Simulating the Bitonic Sort on a 2D-mesh with P Systems

Rodica Ceterchi[1], Mario J. Pérez-Jiménez[2], Alexandru Ioan Tomescu[1]

[1] Faculty of Mathematics and Computer Science, University of Bucharest
Academiei 14, RO-010014, Bucharest, Romania
[2] Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
rceterchi@gmail.com, mario.perez@cs.us.es, alexandru.tomescu@gmail.com

Summary. This paper gives a version of the parallel bitonic sorting algorithm of Batcher, which can sort N elements in time $O(\log^2 N)$. When applying it to the 2D mesh architecture, two indexing functions are considered, row-major and shuffled row-major. Some properties are proved for the later, together with a correctness proof of the proposed algorithm. Two simulations with P systems are proposed and discussed. The first one uses dynamic communication graphs and follows the guidelines of the mesh version of the algorithm. The second simulation requires only symbol rewriting rules in one membrane.

1 Introduction

P systems, introduced in [19], are powerful computational models, with non-deterministic as well as parallel features. Deterministic P systems can be also considered, and the power of their parallel features compared against the power of other computational models which enjoy parallelism. Along this line we refer to previous work, which relates P systems with parallel networks of processors, functioning according to the SIMD paradigm (Single Instruction Multiple Data machines), in [8], [9], for shuffle-exchange networks, and in [6] for 2D mesh networks. The comparison was approached by designing P systems which simulate the functioning of a specific architecture, when solving a specific problem. In [7] the general features of this type of approach were abstracted, giving a "blueprint" for the design of a class of deterministic P sytems, with *dynamic communication graphs*, which simulate a given parallel architecture, functioning to implement a given algorithm.

Among the choices to be made for the problem to solve, the *static sorting* imposes itself, being a central theme in computer science. Although it is well known that comparison-based sorting algorithms require at least $O(N \log N)$ comparisons to sort N items, performing many comparisons in parallel can reduce the sorting time. This paper analyses the bitonic sorting algorithm, one of the fastest parallel sorting algorithms where the sequence of comparisons is not data-dependent. The bitonic sorting network was discovered by Batcher [3], who also discovered the network for odd-even sort. These were the first networks capable of sorting N elements in time $O(\log^2 N)$. Stone [24] maps the bitonic sort onto a perfect-shuffle interconnection network, sorting N elements by using N processors in time $O(\log^2 N)$. Siegel [23] shows that bitonic sort can also be performed on the hypercube in time $O(\log^2 N)$. The shuffled row-major indexing formulation of bitonic sort on a mesh-connected computer is presented by Thompson and Kung [25]. They also show how the odd-even merge sort can be used with snakelike row-major indexing. Nassimi and Sahni [16] present a row-major indexed bitonic sort formulation for a mesh with the same performance as shuffled row-major indexing.

Static sorting algorithms have been developped and proposed also in the P systems area. Among the first approaches, made independently, we mention [2] and [4], [5]. The problem of sorting with P systems occupies Chapter 8, [1], of the monograph [10].

We analyze in this paper a version of the bitonic sorting algorithm of Batcher, and its implementation on the 2D mesh architecture. Section 2 introduces the mesh topology and the model of computation. In 2.2 we begin the formal study of indexing functions, and we stress their importance for the passing to a network architecture. (Some similar work has been done in [12] and [13], but we develop our own formalism.) In 2.3 we present the algorithm, and the main result, Theorem 1, whose Corrolary is the correctness proof of the algorithm. Other results in this subsection, like Lemma 3, and the Remarks, are subsequently used to prove assertions about the algorithm, and, in Section 3, about the simulations with P systems.

Section 3 is devoted to the presentation of two different simulations of the algorithm with P systems. The first simulation uses dynamic communication graphs, as in [7]. A generative approach to the sequence of graphs used to communicate values between the membranes is a novel feature. The second simulation, uses only one membrane, and symbol rewriting rules.

2 Preliminaries: The bitonic sort on the 2D-mesh

2.1 Model of Computation

The presentation of the bitonic sort on the 2D-mesh architecture is made here based mainly on the paper [25]. It is the same algorithm as in [21], but with more emphasis on the routings necessary to compare elements situated at greater distances on the mesh. Also, some restrictions imposed in [25], will be eliminated, or re-examined, since they were dictated by their explicit connection to the ILLIAC IV-type parallel computer. In general, our references to parallel machines/architectures will be at the level of generalization to be found for instance in [21]. We also found it useful to formalize properly some aspects related to the indexing function.

Let us assume as in [25] that we have a parallel computer with $N = n \times n$ identical processors, disposed in a 2D-mesh structure. A processor is connected to all of its four vertical or horizontal neighbors, except for the processors situated on the perimeter, which have at most two or three neighbors, as no "wrap-around connections" are permitted.

Another assumption is that it is a SIMD (Single Instruction Multiple Data) machine. During each time unit, a single instruction is executed by a set of processors. In what follows, only two processor registers and two instructions are needed. For inter-processor data moves, we will use a routing instruction which copies the value of a register to a register of a neighbor processor. The second instruction is the internal comparison between the values of the two registers of a processor.

We define t_R the time for one-unit distance routing step, and t_C the time required for one comparison step. Concurrent data movement is allowed, as long as it is in the same direction; also any number of parallel comparisons can be made simultaneously.

2.2 The sorting problem and indexing functions

We assume to have an indexing function on the processors that is a one-to-one mapping from $\{0, 1, \ldots, n-1\} \times \{0, 1, \ldots, n-1\}$ onto $\{0, 1, \ldots, N-1\}$ and that initially N integers are loaded in the N processors. Therefore the sorting problem is defined as moving the jth smallest element to the processor indexed by j, for all $j \in \{0, 1, \ldots N-1\}$.

Let $I : \{0, 1, \ldots, n-1\} \times \{0, 1, \ldots, n-1\} \longrightarrow \{0, 1, \ldots, N-1\}$ denote an indexing function. Two indexing schemes are the following:

(i) *Row-major indexing.* This is illustrated in Figure 1, and we denote it by $I = RM$.
(ii) *Shuffled row-major indexing.* This is illustrated in Figure 2, and we denote it by $I = sRM$.

In order not to make the notation cumbersome, we let the same letter, say i, stand for an integer in $\{0, 1, \ldots, n-1\}$, and for its binary representation as a string. For $n = 2^k$, as the case will be, i will be a binary string of length k. Whenever necessary, we complete with zeroes (obviously, to the left) to obtain strings of the same length. (When we refer to bits of such a string, we count from 1 to n, starting from right to left, such that the "first" bit will be that of the least significant digit, and so forth. However, when we write such a string, we will write it with bits numbered from right to left.)

Consider the following definitions:

Definition 1. *The row-major indexing function RM is defined by $RM(i, j) = ij$, where in the right hand-side we have denoted by ij the string concatenation of the*

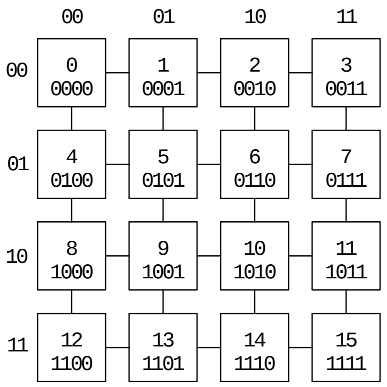

Fig. 1. Row-major indexing scheme for a 4×4 mesh

binary representations of the integers i and j, after they have been brought to the same length.

