On Two-Dimensional Mesh Networks and Their Simulation with P Systems

Rodica Ceterchi[1] and Mario J. Pérez–Jiménez[2]

[1] Faculty of Mathematics and Computer Science, University of Bucharest,
Academiei 14, 010014 Bucharest, Romania
`rc@funinf.cs.unibuc.ro`

[2] Research Group on Natural Computing,
Department of Computer Science and Artificial Intelligence,
University of Sevilla,
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
`Mario.Perez@cs.us.es`

Abstract. We analize in this paper the possibility of simulating the parallel architecture **SIMD-MC^2**, also known as the two-dimensional mesh, with P systems with dynamic communication graphs. We illustrate this simulation for an algorithm which computes the sum of given integers. Next, we show how to extend the formalism to the reduction problem.

1 Introduction

P systems are powerful computational devices, with a high degree of parallelism, whose functioning is inspired by biological processes at the level of the cells, and of their membranes ([6], [7]). Among these processes, communication plays an important role (see [5]).

We have started in previous work ([2], and [3]) to analyze the possibility of simulating (classical) parallel architectures with P systems. A parallel machine consists of a large number of processors (each one having an arithmetic logic unit with registers and a private memory) able to solve problems in a cooperative way. The "cooperation" (sharing of data among processors) is accomplished via a specific communication network which characterizes the architecture.

We have considered in [2], and [3], the case of the *shuffle–exchange* architecture. In the present paper we deal with a different type of architecture, the *two-dimensional mesh*, in which the processors are placed in the vertices of a 2D-lattice in the plane, and communication is possible only between adjacent processors.

In the course of this study, a new type of P systems has emerged: P systems with *dynamic communication graphs* of specific types. They are in a way similar to tissue-like P systems, the connections between elementary membranes being described by graph structures, but they have a dynamic behavior: the underlying graph structures change in time. Moreover, rules, which are generally associated to regions inside membranes, are in this new version associated to underlying graphs. This new formalism covers both the simulation of internal processing, modelled by symbol rewriting rules, and the communication of data, modelled by symport/antiport rules.

The paper is organized as follows. Section 2 describes briefly the 2D-mesh parallel architecture. In Section 3 we introduce the P systems *with dynamic communication of 2D-mesh type*, the tools with which we accomplish the desired simulation. Section 4 illustrates an application of the 2D-mesh architecture to an algorithm for computing the sum of a set of integers, and contains a proof of its correctness. In Section 5 we discuss several simulations of the sum algorithm with P systems with dynamic communication of 2D-mesh type. In the next two sections we give some indications on how to extend the formalism to solve a general reduction problem.

2 The 2D-Mesh Architecture

Recall that a parallel machine consists of a large number of processors (each one having an arithmetic logic unit with registers and a private memory) able to solve problems in a cooperative way; that is, the machine is capable of executing several instructions in the same time unit.

According to Flynn's classification of computers (see [4]), a form of synchronous parallelism is called **SIMD** (Single–Instruction–Multiple–Data). A **SIMD** machine consists of a set of identical processors capable of simultaneously performing the same instruction issued by a central control unit, on different sets of data, and in a synchronous manner: each processor executing an instruction in parallel must be allowed to finish before the execution of the next instruction starts.

Several different methods of connecting processors in a parallel computer have been proposed. Quinn [8], quoting Ullman [9], mentions six important processor organizations, among which the **mesh network**. In a mesh network the processors are arranged into a q-dimensional lattice, and communication is allowed only between neighboring nodes, hence interior nodes communicate with $2q$ other processors. A **SIMD** machine in which the processors may communicate with each other via a mesh-connected network of dimension q is called in [8] a **SIMD-MC**^{q} machine.

We deal in this paper with **SIMD-MC**^{2} machines, i.e., a set of processors working according to the **SIMD** paradigm, and able to communicate (share) data among them according to a two–dimensional lattice architecture, which we will call in the sequel the **2D-mesh architecture**.

In general, in a given parallel architecture, we say that two processors are *adjacent* if they are directly connected. The *distance* between a pair of processors in a given architecture is the smallest length of a path between the processors. The *diameter* of an architecture is defined as the largest distance between any processors in the network. The *two–dimensional mesh* architecture provides a network with a large number of communication links connected to each processor, permitting to reduce the diameter of the network.

In general, we consider a *two–dimensional mesh–connected* parallel computer as a SIMD machine consisting of $n \times m$ identical processors, $P_{11}, \ldots, P_{nm}$, placed in the nodes of a 2D-lattice, or, equivalently, arranged in a 2D array.

Each processor has a local memory consisting of a number of registers, and it can perform a number of operations on data stored in these registers.

The communication between processors (mainly transmission of values of local variables) can take place only according to the 2D-lattice structure of the underlying network. Thus the processor placed in row i and column j, is denoted by $P_{i,j}$, (or P_{ij}) with $1 \leq i \leq n$ and $1 \leq j \leq m$. If $P_{i,j}$ is an interior node, i.e., $i \neq 1$, $j \neq 1$, $i \neq n$, $j \neq m$, then $P_{i,j}$ has four neighbors: $P_{i-1,j}$ and, $P_{i+1,j}$ on the same column, j, and $P_{i,j-1}$, $P_{i,j+1}$ on the same row, i. The rows $i = 1$ and $i = n$ are the *boundary rows*, and columns $j = 1$ and $j = m$ are the *boundary columns*. The nodes at the intersection of a boundary row and a boundary column have each one precisely two neighbors, and the rest of the boundary nodes have three neighbors.

3 P Systems with Dynamic Communication Graphs of 2D-Mesh Type

The model we develop here for the simulation of the 2D-mesh architecture is along the same general lines as the model proposed for the shuffle-exchange networks in [2] and [3], with some differences which arise inherently from the differences in the two parallel architectures in question.

We have a **SIMD-MC**^{2} machine, composed of $n \times m$ processors, denoted P_{ij}, with $1 \leq i \leq n$, $1 \leq j \leq m$, organized in a 2D-mesh architecture. To each processor P_{ij} we will associate a membrane, which we will still denote P_{ij}. Similarly to tissue-like P systems, we will have a collection of elementary membranes, connected by certain graphs, at certain moments of their evolution in time. The graphs we will consider will be sub-graphs of the total graph of the 2D-mesh network, also sub-graphs of the identity graph of the 2D-mesh network, as we will explain in the sequel.

The *contents* of each processor P_{ij} 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 applications in the following sections). If each processor has to contain say r variables with positive integer values, they can be in principle codified with an alphabet with r letters. In the case of the applications illustrated further, we will have to represent at most two integer variables. Their simulation with P systems will use accordingly an alphabet with two (or four) symbols.

