{Id}])^n.$$

The maximum $M = max\{a_j \mid 0 \leq i \leq k-1\}$ is obtained in each membrane, i.e. as the configuration

$$\{P_j : a^M \mid 0 \leq j \leq k-1\}.$$

6 Final Remarks

We have explored in this paper the possibility of simulating the **SIMD** paradigm in the formal framework of P systems, a formal framework naturally equipped for dealing with parallelism. Within this paradigm, we have explored the possibilities offered by the Shuffle–Exchange networks. This has led us to consider P systems with a tissue-like behaviour and structure, but which depart from the tissue-like P systems considered so far in the literature on the subject, in at least two respects: the underlying graph has *dynamic* features, and, moreover, *rules are associated to (comunication) graphs* instead of membranes.

We have illustrated the possibilities of such a concept with the simulation of two S–E algorithms, one for computing the sum, the other for computing the maximum, of a given number of integers.

The P systems presented here exhibit *periodic* behaviour with respect to finite sequences $[rules, graph]$, which simulate the one-step-iteration of a algorithm. Other patterns beside periodic ones are worth considering for their generative power.

Acknowledgement. The support this research through the project TIC2002-04220-C03-01 of the Ministerio de Ciencia y Tecnología of Spain, cofinanced by FEDER funds, is gratefully acknowledged.

References

[1] Ceterchi, R., Martín–Vide, C.: P systems with Communication for Static Sorting. In M. Cavaliere, C. Martín–Vide and G. Păun (eds.), *Proceedings of the Brainstorming Week on Membrane Computing*, Report GRLMC 26/03, 2003, 101–117.

[2] Păun, A., Păun, G.: The power of Communication: P Systems with Symport/Antiport. *New Generation Computers*, 20, 3 (2002), 295–306.

[3] Păun, G.: Computing with Membranes. *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143, and *Turku Center for CS-TUCS Report* No. 208, 1998.

[4] Păun, G.: *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, Heidelberg, 2002.

[5] Quinn, M.J.: *Parallel Computing. Theory and Practice*, McGraw–Hill Series in Computer Science, 1994.