More precisely, for every k, we have the bijections

$$RM_k : \{0, 1, \ldots, 2^k - 1\} \times \{0, 1, \ldots, 2^k - 1\} \longrightarrow \{0, 1, \ldots, 2^{2k} - 1\},$$

defined by

$$RM_k(i_1 i_2 \cdots i_k, j_1 j_2 \cdots j_k) = i_1 i_2 \cdots i_k j_1 j_2 \cdots j_k.$$

Let sh (from *'shuffle'*) stand generically for the family of bit-shuffle functions $sh_k : \{0, 1, \ldots, 2^{2k} - 1\} \longrightarrow \{0, 1, \ldots, 2^{2k} - 1\}$, defined by

$$sh_k(i_1 i_2 \cdots i_k j_1 j_2 \cdots j_k) = i_1 j_1 i_2 j_2 \cdots i_k j_k.$$

This is a bijection, with the obvious inverse

$$ush_k(i_1 j_1 i_2 j_2 \cdots i_k j_k) = i_1 i_2 \cdots i_k j_1 j_2 \cdots j_k,$$

where ush comes from *'un-shuffle'*. In the following we will drop the index k whenever it is clear from the context.

Definition 2. *The shuffled row-major indexing function sRM is defined by $sRM(i, j) = sh(ij)$, where in the right hand-side we have denoted by ij the string concatenation of the binary representations of the integers i and j, after they have been brought to the same length, and sh is the appropriate bit-shuffle function.*

More precisely, for every k, we have the bijections

$$sRM_k : \{0, 1, \ldots, 2^k - 1\} \times \{0, 1, \ldots, 2^k - 1\} \longrightarrow \{0, 1, \ldots, 2^{2k} - 1\},$$

defined by

$$sRM_k(i_1 i_2 \cdots i_k, j_1 j_2 \cdots j_k) = sh_k(i_1 i_2 \cdots i_k j_1 j_2 \cdots j_k) =$$

$$= i_1 j_1 i_2 j_2 \cdots i_k j_k.$$

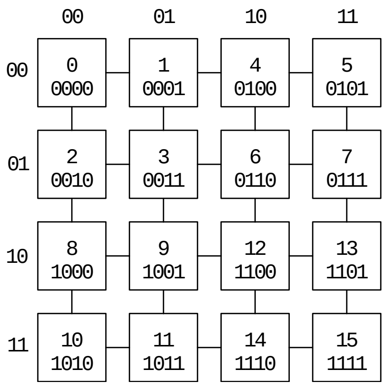

Fig. 2. Shuffled row-major indexing scheme for a 4×4 mesh

Lemma 1. *For every* $i, j_1, j_2 \in \{0, 1, \ldots, 2^k - 1\}$ *with* $j_1 < j_2$*, we have that* $sRM(i, j_1) < sRM(i, j_2)$*. Analogously, for every* $i_1, i_2, j \in \{0, 1, \ldots, 2^k - 1\}$ *with* $i_1 < i_2$*, we have that* $sRM(i_1, j) < sRM(i_2, j)$*.*

Proof. The proof is immediate, from the definition of sRM. □

Let us consider the following general definition:

Definition 3. *We call indexing function on the* $2^k \times 2^k$ *mesh a bijection*

$$I_k : \{0, 1, \ldots, 2^k - 1\} \times \{0, 1, \ldots, 2^k - 1\} \longrightarrow \{0, 1, \ldots, 2^{2k} - 1\}.$$

The problem of sorting on a $2^k \times 2^k$ mesh is obviously related to an indexing function I: given a family of values $(P_{ij})_{ij}$, to sort them means to sort the corresponding 'linear' family $(P_{I(i,j)})_{I(i,j)}$, i.e., to find the permutation σ_k of $\{0, 1, \ldots, 2^{2k} - 1\}$ such that $(P_{\sigma I(i,j)})_{\sigma I(i,j)}$ is ascending (or descending).

Furthermore, when designing or implementing sorting algorithms on the 2D mesh, through an indexing function I they will be translated into algorithms for sorting a linear set, $(P_{I(i,j)})_{I(i,j)}$. But, the linear version of the algorithm, for sorting say an array $\langle P_0, \ldots, P_{2^{2k}-1} \rangle$, has to be such that its translation into 2D-mesh operations be admissible. By this last word we mean to obey certain rules for the functioning of the 2D mesh as a network of processors. One such rule is the possibility of a processor to communicate only with its neighbours. Other rules may involve simultaneous communication with neighbours: a processor may communicate with two neighbours simultaneously, provided they are on the same line or column, and that, if the same register is involved, the old value is read and communicated while the new value is written. Still further rules may involve the parallel functioning of the network: communications in parallel may be allowed only if either *only lines* or *only columns* are involved at one parallel step.

The above mentioned restrictions can be formulated in a formal manner, leading to a notion of *good* indexing function, but this is beyond the scope of this paper. Let us just say for now, that, if the linear version of the algorithm performs a comparison between P_r and P_s, then $P_{I^{-1}(r)}$ and $P_{I^{-1}(s)}$ must be neighbours in the 2D mesh topology. Since the linear version of the algorithm is also parallel, whole pairs of families must be mapped by I^{-1} into adjacent families, and the last ones are naturally the adjacent lines or columns of the mesh. This justifies to a certain degree results present in this paper such as Lemma 3.

The choice of the indexing function $I_k = sRM_k$ provides the connection between the bitonic sorting algorithm as presented in [21] and the bitonic sorting network of [14] and Figure 3.

Let us also note that both ${RM_k}^{-1}$ and ${sRM_k}^{-1}$ are easy to compute. For a linear index $r = i_1 i_2 \cdots i_k j_1 j_2 \cdots j_k$,

$$
{RM_k}^{-1}(r) = {RM_k}^{-1}(i_1 i_2 \cdots i_k j_1 j_2 \cdots j_k) =
$$

$$
= (i_1 i_2 \cdots i_k, j_1 j_2 \cdots j_k) = (r \,\mathsf{div}\, 2^k, r \,\mathsf{mod}\, 2^k),
$$

and similarly for ${sRM_k}^{-1}$.

2.3 The bitonic sorting algorithm

In Batcher's bitonic sorting network [3] of order n, the input is a bitonic sequence a of $n/2$ increasing elements followed by $n/2$ decreasing elements. These two sequences are merged by first applying $n/2$ comparators to a_0 and $a_{n/2}$, a_1 and $a_{(n/2)+1}$, ... $a_{n/2}$ and a_{n-1}. This first-phase partitions the elements into two bitonic sequences of $n/2$ smaller elements and of $n/2$ larger elements. These two bitonic sequences are further sorted by applying two bitonic merging networks of size $n/2$ to each sequence. A bitonic sorting network for 16 elements appears in Figure 3.

Lemma 2. *[3] Given a bitonic sequence $\langle a_1, a_2, \ldots, a_{2n} \rangle$ the following hold.*

1. $d = \langle \min\{a_i, a_{n+i}\}_{i=1}^{n} \rangle = \langle \min\{a_1, a_{n+1}\}, \min\{a_2, a_{n+2}\}, \ldots, \min\{a_n, a_{2n}\} \rangle$ *is bitonic.*
2. $e = \langle \max\{a_i, a_{n+i}\}_{i=1}^{n} \rangle = \langle \max\{a_1, a_{n+1}\}, \max\{a_2, a_{n+2}\}, \ldots, \max\{a_n, a_{2n}\} \rangle$ *is bitonic.*
3. $\max(d) < min(e)$.

By an abuse of notations, we shall refer to a sequence of processors as the sequence of integers stored in one designated register A of the processors at a certain moment. Similarly, we shall use $\min/\max\{P_i, P_j\}$ meaning $\min/\max\{P_i[A], P_j[A]\}$ and refer to such operations as a comparison and interchange of values between processors P_i and P_j.