Basically, we have to model:

– *Patterns of specific internal processing* in each processor: these will be modelled by symbol rewriting rules.
– *Patterns of communication* between processors.

Recall that in the 2D-mesh architecture the communication between processors takes place along edges which connect two neighboring processors. As we have done in general for parallel architectures, and, in particular, for the perfect–shuffle architecture in previous work (see [2], [3]), we will speak of the (underlying) *communication graph* associated to a given architecture: the *vertices* of the graph are the processors, and the *edges* (oriented or not) are the network connections characteristic of the architecture. In the case of the 2D-mesh, the underlying communication graph is composed of all edges between neighboring nodes. We will call it the *total graph*, and we distinguish between *horizontal edges* and *vertical edges*. We use the following notation:

$$G_{total} = G_h \cup G_v = \bigcup_{i=1}^{n} G_{i*} \cup \bigcup_{j=1}^{m} G_{*j},$$

where

$$G_{i*} = \{((i,j),(i,j+1)) \mid 1 \leq j \leq m-1\}, \text{ for all } 1 \leq i \leq n,$$

$$G_{*j} = \{((i,j),(i+1,j)) \mid 1 \leq i \leq n-1\}, \text{ for all } 1 \leq j \leq m.$$

G_{i*} is the set of all horizontal edges on line i, with $1 \leq i \leq n$, and G_{*j} is the set of all vertical edges on column j, for $1 \leq j \leq m$.

For every particular algorithm implemented on a 2D-mesh network of processors, not all edges of the communication graph are used simultaneously for transmitting values of local variables, as we will see in the illustrations which follow. For this reason we will speak of the total *virtual* communication graph, and of *active sub-graphs* of G_{total}, composed of sets of edges along which actual communication takes place, in parallel, at certain steps of a given algorithm.

For modelling the internal processing steps, in order to have unity of notation, we will associate the rules to the *identity* graph: the set of vertices is composed of all processors/membranes, the set of edges is defined as

$$Id = \{((i,j),(i,j)) \mid 1 \leq i \leq n, 1 \leq j \leq m\}.$$

If an internal processing occurs only in a subset of the processors/membranes, then we will consider the respective *active sub-graphs* of Id. For instance, if internal processing occurs only for processors/membranes on line i, we will associate it with the active sub-graph

$$Id_{i*} = \{((i,j),(i,j)) \mid 1 \leq j \leq m\}.$$

Similarly, if an internal processing occurs only for processors/membranes on column j, we will associate it with the active sub-graph

$$Id_{*j} = \{((i,j),(i,j)) \mid 1 \leq i \leq n\}.$$

The P systems which we consider in the sequel, for modelling the 2D-mesh architecture, similarly to those considered in [2] and [3], depart from the classical P systems in two respects:

– The connections between individual membranes of a P system, μ, which was a tree-like structure of membranes (see [6]), and which in tissue-like P systems becomes a graph structure, is now, a *sequence of graphs.*
– The rules of a P system, usually associated to *membranes*, will now be associated to *communication graphs* between membranes.

  (a) We simulate the internal computations performed by a subset of processors by the action of symbol or object rewriting rules, at work simultaneously inside the corresponding subset of membranes. We will associate such rules to the corresponding active subsets of Id.
  (b) We simulate the exchange of data performed by the processors with communication rules (symport/antiport rules) between membranes. The communication rules will be associated to the active sub-graphs of G_{total}.

We will consider the edges to have an *orientation* which gives meaning to the *in* and *out* of the symport/antiport rules: *out* means travelling in the sense of the edge's orientation, *in* means travelling in the opposite sense. Thus, rules such as (a, out), (a, in) function along an oriented edge in the same way they would function if they were attached to the source vertex of the edge.

As in [2] and [3] (with slight modifications), we will use pairs $[graph, rules]$ to describe the evolution of a P system which simulates the behavior of a given algorithm, in the 2D-mesh architecture.

For every particular architecture, its underlying network structure imposes restrictions on the set *Graphs* to which the first member of a pair $[graph, rules]$ can belong. For the 2D-mesh architecture *Graphs* is either a subset of G_{total}, or a subset of 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.

We model *deterministic algorithms*, each *iterative step* of such an algorithm will be modelled by *a finite sequence of pairs* $[graph, rules]$ (sometimes each pair simulating the effect of "an instruction", but not necessarily). The entire execution of such an algorithm will be modelled by a sequence of pairs $[graph, rules]$, denoted in the sequel by R_μ.

A P system which simulates a particular algorithm in the 2D-mesh architecture will thus be a construct

$$\Pi = (V, P_{11}, \cdots, P_{nm}, R_\mu),$$

where $P_{11}, \cdots, P_{nm}$ 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 $[graph, rules]$, such that: (i) if $graph \subset Id$, then its rules are rewriting rules; (ii) if $graph \subset G_{total}$, then its rules are communication rules. We will call such a system a *P system with dynamic communication of 2D-mesh type.*

The P system starts in an initial configuration, with its elementary membranes $P_{11}, \cdots, P_{nm}$ simulating the initial configuration of the corresponding processors. Each application of an element $[graph, rules] \in R_\mu$ to a configuration consists of considering the (active) graph *graph* and applying the rules *rules* associated to it: (i) if $graph \subset Id$, the corresponding rewriting rules are applied in the membranes which are vertices of *graph*; (ii) if $graph \subset G_{total}$, the corresponding symport/antiport rules are applied along the edges of *graph*. This leads to the next configuration. The final configuration is obtained after the application of the entire sequence R_μ.

Note that the general presentation of a P system with dynamic communication graph which simulates a given (arbitrary) parallel architecture based on communication networks (denote it by X) is the same: it specifies an alphabet of symbols used to codify the contents of membranes, a finite set of elementary membranes, and, finally, a finite sequence R_μ of pairs $[graph, rules]$. The specifics of each architecture X impose certain particular forms for the sets *Graphs* to which the first member of a pair $[graph, rules]$ can belong, and for the pairing $[graph, rules]$. Further, the specifics of each architecture, and sometimes of an algorithm implemented on it, govern the *structure* of the entire sequence R_μ.

Denote by $Graphs(X)$ the active graphs associated to the architecture X. For X = perfect shuffle SIMD = **PS-SIMD** we have (see [2], [3]):

$$Graphs(\textbf{PS-SIMD}) = \{G_s, G_e, G_{Id}\},$$

and the conditions for pairs are:

1. (i) if $graph = G_{Id}$, then its rules are rewriting rules;
2. (ii) if $graph \in \{G_s, G_e\}$, i.e., *graph* is either of type *shuffle* or of type *exchange*, then its rules are communication rules.

