Sorting Omega Networks Simulated with P Systems: Optimal Data Layouts

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] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mails: rceterchi@gmail.com, marper@us.es, alexandru.tomescu@gmail.com

Summary. The paper introduces some sorting networks and their simulation with P systems, in which each processor/membrane can hold more than one piece of data, and perform operations on them internally. Several data layouts are discussed in this context, and an optimal one is proposed, together with its implementation as a P system with dynamic communication graphs.

1 Introduction

Paper [9] proposed two models to sort a sequence of N numbers, based on the bitonic sorting network. The first one consisted of N membranes, each storing two numbers; one number was an element of the sequence, and the other one was an auxiliary register used to route values. A number x was codified as the number of appearances of a symbol a in each membrane. Moreover, the membranes were disposed on a 2D-mesh, where only communication between neighbor membranes on the mesh are permitted. This model, using a variant of P Systems, called P systems with dynamic communication graphs, (see [8]), follows closely the implementation of the bitonic sort on the 2D-mesh.

The second model consisted of only one membrane, where all the N numbers were encoded as occurrences of N different symbols. Restrictions on communication were no longer imposed, as if the underlying communication graph were the complete graph.

In this paper we introduce a model in between the two. First of all, observe that the first model has the advantage of a codifying alphabet of fixed size, while the second has the advantage of a small communication overhead. The model we put forth in this paper captures these two benefits. Each membrane holds a fixed number of values, and each of the membranes can communicate with any other.

Additionally, in order to minimize the communication between membranes, we use a periodic remap of values to membranes, according to the steps of the omega network.

The problem of mapping values to processors has been previously addressed in the context of parallel sorting algorithms. The bitonic sorting network, which can sort N keys in time $O(\log^2 N)$, is probably one of the most well-known parallel sorting algorithms. However, modern architectures differ greatly from the theoretical models under which such good results were obtained. As coarse-grained processors can store internally more than one value, the following problem arises: how to map N keys to P processors ($N > P$), such that inter-processor communication is minimized. In the bitonic sorting algorithm, and for $N \geq P^2$, the solution given in [13, 14] consisted in alternating a blocked layout with a cyclic layout, performing thus the minimal number of remaps. This paper gives an optimal mapping strategy for the bitonic sort for any $N > P$, and then applies this result to P Systems.

The paper is organized as follows. Section 2 presents preliminaries on bitonic sorting networks and defines omega networks. Section 3 approaches the problem of mapping N keys among P processors, each processor manipulating $n = N/P$ keys, such that overall communication is minimized. Optimal data layouts for the omega network are proposed along the lines of [20], and some essential results are proved about them. Section 4 discusses about internal processing in one processor, and how we model it in our implementation with P systems. Section 5 introduces the P system which simulates the omega network with optimal data layouts, and the algorithms which generate the sequence of dynamic communication graphs of this model. Complexity issues are addressed at the end of Sections 3 and 5.

2 Preliminaries on Bitonic Sorting Networks and Omega Networks

A bitonic sequence is a concatenation of two monotonic sequences, one ascending, and the other one descending, or a sequence such that a cyclic shift of its elements would put them in such a form.

The key components of a bitonic network are the bitonic splitters and the bitonic mergers. The splitter of size N takes as input a bitonic sequence of length N and partitions it in two bitonic sequences of equal length, such that all the elements in the first sequence are smaller than (or greater than) all the elements in the second sequence. A bitonic merger of size N consists of a splitter of size N and of two mergers of size $N/2$, of opposite direction. It accepts as input a bitonic sequence and sorts it in ascending or descending order (direction).

As any sequence of two numbers is bitonic, the sorting network uses bitonic mergers of increasing size and alternating direction to construct bitonic sequences of increasing length. The last such merger, of size N, renders the whole sequence of N numbers sorted.

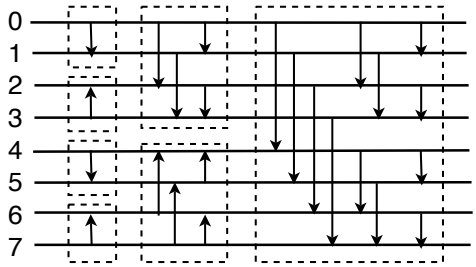
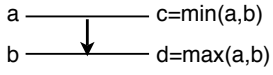
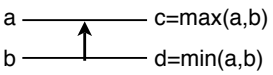


Fig. 1. A bitonic sorting network of size $N = 8$. The network can be partitioned in three stages, each containing bitonic mergers of size 2, 4, and 8, respectively.