We shall give a generic algorithm for Batcher's bitonic sorter on an array $\langle P_0, \ldots, P_{2^{2k}-1} \rangle$ of processors, independent of the indexing function used. The algorithm (as illustrated in Figure 3 for $k = 2$) will consist of $2k$ stages, numbered from 1 to $2k$. After each Stage i, the sequence $\langle P_{2^i j}, \ldots, P_{2^i j + 2^i - 1} \rangle$ with

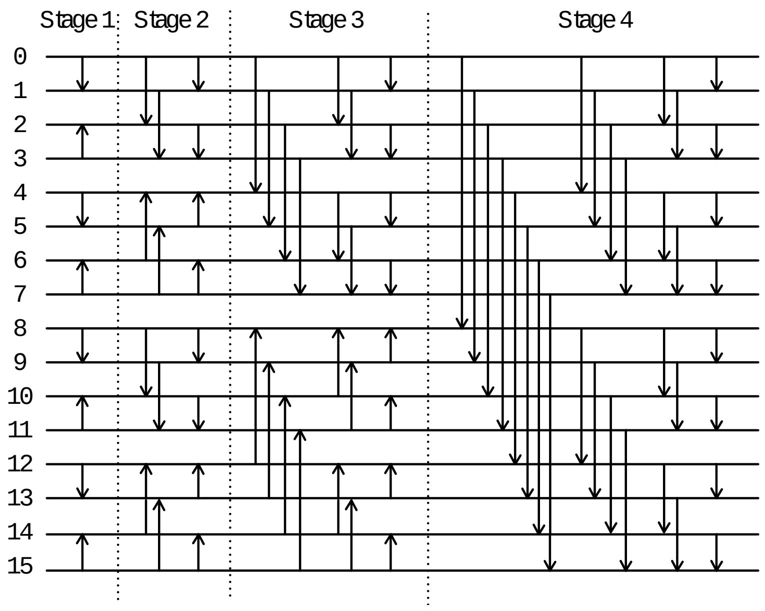

Fig. 3. A bitonic sorting network of size 16

$0 \leq j \leq 2^{k-i} - 1$ will be an ascending sequence for all j even, and a descending sequence, for all j odd.

```
Input: an array $\langle P_0, \ldots, P_{2^{2k}-1} \rangle$ of processors
Output: the sequence $\langle P_0, \ldots, P_{2^{2k}-1} \rangle$ is ascending

Stage($i$)
    for $t \leftarrow i$ downto 1 do
        // compare processors with indices differing on bit $t$
        forall $j \leftarrow 0$ to $2^{2k-t} - 1$ in parallel do
            if $2^t j$ div $2^i$ is even then order = ascending
            else order = descending
            Merge($2^t j, 2^t j + 2^t - 1, order$)
end

Bitonic-Sort
    for $i \leftarrow 1$ to $2k$ do
        Stage($i$)
end
```

Algorithm 1: Bitonic sort on an array of 2^{2k} processors

Given a bitonic sequence of processors $\langle P_1, P_2, \ldots, P_{2n} \rangle$, by **Merge**$(1, 2n, ascending)$ we mean an operation which yields the sequence:

$$\langle \min\{P_1, P_{n+1}\}, \min\{P_2, P_{n+2}\}, \ldots, \min\{P_n, P_{2n}\},$$

$$\max\{P_1, P_{n+1}\}, \max\{P_2, P_{n+2}\}, \ldots, \max\{P_n, P_{2n}\}\rangle.$$

Analogously, a call to **Merge**$(1, 2n, \mathit{descending})$ produces

$$\langle \max\{P_1, P_{n+1}\}, \max\{P_2, P_{n+2}\}, \ldots, \max\{P_n, P_{2n}\},$$

$$\min\{P_1, P_{n+1}\}, \min\{P_2, P_{n+2}\}, \ldots, \min\{P_n, P_{2n}\}\rangle.$$

Theorem 1. *After each Stage i, the sequence $\langle P_{2^i j}, \ldots, P_{2^i j + 2^i - 1}\rangle, 0 \leq j \leq 2^{k-i} - 1$ will be an ascending sequence for all j even, and a descending sequence, for all j odd.*

Proof. We shall reason by induction on i. For the base case $i = 1$ it is immediate that the statement holds. Now let the statement be true for i and show that is it also true for $i + 1$.

First, $t = i + 1$ and $0 \leq j \leq 2^{k-i-1} - 1$. The sequence S for the $i + 1$ case can be written as

$$S = \langle P_{2^{i+1} j}, \ldots, P_{2^{i+1} j + 2^i - 1}\rangle =$$

$$\langle P_{2^i 2j}, \ldots, P_{2^i 2j + 2^i - 1}, P_{2^i (2j+1)}, \ldots, P_{2^i (2j+1) + 2^i - 1}\rangle.$$

From the induction hypothesis, we have that the sub-sequence

$$S_1 = \langle P_{2^i 2j}, \ldots, P_{2^i 2j + 2^i - 1}\rangle$$

is ascending as $2j$ is even for any j, and that

$$S_2 = \langle P_{2^i (2j+1)}, \ldots, P_{2^i (2j+1) + 2^i - 1}\rangle$$

is descending as $2j + 1$ is odd for any j. Therefore, the whole sequence S is bitonic.

At this point we apply the **Merge** operation on S, and get $S' = S'_1 S'_2$. By Lemma 2 we have that S'_1 and S'_2 are both bitonic. Moreover, when doing an ascending merge, $\max(S'_1) < \min(S'_2)$ and when doing a descending merge, $\min(S'_1) > \max(S'_2)$. This ensures that the two sequences are relatively ordered and can be sorted independently in parallel.

For $1 \leq t < i + 1$ the **Merge** operations are the same as in a merging network. We note that for all $2^{i+1} j \leq l < 2^{i+1}(j + 1)$, l div $2^{i+1} = j$ and therefore all subsequent **Merge** operations for $t < i + 1$ on these processors will have the same order as when $t = i + 1$. $\square$

Corollary 1. *Given a sequence $\langle P_0, \ldots, P_{2^{2k}-1}\rangle$ of processors, Algorithm 1 is correct.*

Proof. The proof is immediate by Theorem 1. At Stage k we have $j = 0$ and hence the sequence $\langle P_0, \ldots, P_{2^{2k}-1}\rangle$ is ascending. $\square$

Lemma 3. *Given a $2^k \times 2^k$ 2D-mesh indexed with the function sRM and using Algorithm 1, for any two processors $x = 2^t j + l$ and $y = 2^t j + l + 2^{t-1}$, with $0 \leq l \leq 2^{t-1} - 1$, $1 \leq t \leq i$, and $0 \leq j \leq 2^{k-t} - 1$, which compare and interchange values inside a call of the form* **Merge** $(2^t j, 2^t j + 2^t - 1, order)$*, the following hold:*

(i) the binary representations of x and y differ only on bit t;
(ii) if t is even then x and y reside on the same vertical line of the mesh; if t is odd they are on the same horizontal line;
(iii) the distance on the mesh between x and y is $2^{\lceil t/2 \rceil - 1}$;
(iv) all processors situated on the same line between x and y are involved in the same **Merge** *operation (i.e., have indices between $2^t j$ and $2^t j + 2^t - 1$).*

Proof. (i) Since $x = 2^t j + l$ and $l \leq 2^{t-1} - 1$, we have that l contributes to bits 1 to $t-1$ and that $2^t j$ contributes to bits $t+1$ to $2k$. Therefore bit t of x is 0. Similarly, since $y = x + 2^{t-1}$, bit t of y is 1, and all other bits are the same as those of x.

(ii) We apply the 'un-shuffle' function to x and y and get $ush(x) = i_1 j_1$ and $ush(y) = i_2 j_2$. By the definition of sRM we have that the i is the row index, while j is the column index. From $i)$ we have that x and y differ on bit t, and hence the following two cases hold: t is even and $i_1 \neq i_2$, $j_1 = j_2$, or t is odd and $i_1 = i_2$, $j_1 \neq j_2$. In the first case x and y are on the same column, and in the latter, they are on the same row.

(iii) Using the notations above, let us assume that t is even and $i_1 \neq i_2$, $j_1 = j_2$. If x and y differ on bit t, then i_1 and i_2 will differ on bit $t/2$, and therefore $|i_1 - i_2| = 2^{t/2-1}$. From $ii)$ x and y are on the same line of the mesh and the distance between them is $|i_1 - i_2| = 2^{t/2-1}$. Similarly, when t is odd and $i_1 = i_2$, $j_1 \neq j_2$, we have that j_1 and j_2 differ on bit $\lceil t/2 \rceil$. As before, the distance between x and y is $2^{\lceil t/2 \rceil - 1}$.