For X = 2D-mesh SIMD = $\textbf{SIMD-MC}^2$ we have:

$$Graphs(\textbf{SIMD-MC}^2) = \mathcal{P}(G_{total}) \cup \mathcal{P}(Id),$$

and the conditions for pairs are:

1. (i) if $graph \subset Id$, then its rules are rewriting rules;
2. (ii) if $graph \subset G_{total}$, then its rules are communication rules.

As for the differences in the structure of the sequence R_μ, in the case of the perfect shuffle architecture R_μ was periodic, while in the case of the 2D-mesh architecture it is not necessarily so, as we will see in the example illustrated in the next section.

4 The Sum on the 2D-Mesh

We compute the sum of n integer numbers $a_{11}, a_{12}, \ldots, a_{ll}$, where $n = l^2$, using the 2D-Mesh architecture, each integer held in one processor.

We have $n = l^2$ processors P_{ij} $(1 \leq i, j \leq l)$ that possess two local variables: x_{ij} (initialized by a_{ij}) and t_{ij} (initialized by 0).

The following procedure computes the sum $a_{11} + a_{12} + \cdots + a_{ll}$.

```
procedure sum$_{\mathbf{2D-MESH}}(a_{11}, a_{12}, \cdots, a_{ll})$
begin
    for all $i, j$ where $1 \leq i, j \leq l$ do
        $x_{ij} \leftarrow a_{ij}$; $t_{ij} \leftarrow 0$
    endfor
    for $j \leftarrow l-1$ downto 1 do
        for all $i$ where $1 \leq i \leq l$ do
            $t_{ij} \Leftarrow x_{i(j+1)}$
            $x_{ij} \leftarrow x_{ij} + t_{ij}$
        endfor
    endfor
    for $i \leftarrow l-1$ downto 1 do
            $t_{i1} \Leftarrow x_{i+1,1}$
            $x_{i1} \leftarrow x_{i1} + t_{i1}$
    endfor
end
```

where $t_{ij} \Leftarrow x_{i(j+1)}$ means that processor $P_{i,(j+1)}$ links by the mesh with processor P_{ij} and communicates the value of variable x, and processor P_{ij} puts it in variable t.

In order to prove the correctness of this algorithm we reformulate in detail what is happening in each processor throughout the execution. We denote by x_{ij}^r and t_{ij}^r the corresponding values of the local variables of processor P_{ij} after step r of the execution.
The above algorithm can now be reformulated in the following, more detailed, manner:

```
procedure sum$_{\mathbf{2D-MESH}}(a_{11}, a_{12}, \cdots, a_{ll})$
begin
    for all $i, j$ where $1 \leq i, j \leq l$ do
        $x_{ij}^0 \leftarrow a_{ij}$; $t_{ij}^0 \leftarrow 0$
    endfor
    for $j \leftarrow l-1$ downto 1 do
        for all $i$ where $1 \leq i \leq l$ do
            $t_{ij}^{l-j} \Leftarrow x_{i(j+1)}^{l-j-1}$
            $x_{ij}^{l-j} \leftarrow x_{ij}^0 + t_{ij}^{l-j}$
        endfor
    endfor
    for $i \leftarrow l-1$ downto 1 do
            $t_{i1}^{2l-i-1} \Leftarrow x_{i+1,1}^{2l-i-2}$
            $x_{i1}^{2l-i-1} \leftarrow x_{i1}^{l-1} + t_{i1}^{2l-i-1}$
    endfor
end
```

Theorem 1. *The formula* $\theta(j) \equiv \forall i\ (1 \leq i \leq l \longrightarrow x_{ij}^{l-1} = \sum_{s=j}^{l} a_{is})$ *is an invariant of the loop "*`for` $j \leftarrow l-1$ `downto 1 do`*" of the procedure* $\texttt{sum}_{\mathbf{2D-MESH}}$.

Proof. By descendant induction on j.

– For $j = l-1$ we have, for each i such that $1 \leq i \leq l$:

$$\begin{aligned} x_{i,l-1}^{l-1} = x_{i,l-1}^{1} &= x_{i,l-1}^{0} + t_{i,l-1}^{1} \\ &= x_{i,l-1}^{0} + x_{i,l}^{0} \\ &= a_{i,l-1} + a_{i,l} \\ &= \sum_{s=l-1}^{l} a_{is} \end{aligned}$$

– Let $j > 1$ and suppose that the formula $\theta(j)$ is true. Let us prove that the formula $\theta(j-1)$ is also true.
  For each i such that $1 \leq i \leq l$ we have:

$$\begin{aligned} x_{i,j-1}^{l-(j-1)} &= x_{i,j-1}^{0} + t_{i,j-1}^{l-(j-1)} \\ &= x_{i,j-1}^{0} + x_{i,j}^{l-j} \\ &\overset{i.h.}{=} a_{i,j-1} + \sum_{s=j}^{l} a_{is} \\ &= \sum_{s=j-1}^{l} a_{is} \end{aligned}$$

□

Theorem 2. *The formula* $\varphi(i) \equiv x_{i1}^{2l-i-1} = \sum_{r=i}^{l}\sum_{s=j}^{l} a_{rs}$ *is an invariant of the loop "*`for` $i \leftarrow l-1$ `downto 1 do`*" of procedure* $\texttt{sum}_{\mathbf{2D-MESH}}$.

Proof. By descendant induction on i.

– For $i = l-1$ we have:

$$\begin{aligned} x_{i,1}^{2l-i-1} = x_{l-1,1}^{2l-(l-1)-1} &= x_{l-1,1}^{l-1} + t_{l-1,1}^{2l-(l-1)-1} \\ &= x_{l-1,1}^{l-1} + x_{l,l}^{l-1} \\ &= \sum_{s=1}^{l} a_{l-1,s} + \sum_{s=1}^{l} a_{l,s} \\ &= \sum_{r=l-1}^{l} \sum_{s=l-1}^{l} a_{rs} \end{aligned}$$

– Let $i > 1$ and suppose that the formula $\varphi(i)$ is true. Let us prove that the formula $\varphi(i-1)$ is also true.