(a) Increasing comparator



(b) Decreasing comparator

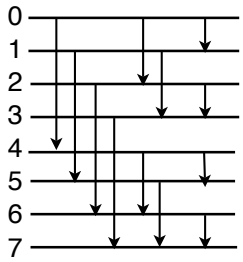
Fig. 2. Network devices

Following [15] it is customary to represent a network as an ordered set of N *lines* (wires) connected by a set of compare-exchange devices (*comparators*, for brevity). A comparator has two input terminals, a and b, and produces two output terminals c and d. If the comparator is increasing, Fig. 2(a), then $c = \min(a, b)$ and $d = \max(a, b)$, while if the comparator is decreasing, Fig. 2(b), $c = \max(a, b)$ and $d = \min(a, b)$. A bitonic sorting network for $N = 8$ is represented in Fig. 1.

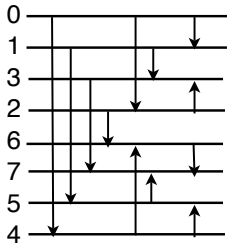
We introduce some more notations regarding the serial and parallel connections of networks T_1 and T_2, of size N. Their serial connection, T_1T_2, is a network in which the i-th output terminal of T_1 is connected to the i-th input terminal of T_2. The parallel connection, $T_1 \circ T_2$, is the union of T_1 and T_2, with terminal i of T_1 becoming terminal i of $T_1 \circ T_2$, and terminal i of T_2 becoming terminal $i + N$ of $T_1 \circ T_2$ $(i = 0, \ldots, N - 1)$.

Definition 1 (Omega network, Fig. 3(d)). *Let D_k, $k \geq 1$ be a one-step network of $N = 2^k$ lines with a device between the pair of lines $(i, i + N/2)$, for $i = 0 \ldots N/2 - 1$. Then the omega network OM_k is recursively defined as $OM_k = D_k(OM_{k-1} \circ OM_{k-1})$.*

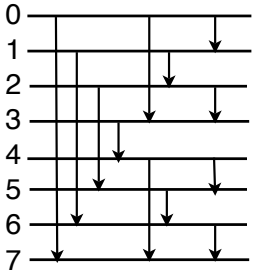
In [6] the striking similarity between the bitonic merger (Fig. 3(a), 3(b)) and the balanced merger (Fig. 3(c)) is investigated. Although prior research [11] showed that there is no permutation of lines to transform the bitonic merger into a balanced merger, a framework is developed under which it is shown that the two



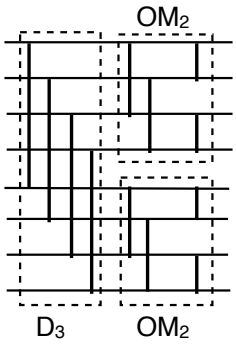
(a) The bitonic merger - classical representation



(b) The bitonic merger - after a permutation of lines



(c) The balanced merger



(d) The omega network OM_3

Fig. 3. The bitonic merger, the balanced merger, and the omega network of size 8

mergers are isomorphic graphs, also isomorphic to the graph of the omega network (Fig. 3(d)).

As a serial connection of $\log N$ identical networks in the class of omega networks forms a sorting network [6], in what follows we will concentrate mainly on the omega network.

3 How They Communicate

A sorting network is a fine-grained theoretical model, containing exactly one input key on each wire. Additionally, comparators require communication between wires,

which can sometimes be more time consuming than the comparison operation itself [1, 3, 10, 16]. When redesigning parallel sorting algorithms for coarse-grained PRAM, one has to pay particular attention to both communication and computation.