(iv) Consider again the case t even and $i_1 \neq i_2$, $j_1 = j_2$. We have to show that for all numbers i with $i_1 \leq i \leq i_2$, we have $2^t j \leq sRM(i, j_1) \leq 2^t j + 2^t - 1$. But since $i_1 \leq i \leq i_2$, form Lemma 1, we have that $x \leq sRM(i, j_1) \leq y$, which concludes our proof as $2^t j \leq x$ and $y \leq 2^t j + 2^t - 1$. Analogously for t odd. □

2.4 Applying the bitonic sorting algorithm to the 2D-mesh

Thompson and Kung [25], and Orcutt [18] showed that Batcher's bitonic sorting algorithm can be applied to sorting on a mesh-connected parallel computer, once the indexing function is chosen. In [25] it is noted that a necessary condition for optimality is that a comparison-interchange on the jth bit be no more expensive than the $(j+1)$th bit, for all j. From (iii) of Lemma 3 we have that the "shuffled row-major" indexing scheme satisfies such condition, and leads to a complexity of $(14(n-1) - 8 \log n) t_R + (2 \log^2 n + \log n) t_C$.

The algorithm for a 4×4 is illustrated below and in Figure 4, where by "well ordered" we reffer to the corresponding comparison directions from Figure 3.

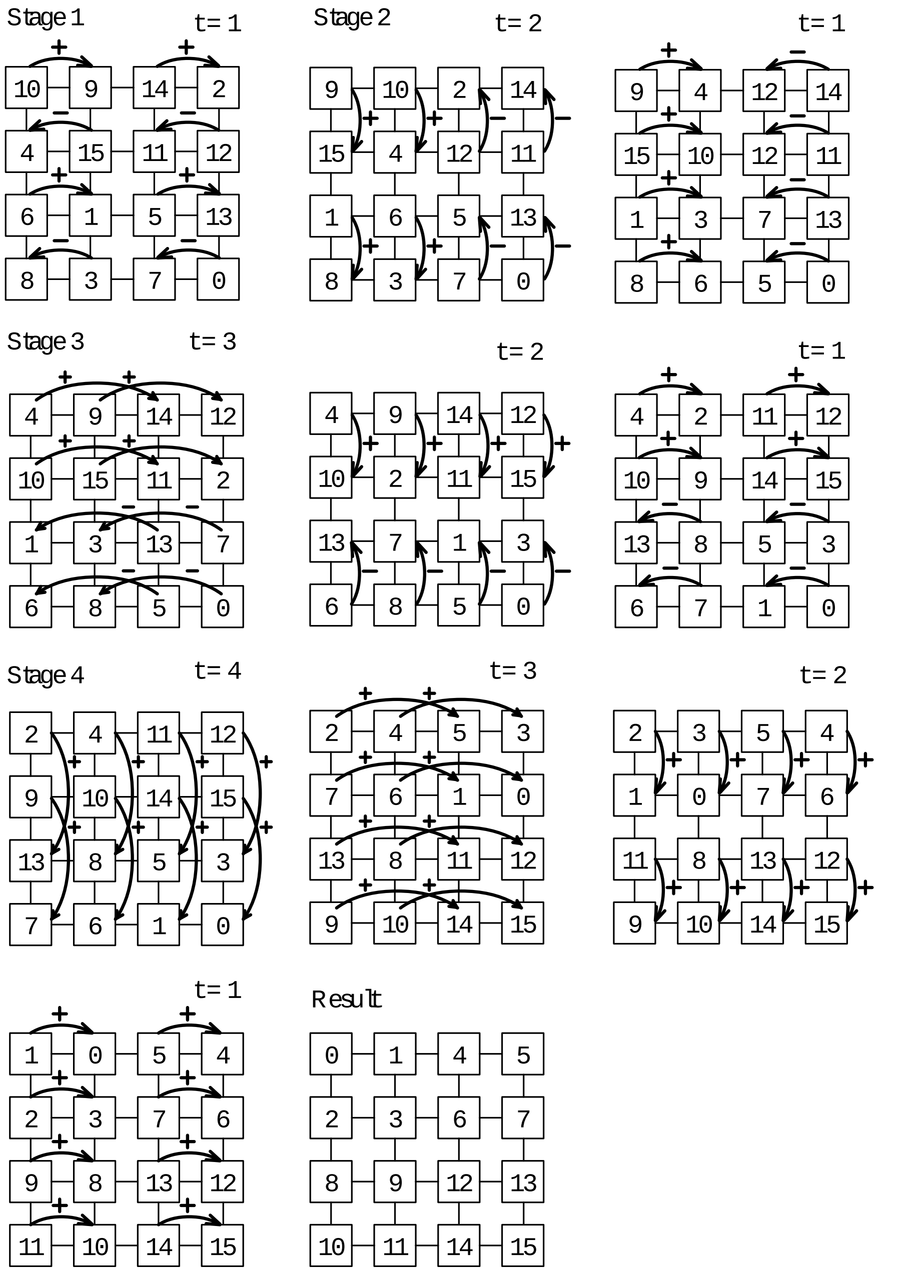

Fig. 4. Bitonic sorting algorithm applied on a 4 × 4 2D-mesh, using shuffled row-major indexing

Stage 1 Bitonic sort on pairs of adjacent 1×1 matrices by the comparison interchange indicated, result: "well ordered" 1×2 matrices. Time: $2t_R + t_C$.
Stage 2 Bitonic sort on 1×2 matrices, result: 2×2 matrices. Time: $4t_R + 2t_C$.
Stage 3 Bitonic sort on 2×2 matrices, result: 2×4 matrices. Time: $8t_R + 3t_C$.
Stage 4 Bitonic sort on the two 2×4 matrices. Time: $12t_R + 4t_C$.

At each stage of Algorithm 1, we have a comparison and interchange of values between two processors. We have seen in Lemma 3 that using the *sRM* indexing function, these two processors will sit on the same vertical or horizontal line of the mesh. In the cases when they are not directly connected, they will have to route their values through neighbour processors, residing on the shortest path between (i.e., the line of the mesh on which they are placed). At each Stage i, we will have a comparison and interchange between processors whose indices differs only on bit t, with $1 \leq t \leq i$. Keeping in mind the parallel structure of our machine, the merging operation becomes a merging of square or rectangular portions of the mesh. Therefore, using the *sRM* indexing, the **Merge** operation defined previously becomes a **Merge** operation on sub-arrays of processors situated on the same line of the mesh. We denote such operation **compare-interchange**.

For a better understanding of the way a call **Merge** $(2^t j, 2^t j + 2^t - 1, order)$ $(1 \leq i \leq 2^k,\ 1 \leq t \leq i,\ 0 \leq j \leq 2^{k-t} - 1)$ is translated to the $2^k \times 2^k$ mesh topology, we shall make the following observations:

Remark 1. The portion of the mesh will have dimensions $2^{t-\lceil t/2 \rceil} \times 2^{\lceil t/2 \rceil}$ (i.e., $2^{t-\lceil t/2 \rceil}$ rows and $2^{\lceil t/2 \rceil}$ columns). This is true since 2^t processors are involved in the **Merge** and from Lemma 3 the maximal length of the sub-arrays involved in the **Merge** situated on the same line is $2 \cdot 2^{\lceil t/2 \rceil - 1}$.

Remark 2. For t even, we have a merging of square portions of the mesh of size $2^{t/2} \times 2^{t/2}$ and the compare interchange operations are done between processors residing on the same column of the mesh, For t odd we have a merging of rectangular portions of the mesh of size $2^{\lceil t/2 \rceil - 1} \times 2^{\lceil t/2 \rceil}$, and the compare interchange operation are done between processors residing on the same row of the mesh.