For each i such that $1 \leq i \leq l$ we have:

$$\begin{aligned} x_{i-1,1}^{2l-(i-1)-1} &= x_{i-1,1}^{l-1} + t_{i-1,1}^{2l-(i-1)-1} \\ &= x_{i-1,1}^{l-1} + x_{i,1}^{2l-i-1} \\ &\overset{Th.1+i.h.}{=} \sum_{s=1}^{l} a_{i-1,1} + \sum_{r=i}^{l}\sum_{s=j}^{l} a_{rs} \\ &= \sum_{r=i-1}^{l}\sum_{s=j}^{l} a_{rs} \end{aligned}$$

□

Corollary 1. *At the end of the execution of procedure* $\mathtt{sum}_{\mathbf{2D-MESH}}$*, the variable x of processor P_{11} contains the value of the sum $a_{11} + \cdots + a_{ll}$.*

Proof. At the end of execution the formula $\varphi(1)$ is true; that is, it is verified that:

$$x_{11}^{2l-1-1} = \sum_{r=1}^{l}\sum_{s=j}^{l} a_{rs} = a_{11} + a_{12} + \cdots + a_{ll}.$$

Recall that x_{11}^{2l-1-1} is the content of the processor P_{11} at the end of the execution.

□

Let us note that the above algorithm, although formulated for a square 2D-mesh, works in the same way on a rectangular mesh, of dimensions $n \times m$, and only slight changes are necessary in the proofs of the corresponding correctness results.

The following procedure computes the sum $a_{11} + a_{12} + \cdots + a_{nm}$.

```
procedure sum$_{\mathbf{2D-MESH}}$($a_{11}, a_{12}, \cdots, a_{nm}$)
begin
    for all $i, j$ where $1 \leq i \leq n$, $1 \leq j \leq m$ do
        $x_{ij} \leftarrow a_{ij}$; $t_{ij} \leftarrow 0$
    endfor
    for $j \leftarrow m-1$ downto 1 do
        for all $i$ where $1 \leq i \leq n$ do
            $t_{ij} \Leftarrow x_{i(j+1)}$
            $x_{ij} \leftarrow x_{ij} + t_{ij}$
        endfor
    endfor
    for $i \leftarrow n-1$ downto 1 do
            $t_{i1} \Leftarrow x_{i+1,1}$
            $x_{i1} \leftarrow x_{i1} + t_{i1}$
    endfor
end
```

Its running time, measured in number of iterative steps, is $(n-1) + (m-1)$.

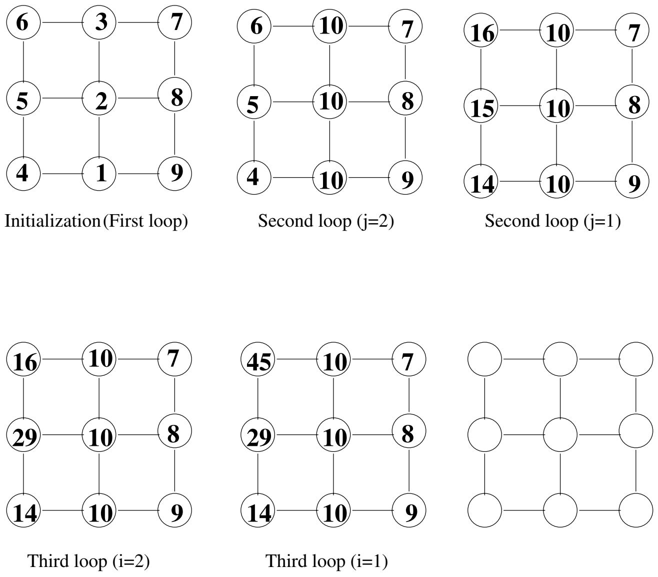

Fig. 1. The sum on a 2D-mesh

Remark: Note that, in the above procedures, the sequence

$$\begin{aligned}&t_{ij} \Leftarrow x_{i(j+1)}\\&x_{ij} \leftarrow x_{ij} + t_{ij}\end{aligned}$$

can be replaced by

$$\begin{aligned}&t_{i(j+1)} \leftarrow x_{i(j+1)}\\&t_{ij} \Leftarrow t_{i(j+1)}\\&x_{ij} \leftarrow x_{ij} + t_{ij}\end{aligned}$$

and the sequence

$$\begin{aligned}&t_{i1} \Leftarrow x_{i+1,1}\\&x_{i1} \leftarrow x_{i1} + t_{i1}\end{aligned}$$

by

$$\begin{aligned}&t_{i+1,1} \leftarrow x_{i+1,1}\\&t_{i1} \Leftarrow t_{i+1,1}\\&x_{i1} \leftarrow x_{i1} + t_{i1}.\end{aligned}$$

The difference is that, instead of communicating directly the value of $x_{i(j+1)}$ to the variable t_{ij} through a mesh connection between P_{ij} and $P_{i(j+1)}$, we first copy in $P_{i(j+1)}$ the value of $x_{i(j+1)}$ into the auxiliary variable $t_{i(j+1)}$, and we use the mesh connections to transmit values of only the auxiliary corresponding variables t. This is an important aspect for the simulations which follow.

In Figure 1 the functioning of the sum algorithm on a 3×3 2D-mesh is illustrated, from the initial configuration (after initialization), through the subsequent ones – each obtained after an iterative step, to the last one (after the fourth iterative step), in which the sum is obtained in processor P_{11}.

5 Simulation of the Sum

We present a simulation of the sum algorithm on 2D-mesh using P systems with dynamic communication graphs of the 2D-mesh type. Actually, as we will see, we will have three possible simulations, depending on the input data, and the requirements we impose on the "memory" of membranes P_{ij}, with $ij \neq 11$, which can be considered auxiliary in the process of summation.

As already stated in Section 3, each processor will be simulated by a membrane, and we keep the same labels for membranes as for processors.

Thus, each membrane P_{ij}, for $1 \leq i \leq n$, $1 \leq j \leq m$, will have to have distinct representations for its two internal variables, x_{ij} and t_{ij} (when both are necessary). Let us denote by $|x|$ the positive part of any integer x. We codify the integer values of the local variables x_{ij} and t_{ij}, in the following manner:

– Let a and $\bar{a}$ be the symbols to codify the integer content of the x_{ij} variable of every membrane P_{ij}. Symbol a is used to represent the positive units, and $\bar{a}$ for the negative ones. Suppose x_{ij} has integer value a_{ij}: if a_{ij} is positive, it will be codified as $a^{a_{ij}}$, if a_{ij} is negative, it will be codified as $\bar{a}^{|a_{ij}|}$.

– Let b and $\bar{b}$ be the symbols to codify the integer content of the t_{ij} variable of every membrane P_{ij}, b for positive units, and $\bar{b}$ for negative ones. They are used in the same fashion as a and $\bar{a}$ are used to codify x_{ij}, described above.

If the data are arbitrary integers, the contents of the processors will be codified with symbol objects over the alphabet $V = \{a, \bar{a}, b, \bar{b}\}$. If the data are always positive integers, the contents of the processors can be codified with symbol objects over the smaller alphabet $V = \{a, b\}$.