Given N keys and P processors ($N > P$), we have to map $n = N/P$ keys to each processor, such that overall communication is minimized. Ionescu and Schauser [13, 14] investigated this problem for the bitonic sorting algorithm. As initially suggested in [10], they proposed a smart periodical switch between a blocked layout and a cyclic layout. They observed that in each stage of the sorting algorithm, the last $\log n$ steps can be performed locally under a blocked layout, while under the cyclic layout the first $\log n$ steps are local. A necessary condition for the two layouts to span enough depth to cover an entire stage of the network is $N \geq P^2$. In addition, the two layouts are particular to the sorting network being implemented. We shall see, for example, that the balanced merger [11, 12], which, as the bitonic merger, belongs to the class of omega networks, also admits data layouts optimizing overall communication.

An approach from the opposite side was put forth by Lee and Batcher [17]. They used a parity strategy for a shared-memory model with $N = 2P$ to store even-parity keys in local memory, while only odd-parity keys were recirculated. This decreased by a factor of 2 the number of shared memory references.

The main contribution of this paper is a general scheme to map N values to P processors, for any $N > P$ and for any sorting network with the topology of the omega network. Our idea captures the essence from the alternating smart layout of [14], and makes it generally applicable, even when $N < P^2$. The number of data layouts is no longer two, but it depends on the granularity of the processors.

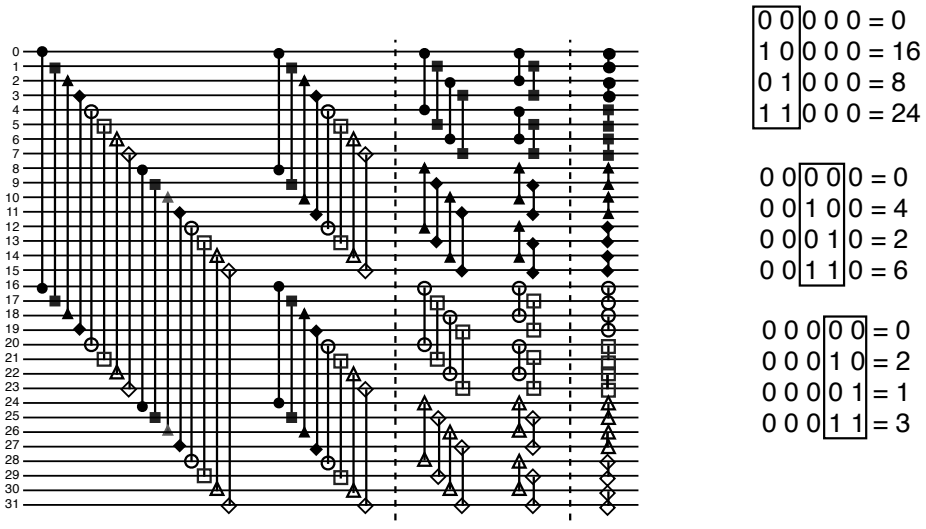
3.1 Optimal Data Layouts for the Omega Network

In the following, without explicitly mentioning it, we assume we have to sort $N = 2^k$ keys using P processors, $N > P$, each processor holding $n = N/P$ keys. Any number $i \in \{0, \ldots, 2^k - 1\}$ has a bit representation $i = a_1 a_2 \cdots a_k$, a_1 being the most significant bit, and a_k the least significant one. To simplify notation, we say that a sequence of bits $a_j \cdots a_i$, where $i, j \in \{1, \ldots, k\}$ and $j > i$, stands for the void sequence. The number of parallel steps of OM_k is k, and step t of the omega network OM_k contains devices linking lines whose bit representations differ of bit t, with $1 \leq t \leq k$. For any $t \in \{1, \ldots, k\}$, consider the function $bc_t : \{0, 1, \ldots, 2^k - 1\} \longrightarrow \{0, 1, \ldots, 2^k - 1\}$, the bit complement of the t-th bit, defined by $bc_t(a_1 a_2 \cdots a_t \cdots a_k) = a_1 a_2 \cdots \bar{a}_t \cdots a_k$. The function bc_t is injective and idempotent.

First, we give a formal definition of a data layout.

Definition 2 (Data layout). *A data layout of N values to P processors is a function $\mathcal{D} : \{0, \ldots, N-1\} \to \{0, \ldots, P-1\}$.*

We introduce the following data layouts, as suggested in [10, 14].



(a) An omega network on size 32. Lines marked with same shape are assigned to the same processor in one data layout.