Let us see what are the necessary routings for the case for $t = 1$ (the processors are directly connected since $2^{\lceil t/2 \rceil - 1} = 1$). Consider a call of the form **Merge**$(x, x+1, order)$ Let processors P_x and P_{x+1} have the two registers denoted by A and B. Then the first instruction performed is a routing from $P_x[A]$ to $P_{x+1}[B]$. Next, perform a comparison operation in processor P_{x+1}, and store the minimum/maximum in register B. Finally, route back to P_x the value of the B register of P_{x+1}, with a total time is $2t_R + t_C$. The pseudo-code is written below, where by **compare**$(P_{x+1}, ascending|descending)$ we understand an internal comparison in processor P_{x+1}, which places the minimal/maximal value in register B.

```
Input: index $x$ and sorting order $order$
Output: the sequence $\langle P_x, P_{x+1} \rangle$ is ordered w.r.t. $order$

route($P_x[A]$, $P_{x+1}[B]$)
compare($P_{x+1}$, order)
route($P_{x+1}[B]$, $P_x[A]$)
```

Algorithm 2: Compare-interchange operation for adjacent processors

Let us now see what is the case when we have to merge an array a of 2^i processors situated on the same line of the mesh, indexed from 0 to $2^i - 1$, and such that $P_{a[j]}$ is neighbour with $P_{a[j+1]}$ for all $0 \leq j < 2^i - 1$. The basic idea is that we have to shift the values of the first half of the array in the B registers of the second half, perform a comparison operation in parallel in these processors, and then shift back the minimal/maximal values. Hence a total time of $2^i t_R + t_C$.

```
Input: array of indices $a$, integer $i$, and sorting order $order$
Output: the sequence $\langle P_{a[0]}, P_{a[1]}, \ldots, P_{a[2^i-1]} \rangle$ is ordered w.r.t. $order$

compare-interchange($a, i, order$)
    forall $j \leftarrow 0$ to $2^{i-1} - 1$ in parallel do
        // route left one unit in the $B$ registers
        route($P_{a[j]}[A]$, $P_{a[j+1]}[B]$)
    for $k \leftarrow 1$ to $2^{i-1} - 1$ do
        // shift the values to the second half of the array
        forall $j \leftarrow 0$ to $2^{i-1} - 1$ in parallel do
            route($P_{a[j+k]}[B]$, $P_{a[j+1+k]}[B]$)
    forall $j \leftarrow 2^{i-1}$ to $2^i - 1$ in parallel do
        // compare internally
        compare($P_{a[j]}, order$)
    for $k \leftarrow 2^{i-1} - 1$ downto $1$ do
        // shift back the results
        forall $j \leftarrow 0$ to $2^{i-1} - 1$ in parallel do
            route($P_{a[j+k+1]}[B]$, $P_{a[j+k]}[B]$)
    forall $j \leftarrow 0$ to $2^{i-1} - 1$ in parallel do
        // final routing back in the $A$ registers
        route($P_{a[j+1]}[B]$, $P_{a[j]}[A]$)
end
```

Algorithm 3: Compare-interchange operation for an array of neighbour processors situated on the same line of the mesh

3 Modeling with membranes

Given the embedded parallel structure of a P system, modeling a 2D-mesh is a natural and straightforward approach. In what follows, we will present two such systems.

3.1 A P system with dynamic communication of 2D-mesh type

The first P system we introduce is along the same general lines as the model proposed in [6]. For each processors P_i, $i \in \{0, 1, \ldots, 2^{2k}-1\}$ we will have an associated membrane which we denote i. The two registers A and B of each processors are coded by two different symbols, say a and b. The number of occurrences of a represents the value of the A register, and analogously for b. 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.

Basically, we have to model:

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

In a slightly different manner from [8] or [9], we shall refer to the communication graph associated to a given architecture with the following conventions: the vertices of the graph are the processors, and the edges (in our case not oriented as communications between processors are both ways) are the network connections characteristic of the architecture.

In the case of the $2^k \times 2^k$ 2D-mesh with the sRM indexing function, let G_{total} be the underlying communication graph composed of all edges necessary to the architecture. We introduce the following notation for the set of vertices of G_{total}:

$$V(G_{total}) = \{0, 1, \ldots, 2^{2k} - 1\}.$$

Hence, the set of edges is

$$E(G_{total}) = \{(sRM(i,j), sRM(i,j+1),) \mid 0 \le i \le 2^k - 1, 0 \le j \le 2^k - 2\} \bigcup$$

$$\{(sRM(i,j), sRM(i+1,j)) \mid 0 \le i \le 2^k - 2, 0 \le j \le 2^k - 1\}.$$

Note that at a certain step of the sorting algorithm not all edges are involved in communication. Therefore we shall call *active sub-graphs* of G_{total} those graphs containing only such edges. We introduce also the *identity* graph, with

$$V(Id) = \{0, 1, \ldots, 2^{2k} - 1\},$$

$$E(Id) = \{(sRM(i,j), sRM(i,j)) \mid 0 \le i \le 2^k - 1, 0 \le j \le 2^k - 1\}$$

for modeling internal processing steps.

As in [6], the P system which we shall consider in the sequel, departs 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 [19]), 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}.

In order to describe the evolution of a P system which simulates the behavior of the bitonic sorting algorithm in the 2D-mesh architecture, we will use pairs $[graph, rules]$. We have *graph* a sub-graph of G_{total} or Id and *rules* a mapping from the set of all edges of *graph*, $E(graph)$, to the set of all symbol/object rewriting rules for routing or comparison operations.

Let R_μ be the finite sequence of pairs $[graph, rules]$ which simulates Algorithm 1, such that: (i) if $E(graph) \subseteq E(Id)$ then its rules are rewriting rules; (ii) if $E(graph) \subseteq E(G_{total})$ then its rules are communication rules.

In order to give such a sequence, we have to closely follow Algorithm 1. In a very intuitive manner, for every **Stage**(i), $1 \leq i \leq 2k$, and for every comparison on bit t, $i \geq t \geq 1$ we will have a sequence of graphs. From Lemma 3, the **Merge** operations executed in parallel in Algorithm 1 involve disjoint sub-matrices of the mesh and have the same length, therefore they can also be executed in parallel when implementing them on a 2D-mesh or P system.

To be more precise, let us analyse a call of the form **Merge**$(2^t j, 2^t j + 2^t - 1, order)$. From Remarks 1-2 we know that the dimensions of the sub-matrix of the mesh involved in the **Merge** are $2^{\lceil t/2 \rceil} \times 2^{t-\lceil t/2 \rceil}$. Hence the maximal sequence of processors situated on the same line which compare and interchange values in a **Merge** operation has length $2^{\lceil t/2 \rceil}$. Using the observations made on Algorithm 3, for each **Merge** operation, we will need a sequence of $2^{\lceil t/2 \rceil} + 1$ graphs. The first $2^{\lceil t/2 \rceil - 1}$ route values in the destination membranes for comparison, then we have an application of the identity graph Id for internal comparisons, and another sequence of $2^{\lceil t/2 \rceil - 1}$ graphs to route back the results. Another important aspect is that for a comparison on bit t, the processors which compare values are the same at every stage, only that the *order* is different. Therefore, we will have the same communication graphs for routing operations, only that the pair $[Id, rules]$ will be different at each **Stage**(i).

Let us denote as below the projection of the first and second argument of sRM^{-1}. These represent the row and column indices, respectively, of a processors indexed with $r \in \{0, 1, \ldots, 2^{2k} - 1\}$.

$$sRM^{-1}_{row}, sRM^{-1}_{col} : \{0, 1, \ldots, 2^{2k} - 1\} \to \{0, 1, \ldots 2^k - 1\}$$

Then, the right / down neighbors of r (defined whenever possible) are:

$$right(r) = sRM(sRM^{-1}_{row}(r), sRM^{-1}_{col}(r) + 1)$$

$$down(r) = sRM(sRM^{-1}_{row}(r) + 1, sRM^{-1}_{col}(r))$$

In order to give an algorithm independent of the parity of bit t, denote (whenever possible):

$$next_t(r) = \begin{cases} right(r), & \text{if } t \text{ odd;} \\ down(r), & \text{if } t \text{ even.} \end{cases}$$

Remark 3. The indices of the first processors on every line l (i.e., the smallest indices on every line l) in a **Merge**$(2^t j, 2^t j + 2^t - 1, order)$ are $sRM(sRM^{-1}_{row}(2^t j) + l, sRM^{-1}_{col}(2^t j))$, with $0 \leq l \leq 2^{t - \lceil t/2 \rceil} - 1$.