Let us illustrate how addition of two integers can be performed using two (adjacent) membranes: consider P_x and P_y two membranes, each containing an integer, x, and respectively y, codified as a^z, for $z \geq 0$ and/or as $\bar{a}^z$ for $z < 0$, $z \in \{x, y\}$. Suppose both integers are positive, thus the initial configuration is $P_x : a^x$, and $P_y : a^y$.

Suppose moreover that we are interested in obtaining the result of the addition in P_x, and that we are not interested if we loose the original initial value of P_y. Then, using an active edge (P_y, P_x) together with the symport rule (a, out), i.e., the pair $[(P_y, P_x), (a, out)]$ accomplishes our desired objective: precisely y occurrences of a travel "along" the edge, and we will have $x + y$ occurrences of a in P_x.

If our integers are arbitrary, i.e., we can have any of the four combinations of initial configurations, P_x : either a^x or $\bar{a}^x$, P_y : either a^y or $\bar{a}^y$, if we are interested in obtaining the result of the addition in P_x, and we are not interested if we loose the original initial value of P_y, then, we can use an active edge (P_y, P_x) together with the symport rules $\{(a, out), (\bar{a}, out)\}$, i.e., the pair $[(P_y, P_x), \{(a, out), (\bar{a}, out)\}]$, followed by $[Id_{P_x}, (a\bar{a} \longrightarrow \lambda)]$. The last internal rewriting step taking place in P_x ensures us that the addition is correctly performed, positive and negative units annihilate themselves in pairs.

If we do wish to keep however also the contents of P_y, additional rewriting steps are necessary, and the use of symbols $\{b, \bar{b}\}$ becomes apparent. The sequence

$$[Id_{P_y}, \{a \longrightarrow ab, \bar{a} \longrightarrow \bar{a}\bar{b}\}], [(P_y, P_x), \{(b, out), (\bar{b}, out)\}],$$
$$[Id, \{b \longrightarrow a, \bar{b} \longrightarrow \bar{a}, a\bar{a} \longrightarrow \lambda\}],$$

accomplishes the task of having $x + y$ represented in P_x and y still in P_y: by the first set, the a's, respectively $\bar{a}$'s, in P_y are duplicated also as b's, respectively $\bar{b}$'s; by the second set of rules all the b's, respectively $\bar{b}$'s, get from P_y into P_x; by the third set, the b's, respectively $\bar{b}$'s in P_x get rewritten as a's, respectively $\bar{a}$'s, and the rule $a\bar{a} \longrightarrow \lambda$ ensures the proper addition of positive and negative units, if it is necessary. The result of adding $x + y$ is represented as a^{x+y} or $\bar{a}^{|x+y|}$ in P_x; in P_y we have the same content as before, since all b's, or $\bar{b}$'s have travelled by the second rule into P_x, and thus none of the rules of the third set are applicable in P_y.

Let us go back now to our system of membranes P_{ij}, for $1 \leq i \leq n$, $1 \leq j \leq m$, seen as the nodes of a (virtual) 2D-lattice of dimension $n \times m$.

Suppose, for the sake of simplicity, that we deal only with positive integers, and that we are only interested in obtaining the result of the addition $a_{11} + \cdots + a_{nm}$ in membrane P_{11}, loosing the initial and intermediate values in all the other membranes.

For our system of membranes and the problem stated above we will have:

– *The initial configuration:* for $1 \leq i \leq n$, $1 \leq j \leq m$, every membrane P_{ij} contains $a^{a_{ij}}$, and no b's.
– *The sequence of horizontal subgraphs:* For the first $m - 1$ iterative steps, we will consider, at each step $j = m-1, \cdots, 1$, the subgraphs $\{((i, j+1), (i, j)) \mid 1 \leq i \leq n\}$ composed of all horizontal edges connecting column $j + 1$ to column j, and we consider them oriented edges. Along each such edge, we have to perform the addition of variables $x_{i(j+1)}$ and x_{ij}, and put the result in x_{ij}. If we are not interested in preserving the values (initial or intermediate) of every processor (with the exception of processor P_{11} which collects the final result), we can simply use the rule (a, out) along each such (oriented) edge. Along each edge $((i, j + 1), (i, j))$, $x_{i(j+1)}$ occurrences of symbol a will travel into membrane P_{ij} where they will increase the number of a's, so that it will be precisely $x_{i(j+1)} + x_{ij}$ (and the codification for variable $x_{i(j+1)}$ in processor $P_{i(j+1)}$ is lost).
– *The sequence of vertical subgraphs:* For the last $n - 1$ iterative steps, we will consider, at each step $i = n - 1, \cdots, 1$, the subgraph $(((i + 1), 1), (i, 1))$ composed of an oriented vertical edge connecting on column 1 line $i + 1$ to line i. Along each such edge, we have to perform again the addition of variables $x_{(i+1)1}$ and x_{i1}, and put the result in x_{i1}. Again, we can use the rule (a, out) along each such (oriented) edge.

Note that we have used an alphabet of only one symbol, $\{a\}$, and that there are no internal computation steps in the form of rewriting taking place in any subset of membranes.

Theorem 3. *Consider the P system with dynamic communication of 2D-mesh type*

$$\Pi_1 = (\{a\}, \{P_{ij} \mid 1 \leq i \leq n, 1 \leq j \leq m\}, R_{\mu 1}),$$

where $R_{\mu 1}$ is the following sequence of pairs $[graph, rules]$:

$$R_{\mu 1} = \{[\{((i, j+1), (i, j)) \mid 1 \leq i \leq n\}, (a, out)],\ j = m-1, \cdots, 1,$$

$$[(((i+1), 1), (i, 1)), (a, out)],\ i = n-1, \cdots, 1\}.$$

Then, starting from the initial configuration $\{P_{ij} : a^{a_{ij}} \mid 1 \leq i \leq n, 1 \leq j \leq m\}$, where all a_{ij} are positive integers, the application of the sequence $R_{\mu 1}$ will lead to the final configuration $P_{11} : a^S$, where $S = a_{11} + \cdots + a_{nm}$, and $P_{ij} : \emptyset$ for all $ij \neq 11$.