(b) Keys mapped to processor 0 in each of the three data layouts

Fig. 4. Three data layouts for the omega network OM_5.

Definition 3 (Blocked layout). *A blocked layout for mapping N keys on P processors is a function $\mathcal{D}^b : \{0, \ldots, N-1\} \to \{0, \ldots, P-1\}$, such that $\mathcal{D}^b(i) = \lfloor i/n \rfloor$, where $n = N/P$.*

Definition 4 (Cyclic layout). *A cyclic layout for mapping N keys on P processors is a function $\mathcal{D}^c : \{0, \ldots, N-1\} \to \{0, \ldots, P-1\}$, such that $\mathcal{D}^c(i) = i \bmod P$.*

We note that Definition 5 in [13], and the definition for the cyclic layout indicated in Section 2.1 of [14] are incorrect, since if we map the i-th key to the i mod n processor, where $n = N/P$, we have that $n \leq P$, which implies $N \leq P^2$, which clearly is not the case considered.

In a blocked layout, the first $\log N - \log n$ steps require remote communication, while the last $\log n$ steps are local. In a cyclic layout, the situation is reversed: the first $\log N - \log n$ steps are local, while the last $\log n$ steps are remote. The idea

proposed in [14] when mapping $N \geq P^2$ values in the bitonic sort is to periodically switch between the two layouts, such that all steps are local. Moreover, as the stages in a bitonic sort have increasing size, the author proposes an improved "smart" remap such that a layout spans through multiple stages of the algorithm, achieving a total of $\log P + 1$ remaps.

Our paper better highlights the reasoning behind these remaps, in the case of the bitonic sort. Consider the omega network OM_k, and consider we choose to map key 0 to processor 0. If each processor can hold 2^m values, which other keys are mapped to processor 0? As we can see, at step 1 we have a device linking line 0 with line $0 + 2^{k-1}$. At step 2 we have a device linking line 0 with line 2^{k-2}, and a device linking line 2^{k-1} and line $2^{k-1} + 2^{k-2}$. We also note that in step 1 lines 2^{k-2} and $2^{k-1} + 2^{k-2}$ were also linked with a device. We continue until step m, where we identify 2^m lines linked by 2^{m-1} devices. It would be natural to map these lines to processor 0, as all comparisons at step m are local. However, one more problem remains: all comparisons at stages 0 through $m-1$ are also local? As we shall see, the answer is yes.

The following lemma is straightforward from the definition of OM_k.

Lemma 1. *At each step $1 \leq t \leq k$ of OM_k, and for any $0 \leq i < 2^k$, line i is linked by a device only with line $bc_t(i)$.*

Lemma 2. *In OM_k, for any $0 \leq i < 2^{k-m}$, $1 \leq m \leq k$ and $0 \leq t \leq k-m$, in steps $t+1, \ldots, t+m$ there is no device linking lines in the set $P_i^{t,m} = \{a_1 a_2 \cdots a_k \mid a_1 \cdots a_t a_{t+m+1} \cdots a_k = i$, where $a_1 \cdots a_k$ is a bit representation$\}$ with lines from $\{0, \ldots, 2^k - 1\} \setminus P_i^{t,m}$.*

Proof. Suppose there are $1 \leq r \leq m$, $l \in P_i^{t,m}$ and $l' \notin P_i^{t,m}$ such that at step $t + r$ there is a device linking l and l'. From Lemma 1 we have that $l' = bc_{t+r}(l)$, which implies $l' \in P_i^{t,m}$, a contradiction.

We can therefore derive the data layouts for the omega network. Suppose we have $N = 2^k$, $n = 2^m$, and $P = 2^{k-m}$. We first assign to each processor P_i all values in the set $P_i^{0,m}$, for $0 \leq i \leq P - 1$. By Lemma 2 we have that the first $\log n = m$ steps are entirely local. After m steps, we remap to each processor P_i all the values in the set $P_i^{m,m}$, and perform the next m stages locally, and so on. We can now give the definition of our proposed data layout.