Consider two adjacent processors P_x and P_y which need to interchange values. The three possible routing operations are: **route**$(P_x[A], P_y[B])$, **route**$(P_x[B], P_y[B])$, **route**$(P_x[B], P_y[A])$. The implementation with rewriting and communication rules of the first operation follows the lines: rewrite $a \to a^*$ into membrane x, apply the communication rule (a^*, out) along the edge (x, y), which transports all the a^* symbols from membrane x into y, and then in membrane y rewrite a^* back to the desired symbol, in this case $a^* \to b$. We give below a specification of a sequence $[graph, rules]$ accomplishing this routing operation.

$$\begin{gathered} [Id_1, rules_1], [G, rules], [Id_2, rules_2], \text{ such that} \qquad \text{(rAB)} \\ Id_1 \subseteq Id, (x, x) \in E(Id_1), \; rules_1((x, x)) = \{a \to a^*\}, \\ G \subseteq G_{total}, (x, y) \in E(G), \; rules((x, y)) = \{(a^*, out)\}, \\ Id_2 \subseteq Id, (y, y) \in E(Id_2), \; rules_2((y, y)) = \{a^* \to b\}. \end{gathered}$$

Similarly, an operation **route**$(P_x[B], P_y[A])$ is specified as:

$$\begin{gathered} [Id_1, rules_1], [G, rules], [Id_2, rules_2], \text{ such that} \qquad \text{(rBA)} \\ Id_1 \subseteq Id, (x, x) \in E(Id_1), \; rules_1((x, x)) = \{b \to b^*\}, \\ G \subseteq G_{total}, (x, y) \in E(G), \; rules((x, y)) = \{(b^*, out)\}, \\ Id_2 \subseteq Id, (y, y) \in E(Id_2), \; rules_2((y, y)) = \{b^* \to a\}. \end{gathered}$$

In the case of a **route**($P_x[B]$, $P_y[B]$), only one communication graph is needed. The reason for not having supplementary rewritings is that such routings are done in parallel. The value from $P_x[B]$ is routed to $P_y[B]$ in parallel with the routing of $P_y[B]$ to a B register of a neighbor processors. Hence the number of symbols b in membrane y is the desired one $P_x[B]$.

$$[G, rules], \text{ such that} \qquad \text{(rBB)}$$
$$G \subseteq G_{total}, (x, y) \in E(G),\ rules((x, y)) = \{(b, out)\}.$$

Consider now an internal comparison operation in processor P_x, **compare**(P_x, order) which places $\mathbf{max}(P_x[A], P_x[B])$ in register B if the order is ascending, or in register A if the order is descending. This can be formalised as:

$$[Id', rules], \text{ such that } Id' \subseteq Id,\ (x, x) \in E(Id'), \qquad \text{(C)}$$
$$rules((x, x)) = \begin{cases} \{ab \to ab, a \to b, b \to b\}, & \text{if order is ascending,} \\ \{ab \to ab, a \to a, b \to a\}, & \text{if order is descending.} \end{cases}$$

For all $s = 0, 2^{\lceil t/2 \rceil - 1} - 1$ denote with G_s^t (sub-graphs of G_{total}) the communication graphs which simulate all parallel routing operations when comparing of bit t, in Algorithm 1.

From the above considerations and the steps illustrated in Algorithm 3, we introduce the following algorithms: Algorithm 4 to generate the edges of a communication graph, and Algorithm 5 to generate sub-graphs of the Id where comparisons are to be performed:

```
Input: integers $k, t$
Output: communication graphs $G_s^t$, for all $s = 0, 2^{\lceil t/2 \rceil} - 1$
set all $E(G_s^t) \leftarrow \emptyset$
for $j \leftarrow 0$ to $2^{2k-t} - 1$ do
    // for every Merge operation
    for $l \leftarrow 0$ to $2^{t-\lceil t/2 \rceil} - 1$ do
        // for every line in the Merge operation
        for $s \leftarrow 0$ to $2^{\lceil t/2 \rceil - 1} - 1$ do
            // for each communication graph
            node = $sRM(sRM_{row}^{-1}(2^t j) + l, sRM_{col}^{-1}(2^t j) + s)$
            for $q \leftarrow 1$ to $2^{\lceil t/2 \rceil - 1}$ do
                // add the $2^{\lceil t/2 \rceil - 1}$ edges
                $E(G_s^t) \leftarrow E(G_s^t) \cup \{(\text{node}, next_t(\text{node}))\}$
                node = $next_t$(node)
```

Algorithm 4: Generating all communication graphs G_s^t to compare on bit t

```
Input: integers $k, t$
Output: internal processing graphs $Id^t$
set all $E(Id^t) \leftarrow \emptyset$
for $j \leftarrow 0$ to $2^{2k-t} - 1$ do
    // for every Merge operation
    for $l \leftarrow 0$ to $2^{t-\lceil t/2 \rceil} - 1$ do
        // for every line in the Merge operation
        node = $sRM(sRM^{-1}_{row}(2^t j) + l, sRM^{-1}_{col}(2^t j) + 2^{\lceil t/2 \rceil - 1})$
        for $q \leftarrow 1$ to $2^{\lceil t/2 \rceil - 1}$ do
            $E(Id^t) \leftarrow E(Id^t) \cup \{(\text{node},\text{node})\}$
            node = $next_t(\text{node})$
```

Algorithm 5: Generating internal processing graphs Id^t to compare on bit t

Let us denote with G_μ the sequence of graphs simulating the algoritm. Then G_μ can be obtained also algoritmically, using Algorithm 6:

```
set $G_\mu \leftarrow \lambda$
for $i \leftarrow 1$ to $2k$ do
    // compare interchange on bit $t$
    for $t \leftarrow i$ downto 1 do
        // route to the second half
        for $s \leftarrow 0$ to $2^{\lceil t/2 \rceil - 1} - 1$ do
            $G_\mu \leftarrow G_\mu \cdot G^t_s$
        // compare internally
        $G_\mu \leftarrow G_\mu \cdot Id^t$
        //route back to the first half
        for $s \leftarrow 2^{\lceil t/2 \rceil - 1} - 1$ downto 0 do
            $G_\mu \leftarrow G_\mu \cdot G^t_s$
```

Algorithm 6: Generating the sequence of graphs G_μ for simulating the bitonic sorting algoritm on the $2^k \times 2^k$ 2D mesh

where by λ we denote the empty sequence, and by "$\cdot$" we denote the concatenation of two sequences.

The above algorithms could be easily modified to produce a the finite sequence R_μ of pairs $[graph, rules]$ which simulates Algorithm 1. Keeping in mind the way routing and comparison operations are transformed into communication and rewriting rules of a P system (rAB, rBA, rBB, C), every time when adding an edge (x, y) to a graph G (subgraph of G_{total} or Id), the appropriate image $rules((x, y))$ should be specified.

3.2 Bitonic sorting in one membrane

We propose here a simulation of the bitonic sorting, which uses only one membrane. We will use (cooperative) symbol rewriting rules. The cooperation will be 'minimal', i.e., of degree two, since we follow closely the algorithm, and thus the whole process is based on comparators.

Consider an alphabet with 2^{2k} symbols, $V = \{v_0, v_1, \cdots v_{2^{2k}-1}\}$. We will call it the *primary alphabet.*

We will consider also *auxiliary* alphabets, which we will specify in the sequel, in order to achieve sorting by rewritings.