Let us consider now the case in which we have to sum arbitrary integers, and suppose that we do not wish to keep the initial or intermediate contents of any other membrane but P_{11}. We will have:

- *The initial configuration:* for $1 \leq i \leq n$, $1 \leq j \leq m$, for $a_{ij} \geq 0$ every membrane P_{ij} contains $a^{a_{ij}}$, and for $a_{ij} < 0$ every membrane P_{ij} contains $\bar{a}^{|a_{ij}|}$.
- *The sequence of horizontal subgraphs – communication followed by internal computations:* For the first $m-1$ iterative steps, we will consider, at each step $j = m-1, \cdots, 1$, the subgraphs $\{((i, j+1), (i, j)) \mid 1 \leq i \leq n\}$ composed of all horizontal edges connecting column $j+1$ to column j, and we consider them oriented edges. To each such edge we associate the symport rules $\{(a, out), (\bar{a}, out)\}$ which ensure that either positive or negative units, depending on the case, travel all into membrane P_{ij}. (The codification for the value $a_{i(j+1)}$ in membrane $P_{i(j+1)}$ is lost). In order to ensure proper addition of integers with different signs on column j, we will use an internal computation step modelled as $[Id_{*j}, a\bar{a} \longrightarrow \lambda]$, where Id_{*j} is the subgraph of the identity graph associated to column j.
  The sequence $[graph, rules]$ which models this stage will thus be

  $$R_h{}^j = [\{((i, j+1), (i, j)) \mid 1 \leq i \leq n\}, \{(a, out), (\bar{a}, out)\}], [Id_{*j}, a\bar{a} \longrightarrow \lambda],$$

  for $j = m-1, \cdots, 1$.
- *The sequence of vertical subgraphs – communication followed by internal computations:* For the last $n-1$ iterative steps, we consider, at each step $i = n-1, \cdots, 1$, the subgraph $((i+1, 1), (i, 1))$ composed of an oriented vertical edge connecting on column 1 line $i+1$ to line i. Along each such edge, we use the rules $\{(a, out), (\bar{a}, out)\}$. After such a communication step, we have an internal computation step, which we model with $[Id_{i*}, a\bar{a} \longrightarrow \lambda]$, where Id_{i*} is the subgraph of the identity graph associated to line i. (We could have used only the subgraph Id_{i1}.)
  The sequence $[graph, rules]$ which models this stage will thus be

  $$R_v{}^i = [\{((i+1, 1), (i, 1))\}, \{(a, out), (\bar{a}, out)\}], [Id_{i*}, a\bar{a} \longrightarrow \lambda],$$

  for $i = n-1, \cdots, 1$.

Theorem 4. *Consider the P system with dynamic communication of 2D-mesh type*

$$\Pi_2 = (\{a, \bar{a}\}, \{P_{ij} \mid 1 \leq i \leq n, 1 \leq j \leq m\}, R_{\mu 2}),$$

where $R_{\mu 2}$ is the following sequence of pairs $[graph, rules]$:

$$R_{\mu 2} = {R_h}^{m-1} \cdots {R_h}^{j} \cdots {R_h}^{1} \cdot {R_v}^{n-1} \cdots {R_v}^{i} \cdots {R_v}^{1}.$$

Then, starting from the initial configuration $\{P_{ij} : a^{a_{ij}} \mid 1 \leq i \leq n, 1 \leq j \leq m, a_{ij} \geq 0\}$, and $\{P_{ij} : \bar{a}^{|a_{ij}|} \mid 1 \leq i \leq n, 1 \leq j \leq m, a_{ij} < 0\}$, where all a_{ij} are arbitrary integers, the application of the sequence $R_{\mu 2}$ will lead to the final configuration $P_{11} : a^S$, where $S = a_{11} + \cdots + a_{nm}$, and $P_{ij} : \emptyset$ for all $ij \neq 11$.

Consider now the most general case: we have to sum arbitrary integers, and we do not wish to destroy the contents of any membrane when it links by mesh to another membrane and transmits the value of a variable. We will use for transmission only the auxiliary variables t_{ij}, codified over the set of symbols $\{b, \bar{b}\}$. Some more internal processing in the form of rewriting will be necessary, both before, and after the communication step. (Note that in the previous versions, we did not have to simulate in our membranes the auxiliary variables t.) The simulation which follows resembles the modified version of the algorithm as in the Remark of Section 4.

We will have:

– *The initial configuration:* for $1 \leq i \leq n$, $1 \leq j \leq m$, for $a_{ij} \geq 0$ every membrane P_{ij} contains $a^{a_{ij}}$, and for $a_{ij} < 0$ every membrane P_{ij} contains $\bar{a}^{|a_{ij}|}$. There are neither b's, nor $\bar{b}$'s (the t variables are all initialized to 0).

– *The sequence of horizontal subgraphs – communication and internal computations:*
Consider the sequence $[graph, rules]$:

$${R_h}^{j} = [Id_{*(j+1)}, \{a \longrightarrow ab, \bar{a} \longrightarrow \bar{a}\bar{b}\}],$$

$$[\{((i, j+1), (i, j)) \mid 1 \leq i \leq n\}, \{(b, out), (\bar{b}, out)\}],$$

$$[Id_{*j}, \{b \longrightarrow a, \bar{b} \longrightarrow \bar{a}, a\bar{a} \longrightarrow \lambda\}],$$

for $j = m-1, \cdots, 1$. Each sequence ${R_h}^{j}$ simulates the j iterative step of the first part of the algorithm. By the first set, in column $j+1$ simultaneous rewriting take place, simulating copying the contents of variable $x_{i(j+1)}$ into $t_{i(j+1)}$. By the second set, the b's (respectively $\bar{b}$'s) travel along the active horizontal edges of the mesh, from column $j+1$ to column j on each line i. By the third set, another internal computation takes place, in column j, simulating the addition of the auxiliary variable t_{ij} to a_{ij} and putting the result in a_{ij}. (Again, the t_{ij}'s will be zero.)
The sequence obtained by catenation

$$R_h = {R_h}^{m-1} \cdots {R_h}^{j} \cdots {R_h}^{1}$$

simulates the sequential execution of the first $m-1$ iterative steps of the sum algorithm.

– *The sequence of vertical subgraphs – communication and internal computations:*
Consider for each $i = n-1, \cdots, 1$, the sequence $[graph, rules]$:

$$R_v{}^i = [Id_{(i+1)1}, \{a \longrightarrow ab, \bar{a} \longrightarrow \bar{a}\bar{b}\}],$$

$$[\{((i+1,1),(i,1))\}, \{(b, out), (\bar{b}, out)\}],$$

$$[Id_{i*}, \{b \longrightarrow a, \bar{b} \longrightarrow \bar{a}, a\bar{a} \longrightarrow \lambda\}].$$

It functions in a similar way to the sequences associated to the horizontal subgraph. By catenation we obtain the sequence

$$R_v = R_v{}^{n-1} \cdots R_v{}^i \cdots R_v{}^1$$

which simulates the last $n-1$ iterative steps of the algorithm.