Definition 5. *Given $N = 2^k$ keys and $P = 2^{k-m}$ processors, which can store $n = 2^m$ values, $m \geq 1$, the sequence of optimal data layouts consists of $\lceil \log N / \log n \rceil = \lceil k/m \rceil$ data layouts. In each data layout $\mathcal{D}_s$, $0 \leq s \leq \lceil k/m \rceil - 1$, values in the set $P_i^{sm,m}$ are mapped to processor P_i, for all $0 \leq i \leq 2^{k-m}$. More formally, for any $0 \leq u < 2^k$ such that $u \in P_i^{sm,m}$, we have $\mathcal{D}_s(u) = i$.*

The following is a consequence of Lemma 1 of [14].

Lemma 3. *The maximum number of successive steps of the omega network that can be executed locally, under any data layout is $\log n$, where $n = N/P$.*

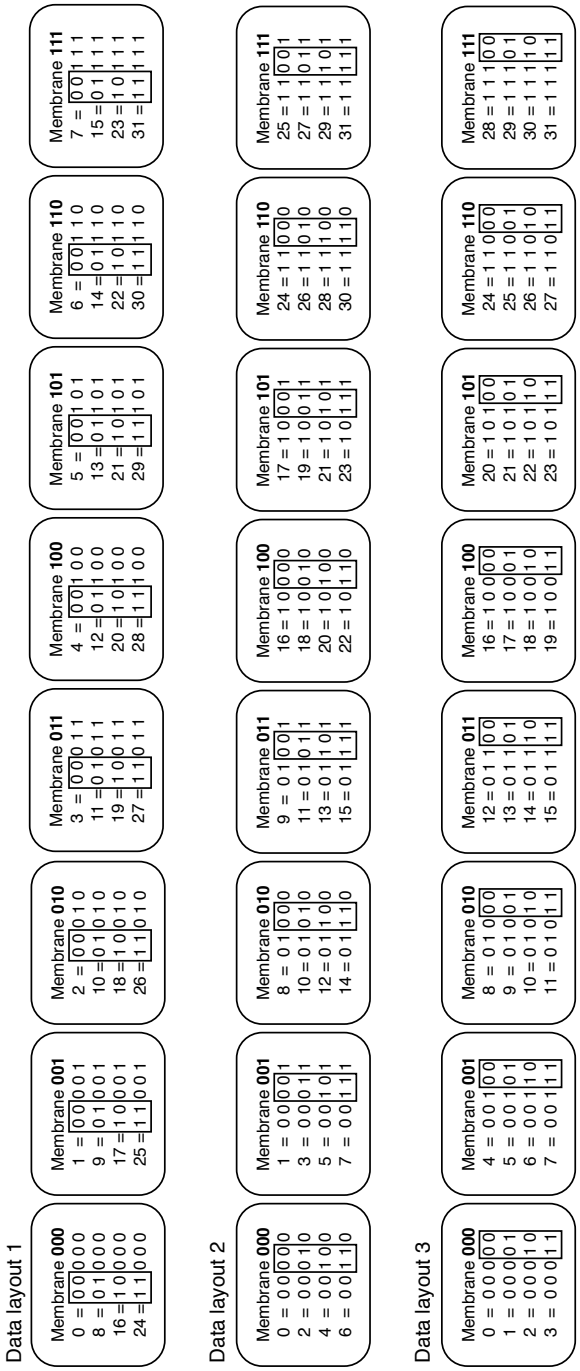


Fig. 5. The three data layouts for the omega network in Figure 4(a).

In each data layout $\mathcal{D}_s$, $0 \leq s \leq \lceil k/m \rceil - 2$, $\log n = m$ steps are local. For $s = \lceil k/m \rceil - 1$, the last $k \bmod m$ steps of the network are local. From Lemma 3 we have that the proposed data layouts for the omega network are optimal.

In the case $N \geq P^2$, we notice that $2m > k$, hence two data layouts are enough to cover the whole omega network. However, they do not coincide with $\mathcal{D}^b$ or $\mathcal{D}^c$, as in the blocked layout, the last m stages are local, while in the cyclic layout, the first $k - m$ stages are local.

3.2 Computation Complexity

In each data layout, a processor holds n values and performs $\log n$ steps locally, taking time $O(n \log n)$. As we have $\lceil \log N / \log n \rceil$ data layouts, we get an overall time complexity of the omega network of $O(n \log N)$. From [6] we have that a serial connection of $\log N$ omega networks of size N is enough to sort a sequence of N numbers. Hence, the complexity to sort N numbers using P processors, each holding $n = N/P$ values, using our proposed data layouts, is $O(n \log^2 N)$.