We want to sort in ascending order the sequence of *distinct* integers

$$\langle x_0, x_1, \cdots x_{2^{2k}-1} \rangle,$$

codified over V as the multiset

$$w = {v_0}^{x_0} {v_1}^{x_1} \cdots {v_{2^{2k}-1}}^{x_{2^{2k}-1}}.$$

We want to design a P system which, by rewritings acting in a maximal parallel manner and competing for objects, produces, from the initial configuration w, the configuration

$$w_f = {v_0}^{\sigma(x_0)} {v_1}^{\sigma(x_1)} \cdots {v_{2^{2k}-1}}^{\sigma(x_{2^{2k}-1})},$$

where σ is the permutation which yeilds the total order, i.e., such that $\sigma(x_0) < \sigma(x_1) < \cdots < \sigma(x_{2^{2k}-1})$.

Consider the alphabet V as ordered, by the natural order given by the indices, and let $v = v_0 v_1 \cdots v_{2^{2k}-1}$ be the *alphabet word* (see [1]), i.e., the word obtained by concatenating the letters of V in their natural order. We call *extended alphabet words* over V, all words in V^* in which all the letters appear in their natural order. Note that both w and w_f, the initial and the final configuration of our P system, are extended alphabet words. Actually, all the intermediate configurations over V will be of this type.

Let $M_j(u)$ denote the multiplicity of letter v_j in a word $u \in V^*$. Then

$$w = {v_0}^{x_0} {v_1}^{x_1} \cdots {v_{2^{2k}-1}}^{x_{2^{2k}-1}} = {v_0}^{M_0(w)} \cdots {v_{2^{2k}-1}}^{M_{2^{2k}-1}(w)}.$$

Consider first the case $n = 2$ ($k = 0$). We have 2 integers codified over $\{v_0, v_1\}$ as an extended alphabet word. Consider the auxiliary alphabets

- $\{a, b\}$, for writing sources of a comparator
- $\{c^+, d^+\}$, for writing targets of a $\oplus$-comparator
- $\{c^-, d^-\}$, for writing targets of a $\ominus$-comparator

Consider the rules:

$$C_\oplus = \{v_0 \to a, v_1 \to b\} \cup \{ab \to c^+ d^+, a \to d^+, b \to d^+\} \cup \{c^+ \to v_0, d^+ \to v_1\}.$$

The first group rewrites all v_0s to as and v_1s to bs, the second group performs the comparison and produces the ascending order, and the last group rewrites back into the original alphabet. We have the sequence of configurations

$$v_0{}^{x_0}v_1{}^{x_1} \to a^{x_0}b^{x_1} \to c^{+min(x_0,x_1)}d^{+max(x_0,x_1)} \to v_0{}^{min(x_0,x_1)}v_1{}^{max(x_0,x_1)}.$$

Similarly, the rules:

$$C_\ominus = \{v_0 \to a, v_1 \to b\} \cup \{ab \to c^- d^-, a \to c^-, b \to c^-\} \cup \{c^- \to v_0, d^- \to v_1\},$$

achieve a descending comparator, generating the sequence of configurations

$$v_0{}^{x_0}v_1{}^{x_1} \to a^{x_0}b^{x_1} \to c^{-max(x_0,x_1)}d^{-min(x_0,x_1)} \to v_0{}^{max(x_0,x_1)}v_1{}^{min(x_0,x_1)}.$$

Lemma 4. *On a two-letter alphabet, starting from an initial configuration $w = v_0{}^{x_0}v_1{}^{x_1}$, by applying rules in $C_\oplus$ we obtain w_f such that $(M_i(w_f))_i$ is ascending, and by applying rules in $C_\ominus$ we obtain w_f such that $(M_i(w_f))_i$ is descending.* $\square$

Note that rules $C_\oplus$ simulate a **Merge**$(0, 1, +)$, and $C_\ominus$ a **Merge**$(0, 1, -)$.

We now want to simulate a whole family of merge operations done in parallel.

We take 2 auxiliary alphabets, S^+ and S^- to codify sources of + or − comparators, and another pair, T^+ and T^-, to codify outputs (targets) of + or − comparators. We label them in a bijective correspondence with V.

$$S^+ = \{s_0{}^+, \cdots s^+_{2^{2k}-1}\},$$

$$T^+ = \{t_0{}^+, \cdots t^+_{2^{2k}-1}\},$$

and similarly for −. (For the time being, only 4 copies of the initial alphabet. We will probably need 4 different copies for every stage, in order to keep them independent.)

At Stage (1) we have to simulate **Merge**$(2j, 2j + 1, order)$, for all $0 \le j \le 2^{2k-1} - 1$, where $order = +$ for all j even, and $order = -$ for all j odd.

This is equivalent to:

- Rewrite all symbols of V into start symbols for appropriate comparators, using the sets of rules

$$\{v_{2j} \to s_{2j}{}^+, v_{2j+1} \to s_{2j+1}{}^+ \mid 0 \le j \le 2^{2k-1} - 1 \text{ , j even}\}\cup$$
$$\cup\{v_{2j} \to s_{2j}{}^-, v_{2j+1} \to s_{2j+1}{}^- \mid 0 \le j \le 2^{2k-1} - 1 \text{ , j odd}\}.$$

- Apply in parallel the rewritings of symbols which correspond to the simulations of the comparators:

$$\{s_{2j}{}^+s_{2j+1}{}^+ \to t_{2j}{}^+t_{2j+1}{}^+, s_{2j}{}^+ \to t_{2j+1}{}^+, s_{2j+1}{}^+ \to t_{2j+1}{}^+ \mid$$
$$0 \le j \le 2^{2k-1} - 1 \text{ , j even}\} \bigcup$$
$$\cup\{s_{2j}{}^-s_{2j+1}{}^- \to t_{2j}{}^+t_{2j+1}{}^-, s_{2j}{}^- \to t_{2j}{}^-, s_{2j+1}{}^- \to t_{2j}{}^- \mid$$
$$0 \le j \le 2^{2k-1} - 1 \text{ , j odd}\}.$$

- Rewrite back all symbols of T's into V.

$$\{v_{2j} \leftarrow {t_{2j}}^{+}, v_{2j+1} \leftarrow {t_{2j+1}}^{+} \mid 0 \leq j \leq 2^{2k-1} - 1 \text{ , j even}\} \cup$$

$$\cup \{v_{2j} \leftarrow {t_{2j}}^{-}, v_{2j+1} \leftarrow {t_{2j+1}}^{-} \mid 0 \leq j \leq 2^{2k-1} - 1 \text{ , j odd}\}.$$

The general scheme is as follows:
Input: an extended alphabet word w over V
Output: the extended alphabet word w_f over V, such that $\langle M_i(w_f)\rangle_i$ is ascending

```
Sim-Stage($i$)
    for $t \leftarrow i$ downto 1 do
        Take 4 extra copies of the start and the terminal alphabets, $S_t^+$, $S_t^-$,
        $T_t^+$, $T_t^+$, different for each value of $t$. For $t$'s smaller than $i$ we can
        re-use the alphabets of previous stages.
        forall $j \leftarrow 0$ to $2^{2k-t} - 1$ in parallel do
            if $2^t j$ div $2^i$ is even then order = ascending
            else order = descending
            // Simulate the calls Merge($2^t j, 2^t j + 2^t - 1, order$)
            (WF) Rewrite all symbols in $V$ with the appropriate symbol in
            $S_t^+ \cup S_t^-$.
            (C) Apply the rewritings which simulate the appropriate
            comparators.
            (WB) Rewrite back all symbols in $T_t^+ \cup T_t^-$ to symbols of $V$.
end

Sim-Bitonic-Sort
    for $i \leftarrow 1$ to $2k$ do
        Sim-Stage($i$)
end
```