Theorem 5. *Consider the set $\{a_{ij} \mid 1 \leq i \leq n, 1 \leq j \leq m\}$ of arbitrary integers, and consider the P system with dynamic communication of 2D-mesh type*

$$\Pi_3 = (\{a, \bar{a}, b, \bar{b}\}, \{P_{ij} \mid 1 \leq i \leq n, 1 \leq j \leq m\}, R_{\mu 3}),$$

where $R_{\mu 3}$ is the following sequence of pairs $[graph, rules]$:

$$R_{\mu 3} = R_h \cdot R_v = R_h{}^{m-1} \cdots R_h{}^j \cdots R_h{}^1 \cdot R_v{}^{n-1} \cdots R_v{}^i \cdots R_v{}^1,$$

with $R_h{}^j$ and $R_v{}^i$ as defined previously.

Then, starting from the initial configuration $\{P_{ij} : a^{a_{ij}} \mid 1 \leq i \leq n, 1 \leq j \leq m, a_{ij} \geq 0\}$, and $\{P_{ij} : \bar{a}^{|a_{ij}|} \mid 1 \leq i \leq n, 1 \leq j \leq m, a_{ij} < 0\}$, the application of the sequence $R_{\mu 3}$ will lead to a final configuration in which $P_{11} : a^S$, where $S = a_{11} + \cdots + a_{nm}$, and $P_{ij} :\neq \emptyset$ for all $ij \neq 11$ (with the possible exception of those containing integer 0).

We end this section with some remarks on the three versions of simulations we have proposed. We believe that discussing them, versus the parallel algorithm implemented on the 2D-mesh in Section 4, illustrates similarities and differences between "communication" and "internal computations" as understood, on one hand in parallel architectures, and on the other hand, in their simulation with P systems.

First, let us note that the third version, in Theorem 5, can be considered a complete simulation of the algorithm in Section 4, in the sense that each membrane P_{ij}, even with $ij \neq 11$, will have, in a codified form, the same content as processor P_{ij}.

If we concentrate only on the result, i.e., on membrane P_{11} which must finally contain the sum, we note that, in the simulation with P systems, we can almost obtain it with only "communication" rules: rewriting is necessary only in Theorem 4, and only because the input data may contain integers with different signs. Moreover, both "communication" and "internal computations" steps in the parallel architecture, were modelled with "communication-only" rules in the P systems simulation (the case of Theorem 3). In the first two simulations, Theorem 3 and Theorem 4, there was no need to even represent the auxiliary variable t, a local memory essential for the functioning of the parallel algorithm. This implies that the notion of "communication" in P systems is stronger than the corresponding notion in classical parallel models, of course, at the cost of "local memory loss".

6 The Reduction on the 2D-Mesh Architecture

Let $(A, *, 0)$ be a commutative monoid, i.e., A is a set, $*$ is a binary associative and commutative operation over A, and 0 is the neutral element of $*$.

Let $a_0, \ldots, a_{k-1}$ be elements of this set. The *reduction* (see [8], Section 2.3.1) is the process of computing $a_0 * \cdots * a_{k-1}$.

The 2D-mesh architecture can be used to solve the reduction problem. Let us suppose that the total number of elements is $k = (2m+1) \times (2n+1)$. Consider $k = (2m+1) \times (2n+1)$ processors connected by a 2D-mesh network. The nodes of the network will be labelled by pairs (i, j) (or simply ij), with i the row index, $-m \leq i \leq m$, and with j the column index, $-n \leq j \leq n$. The set of elements to which we want to apply the reduction is

$$\{a_{(-m)(-n)}, \ldots, a_{(-m)(n)}, \ldots, a_{00}, \ldots, a_{m(-n)}, \ldots, a_{mn}\}.$$

Processor P_{ij}, with label ij, $-m \leq i \leq m$, $-n \leq j \leq n$, possesses two local variables: x_{ij} (initialized by a_{ij}) and t_{ij} (initialized by 0).

The following procedure computes

$$a_{(-m)(-n)} * \cdots * a_{(-m)(n)} * \cdots * a_{00} * \cdots * a_{m(-n)} * \cdots * a_{mn},$$

and puts the result in processor P_{00}:

```
procedure reduction$_{\mathbf{2D-MESH}}$($a_{(-m)(-n)}, \ldots, a_{00}, \ldots, a_{mn}$)
begin
    for all $i, j$ where $-m \leq i \leq m$, $-n \leq j \leq n$, do
        $x_{ij} \leftarrow a_{ij}$; $t_{ij} \leftarrow 0$
    endfor
    for $j \leftarrow n-1$ downto 0 do
        for all $i$ where $-m \leq i \leq m$ do
            $t_{ij} \Leftarrow x_{i(j+1)}$
            $x_{ij} \leftarrow x_{ij} * t_{ij}$
        endfor
    endfor
    for $j \leftarrow -n+1$ to 0 do
        for all $i$ where $-m \leq i \leq m$ do
            $t_{ij} \Leftarrow x_{i(j-1)}$
            $x_{ij} \leftarrow x_{ij} * t_{ij}$
        endfor
    endfor
    for $i \leftarrow m-1$ downto 0 do
        $t_{i0} \Leftarrow x_{i+1,0}$
        $x_{i0} \leftarrow x_{i0} * t_{i0}$
    endfor
    for $i \leftarrow -m+1$ to 0 do
        $t_{i0} \Leftarrow x_{i-1,0}$
        $x_{i0} \leftarrow x_{i0} * t_{i0}$
    endfor
end
```

The second and fourth for loops are iterative steps, one responsible for performing the $*$ operation on columns from n to 0, the other for performing the $*$ operation on rows from m to 0, similarly to the algorithm for sum in section 4.

However, this algorithm has some improvement over the one in section 4. The third for loop is the "mirror image" of the second one: a sequence of iterative steps, performing the $*$ operation on columns, this time from $-n$ to 0. Note that the execution of iterative step j in the second loop can be done in parallel with the execution of iterative step $-j$ of the third loop. Similarly, the fifth for loop is the "mirror image" of the fourth one: a sequence of iterative steps, performing the $*$ operation on rows, this time from $-m$ to 0. Note that the execution of iterative step i in the fourth loop can be done in parallel with the execution of iterative step $-i$ of the fifth loop.

This ensures that the running time of this algorithm, (if we apply the parallelism mentioned above), measured in iterative steps is $(2m+2n)/2$, i.e., this algorithm takes $\sqrt{k}/2$ steps, compared with $\sqrt{k}$ steps required by the previous algorithm for sum. The simulation which follows makes use of this enhanced parallelism.