This remark has a quite profound significance. In the fine-grained theoretical model we have $n = 1$, and its complexity is $O(\log^2 N)$. The complexity of the network using a more coarse-grained model depends linearly on the degree of parallelism of the model. At the opposite end, when $n = N$ and the entire sorting network is simulated locally, we have a complexity of $O(N \log^2 N)$, which is worse than $O(N \log N)$, the complexity of most sequential sorting algorithms. It would be desirable to choose n such that this bound is not surpassed in the parallel model. We impose $n \log^2 N \leq N \log N$, which implies $n \leq N / \log N$.

An algorithm to find the minimum of a bitonic sequence of size n in time $O(\log n)$, was introduced in [14]. This gives a time complexity of each data layout of $O(n)$. In the case of a network obtained from a serial connection of bitonic mergers, this observation gives an overall time complexity of $O(\frac{n}{\log n} \log^2 N)$.

4 What Happens Inside One Processor/Membrane

One processor (and the membrane which simulates it) will be capable of holding $n = N/P = 2^m$, pieces of data. We label the data with indices in the set $\{0, 1, \cdots, n-1\}$. For any such index we consider its writing as a binary string of length m, for instance $i = x_1 x_2 \cdots x_t \cdots x_m$.

Inside one processor, several comparisons are performed, in parallel, between the n pieces of data, in the following manner: for every bit t, (starting with 1, the most significant bit, and ending with m) we compare and exchange if necessary (to obtain an increasing order) all pairs of values codified with a_i and $a_{bc_t(i)}$. More precisely, we have the following algorithm to be performed inside each processor/membrane:

for $t \leftarrow 1$ **to** m **do**
  forall $i < bc_t(i)$ **in parallel do**
    compare$(a_i, a_{bc_t(i)})$;

Algorithm 1: A parallel algorithm for the bitonic merger

where by **compare**(a_i, a_j) we denote sorting in an ascendant manner the values codified by a_i and a_j, i.e. we end by having the minimum of the two values codified by a_i and the maximum by a_j.

The procedure **compare**(a_i, a_j) works in a membrane in the following manner: let s_i, s_j and t_i, t_j be four auxiliary symbols, for the sources and the targets of a comparator. The set of rules

$$\{a_k \to s_k \mid k = i, j\} \cup \{s_i s_j \to t_i t_j, s_i \to t_j s_j \to t_j\} \cup \{t_k \to a_k \mid k = i, j\}$$

implement an increasing comparator between values codified by a_i and a_j. We first rewrite the as to ss, next we have the comparator which writes the minimum to t_i and the maximum to t_j, and then we rewrite these back to a_i and a_j respectively.

For all the comparisons which are to be done in parallel, take auxiliary alphabets $S = \{s_0, \cdots, s_{n-1}\}$ and $T = \{t_0, \cdots, t_{n-1}\}$. We rewrite all initial symbols to symbols in S:

$$\{a_i \to s_i \mid i = 0, 1, \cdots, n-1\}.$$

Next we put the comparators between appropriate pairs:

$$\{s_i s_j \to t_i t_j, s_i \to t_j, s_j \to t_j \mid i = 0, 1, \cdots, n-1, i < j = bc_t(i)\}.$$

Then we rewrite back to the original alphabet:

$$\{t_i \to a_i \mid i = 0, 1, \cdots, n-1\}.$$

The parallel comparisons at each step t

forall $i < bc_t(i)$ **in parallel do**
  compare$(a_i, a_{bc_t(i)})$;

will thus be simulated in a membrane P by the rules

$$\{a_i \to s_i \mid i = 0, 1, \cdots, n-1\} \cup$$
$$\cup \{s_i s_j \to t_i t_j, s_i \to t_j, s_j \to t_j \mid i = 0, 1, \cdots, n-1, i < j = bc_t(i)\} \cup$$
$$\cup \{t_i \to a_i \mid i = 0, 1, \cdots, n-1\}.$$

5 A P System which Simulates the Omega Network

In this section we introduce a P system with dynamic communication [7], along the same general lines as the model proposed in [8, 9]. For each of the processors

$\mathcal{P}_i$, $i \in \{0, 1, \ldots, P-1\}$ we have an associated membrane, which we label i. The graphs we consider are sub-graphs of the complete graph, K_P, or of the identity graph.