Algorithm 7: Simulating bitonic sort on an alphabet of 2^{2k} letters V

The calls to **Merge**$(2^t j, 2^t j + 2^t - 1, order)$ are equivalent to parallel calls to **Merge**$(x, y, order)$, where x and y are like in Lemma 3. The same result ensures us that, both the rewritings which feed the comparators, and the rewritings which implement the comparators can be done in parallel. For **Merge**$(x, y, order = -)$, we use

$$\{{s_x}^{-}{s_y}^{-} \rightarrow {t_x}^{-}{t_y}^{-}, {s_x}^{-} \rightarrow {t_x}^{-}, {s_y}^{-} \rightarrow {t_x}^{-}\}.$$

We propose the following sets of rules for simulating iteration t at **Sim-Stage(i)**:

(WF) Rewritings to S's, with $* = \begin{cases} +, & \text{if } 2^t j \text{ div } 2^i \text{ is even,} \\ -, & \text{if } 2^t j \text{ div } 2^i \text{ is odd,} \end{cases}$

$$\{v_x \rightarrow {s_x}^{*} \in {S_t}^{*} \mid x \in [2^t j, 2^t j + 2^{t-1}), 0 \leq j \leq 2^{2k-t+1} - 1\}.$$

(C)Rewritings which simulate the comparators, for appropriate pairs of indices:

$$\{{s_x}^+{s_y}^+ \to {t_x}^+{t_y}^+, {s_x}^+ \to {t_y}^+, {s_y}^+ \to {t_y}^+ \mid$$
$$x \in [2^t j, 2^t j + 2^{t-1}),\ y = x + 2^{t-1},\ 0 \le j \le 2^{2k-t} - 1\},$$
$$\{{s_x}^-{s_y}^- \to {t_y}^-{t_x}^-, {s_x}^- \to {t_x}^-, {s_y}^- \to {t_x}^- \mid$$
$$x \in [2^t j, 2^t j + 2^{t-1}),\ y = x + 2^{t-1},\ 0 \le j \le 2^{2k-t} - 1\}.$$

(WR)Rewritings from T's:

$$\{v_x \leftarrow {t_x}^* \in {T_t}^* \mid x \in [2^t j, 2^t j + 2^{t-1}), 0 \le j \le 2^{2k-t+1} - 1\}.$$

4 Conclusions and open problems

We have presented a bitonic sorting algorithm which can be implemented on a 2D mesh of processors. The dependence between its performance and the choice of the indexing function still remains to be fully explored. However, we believe that we have proved some results which explain the choice of sRM as a "good" indexing function.

We have not yet found in the literature a formal proof of the correctness of bitonic sorting, an equivalent, or an analogue of our Theorem 1.

Much work remains to be done concerning the proposed simulations with P systems. The first simulation, derived in a "straightforward" manner from the functioning of the algorithm on the mesh, is inspired from work in [6], [8], [9], and [7], where the general framework was abstracted. It introduces a generative approach to the sequence of communication graphs, a feature to be explored in subsequent work. The second one is at the opposite pole: it requires no routings of values at all, just an appropriate codification of the symbols. It is in this area that other versions of the algorithm could be implemented, independent of the topology of a given structure, and the parallel features of the P systems can be compared against those of other computational devices.

References

1. A. Alhazov, D. Sburlan, Static Sorting P Systems, Chapter 8 in Applications of Membrane Computing, G. Ciobanu, Gh. Păun, M.J. Pérez Jiménez (Eds.), Springer 2006
2. J.J. Arulanandham, Implementing Bead–Sort with P Systems, Unconventional Models of Computation 2002 (C.S. Calude, M.J. Dinneen, F. Peper, Eds.), Lecture Notes in Computer Science 2509, Springer, 2002, 115–125.
3. K. Batcher, Sorting Networks and their Applications, Proc. of the AFIPS Spring Joint Computing Conf., Vol.32, 1968, pp. 307-314
4. R. Ceterchi, C. Martín–Vide, Dynamic P Systems, Membrane Computing International Workshop, WMC-CdeA 2002, Curtea de Argeş, Romania, August 2002, Revised Papers (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron Eds.), LNCS 2597, Springer, Berln, 2003, 146-186

5. R. Ceterchi, C. Martín–Vide, P Systems with Communication for Static Sorting: GRLMC Report 26 (M. Cavaliere, C. Martín–Vide, Gh. Păun, eds.), Rovira i Virgili University, Tarragona, 2003
6. R. Ceterchi, M.J. Pérez Jiménez, On two-dimensional mesh networks and their simulation with P systems, LNCS 3365, 2005, 259–277
7. R. Ceterchi, M.J. Pérez Jiménez, On simulating a class of parallel architectures, Intern. J. Found. Computer Sci., 17 (1), 2006, 91–110
8. R. Ceterchi, M.J. Pérez Jiménez, Simulating Shuffle–Exchange Networks with P Systems, Proceedings of the Second Brainstorming Week on Membrane Computing, (Gh. Păun, A. Riscos, F. Sancho and A. Romero Eds.), Report RGNC 01/04, 2004, 117-129
9. R. Ceterchi, M.J. Pérez Jiménez, A Perfect Shuffle Algorithm for Reduction Processes and its Simulation with P Systems, Proc. Inter. Conf. on Computers and Communications ICCC 2004 (I. Dzitac, T. Maghiar, C. Popescu Eds.), Baile Felix Spa - Oradea, Romania, Editura Univ. Oradea, 2004, 92-97
10. G. Ciobanu, Gh. Păun, M.J. Pérez Jiménez (Eds.), Applications of Membrane Computing, Springer 2006
11. P.F. Corbett, I.D. Scherson, Sorting in Mesh Connected Multiprocessors, IEEE Transactions on Parallel and Distributed Systems, Vol. 3, No. 5, 1992, 626 – 632
12. M. Dowd, Y. Perl, L. Rudolph, M. Saks, The periodic balanced sorting network, Journal of the ACM, 36, 4 (1989), 738–757
13. Y. Han, Y. Igarashi, M. Truszczynski, Indexing functions and time lower bounds for sorting on a mesh-connected computer, Discrete Applied Math., 36, 2 (1992), 141–152
14. D.E. Knuth, The Art of Computer Programming, Vol. 3, Sorting and Searching, Addison-Wesley, Reading, Mass., 1973
15. C. Layer, H.-J. Pfleiderer, A Reconfigurable Recurrent Bitonic Sorting Network for Concurrently Accessible Data, LNCS 3203, 2004, 648–657
16. D. Nassimi, S. Sahni, Bitonic sort on a mesh connected parallel computer, IEEE Transactions on Computers, C28(1), January 1979
17. D. Nassimi, S. Sahni, An Optimal Routing Algorithm for Mesh-Connected Parallel Computers, Journal of the ACM, Vol. 27, No. 1, January 1980, pp. 6-29
18. S.E. Orcutt, Computer Organization and algorithms for very high speed computations, Ph.D. Th., Stanford U.,Stanford, Calif., 1974, Chap. 2, pp. 20-23
19. Gh. Păun, Computing with Membranes, Journal of Computer and System Sciences, 61, 1 (2000), 108–143
20. Gh. Păun, Membrane Computing. An Introduction, Springer-Verlag, Berlin, 2002
21. M.J. Quinn, Parallel Computing. Theory and Practice,, McGraw–Hill Series in Computer Science, 1994
22. K. Sado, Y. Igarashi, Some parallel sorts on a mesh-connected processor array and their time efficiency, J. Parallel and Distributed Computing, Vol. 3, No. 3, 1986, 398 – 410
23. H.J. Siegel, The universality of various types of SIMD machine interconnection networks, Proc. 4th Annual Symposium on Computer Architecture, 23–25, 1977
24. H.S. Stone, Parallel processing with the perfect shuffle, IEEE Transactions on Computers, C-20(2), 153–161, 1971
25. C.D. Thompson, H.T. Kung, Sorting on a mesh-connected parallel computer, Communications of the ACM, 20, 4 (1977), 263–271
26. The membrane computing web page: `http://psystems.disco.unimib.it`