7 Simulation of the Reduction

In order to simulate the algorithm for reduction presented in section 6 with P systems with dynamic communication of 2D-mesh type, we have to work under the following supplementary assumptions:

1. The elements a_{ij} on which the operation $*$ is performed can be codified inside each membrane P_{ij} over a finite alphabet, say V_x; the alphabet will be used to codify all values of variable x_{ij}, both initial and intermediate;
2. Using the same codification, the values of local variables t_{ij} are codified inside each membrane P_{ij} over a finite alphabet, say V_t;
3. V_x and V_t are in bijective correspondence: to each $a \in V_x$ there corresponds a symbol $a' \in V_t$;
4. There exists a set of symbol rewriting rules on $V_x \cup V_t$, denoted r_*, which simulates performing the operation $x_{ij} * t_{ij}$ inside each membrane P_{ij}; the value of the result $x_{ij} * t_{ij}$ is codified over V_x as the new value of x_{ij}.

Under these assumptions, the rest of the model follows the general lines of the models for sum in Section 5. Copying the value of x_{ij} into t_{ij} is done by the set of symbol rewriting rules $\{a \longrightarrow aa' \mid a \in V_x\}$. Since the codifications for x's and t's are the same, the effect is the desired one. Transfer of values for t's over the appropriate edges can be accomplished with the set of symport rules $\{(a', out) \mid a' \in V_t\}$. Finally, the set r_* will compute $x_{ij} * t_{ij}$ and put the result in x_{ij}.

We construct now the sequences of pairs $[graph, rules]$ which simulate the iterative steps of the reduction algorithm. To simplify the notation we write all i instead of $-m \leq i \leq m$.

For each $j = n, \ldots, 1$ (the horizontal steps), we take

$$R_h{}^j = [Id_{*j} \cup Id_{*(-j)}, \{a \longrightarrow aa' \mid a \in V_x\}],$$
$$[\{((i,j),(i,j-1)) \mid \text{all } i\} \cup \{((i,-j),(i,-j+1)) \mid \text{all } i\}, \{(a', out) \mid a' \in V_t\}],$$
$$[Id_{*(j-1)} \cup Id_{*(-j+1)}, r_*].$$

For each $i = m, \ldots, 1$ (the vertical steps on column 0), we take

$$R_v{}^i = [Id_{i0} \cup Id_{(-i)0}, \{a \longrightarrow aa' \mid a \in V_x\}],$$
$$[\{((i,0),(i-1,0))\} \cup \{((-i,0),(-i+1,0))\}, \{(a', out) \mid a' \in V_t\}],$$
$$[Id_{i0} \cup Id_{(-i)0}, r_*].$$

Let $(A, *, 0)$ be a commutative monoid, and consider the following set of elements of A:

$$\{a_{(-m)(-n)}, \ldots, a_{(-m)(n)}, \ldots, a_{00}, \ldots, a_{m(-n)}, \ldots, a_{mn}\}.$$

Assume that every element of A can be codified over an alphabet V_x, and that there exists a set of rewriting rules r_* which simulates performing operation $*$ on two elements of A, one codified over V_x, the other over V_t, and the result of the computation is again codified over V_x. Consider the P system with dynamic communication of 2D-mesh type

$$\Pi_* = (V_x \cup V_t, \{P_{ij} \mid -m \leq i \leq m, -n \leq j \leq n\}, R_{\mu *}),$$

where $R_{\mu *}$ is the sequence of pairs $[graph, rules]$:

$$R_{\mu *} = R_h{}^n \cdots R_h{}^j \cdots R_h{}^1 \cdot R_v{}^m \cdots R_v{}^i \cdots R_v{}^1,$$

with $R_h{}^j$ and $R_v{}^i$ as defined previously.

Then, starting from an initial configuration in which each membrane P_{ij} contains a codification of the value a_{ij} over V_x, for all $-m \leq i \leq m$, $-n \leq j \leq n$, the application of the sequence $R_{\mu *}$ will lead to a final configuration in which P_{00} contains a codification of $S \in A$, where

$$S = a_{(-m)(-n)} * \cdots * a_{(-m)(n)} * \cdots * a_{00} * \cdots * a_{m(-n)} * \cdots * a_{mn}.$$

8 Conclusions

We have analyzed in this paper the possibility of simulating the parallel architecture known as the 2D-mesh with a new version of P systems, P systems with dynamic communication graphs.

In Section 3 a comparison is made between dynamic communication graphs of 2D-mesh type, introduced here, and dynamic communication graphs of shuffle–exchange type, introduced in previous work. We believe this illustrates the power of this new version of P systems as tools for formalizing other network architectures as well.

We have illustrated the proposed simulation with the particular algorithm for computing the sum of a given set of integers. Discussing several possible simulations of the sum algorithm has given us the opportunity to compare "communication" as understood in P systems, and communication in classical parallel architectures.

We have further presented an algorithm to solve the reduction problem implemented on a 2D-mesh, and discussed its simulation with P systems with dynamic communication graphs, simulation possible under some supplementary assumptions.

References

1. R. Ceterchi, C. Martín–Vide, P Systems with Communication for Static Sorting. In M. Cavaliere, C. Martín–Vide and Gh. Păun (eds.), *Proceedings of the Brainstorming Week on Membrane Computing*, Report GRLMC 26/03, 2003, 101–117.
2. R. Ceterchi, M.J. Pérez–Jiménez, Simulating Shuffle–Exchange Networks with P Systems. In Gh. Păun, A. Riscos–Núñez, F. Sancho–Caparrini and A. Romero–Jiménez (eds.), *Proceedings of the Second Brainstorming Week on Membrane Computing*, Report RGNC 01/04, 2004, 117–129.
3. R. Ceterchi, M.J. Pérez–Jiménez, A Perfect Shuffle Algorithm for Reduction Processes and its Simulation with P Systems. In I. Dzitac, T. Maghiar, C. Popescu (eds.), *Proceedings of the International Conference on Computers and Communications ICCC 2004, May 27-29, 2004, Baile Felix Spa – Oradea, Romania*, Editura Univ. Oradea, 2004, 92–97.
4. M.J. Flynn, Very High-Speed Computing Systems, *Proceedings of the IEEE* 54, 12 (1966), 1901–1909.
5. A. Păun, Gh. Păun, The Power of Communication: P Systems with Symport/Antiport. *New Generation Computers*, 20, 3 (2002), 295–306.
6. Gh. Păun, Computing with Membranes. *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143, and *Turku Center for CS-TUCS Report* No. 208, 1998.
7. Gh. Păun, *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, 2002.
8. M.J. Quinn, *Parallel Computing. Theory and Practice*, McGraw–Hill Series in Computer Science, 1994.
9. J.D. Ullman, *Computational Aspects of VLSI*, Computer Science Press, Rockville, MD, 1984.