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

$$V(Id) = \{0, 1, \ldots, P-1\},$$

$$E(Id) = \{(i, i) \mid 0 \leq i \leq P-1\}$$

for modeling internal processing steps.

In order to describe the evolution of such a P system, we use pairs of the type $[graph, rules]$. We have $graph$ a sub-graph of K_P 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.

The formal definition of the P system is

$$\begin{aligned} \Pi = (V = \{a_0, \ldots, a_{n-1}\} \cup \mathcal{A}, \langle [a_0^{x_0^0}, a_1^{x_1^0}, \ldots, a_{n-1}^{x_{n-1}^0}]_0, \ldots, \\ [a_0^{x_0^{P-1}}, a_1^{x_1^{P-1}}, \ldots, a_{n-1}^{x_{n-1}^{P-1}}]_{P-1} \rangle, R_\mu), \end{aligned}$$

where the membrane indices are $\{0, 1, \ldots, P-1\}$. The alphabet $\{a_0, \ldots, a_{n-1}\}$ is of fixed size, and the set $\mathcal{A}$ contains the auxiliary symbols necessary to simulate the omega network, as indicated in Section 4. Numbers x_i^j with $0 \leq i \leq n-1$ are the values stored on the wires mapped to processor j, $0 \leq j \leq P-1$ in the first data layout. Each of them is codified as the number of occurrences of a symbol a_i inside membrane j. Finally, R_μ is the finite sequence of pairs $[graph, rules]$ which guides the computation.

We will see in the sequel that R_μ is generated algorithmically, by concatenating sequences of pairs $[graph, rules]$[3].

Lemma 4. *Given $N = 2^k$ keys and $P = 2^{k-m}$ membranes, which can store $n = 2^m$ values, $m \geq 1$, after the computation for the data layout $\mathcal{D}_s$ is finished, symbol a_i of membrane j codifies the value corresponding to wire $u \in \{0, \ldots, N-1\}$, where the bit representation of u is $u = j_1 \ldots j_{sm} i_1 \ldots i_m j_{sm+1} \ldots j_{k-m}$. By $j_1 \ldots j_{k-m}$ and by $i_1 \ldots i_m$ we denoted the bit representations of j, and i, respectively.*

Proof. The proof is immediate by Definitions 1, 5 and Lemma 2.

We observe that the remap of values from a data layout to the other can be done in $P+1$ steps. When passing from data layout $\mathcal{D}_{s-1}$ to $\mathcal{D}_s$, with $0 < s \leq \lceil k/m \rceil - 1$, in each step j, $0 \leq j \leq P-1$, membrane j sends its contents along the edges of the communication graph C_s^j. To avoid collisions in the destination membranes,

[3] We denote the empty sequence by λ, and the concatenation of two sequences by ".".

it also performs a rewriting of symbols from a_t to a'_t, for all $t \in \{0, \ldots, n-1\}$. In the last step $P+1$, all auxiliary symbols a'_t will be rewritten back to a_t in all membranes, and the local computation can begin in each membrane.

We give below two algorithms generating the communication graphs C_s^j, and the rules associated to each edge.

$E(C_s^j) \leftarrow \emptyset$;
for $j \leftarrow 0$ **to** $P-1$ **do**
  for $i \leftarrow 0$ **to** $n-1$ **do**
    let j have bit representation $j_1 \cdots j_{sm} j_{sm+1} \cdots j_{k-m}$;
    let i have bit representation $i_1 \cdots i_m$;
    `// the destination membrane of value encoded by` a_i `in`
    `membrane` $j\ z \leftarrow j_1 \cdots j_{sm} i_1 \cdots i_m j_{(s+1)m+1} \cdots j_{k-m}$;
    `// the destination symbol of value encoded by` a_i `in`
    `membrane` $j\ t \leftarrow j_{sm+1} \cdots j_{sm+m}$;
    $E(C_s^j) := E(C_s^j) \cup \{j, z\}$;
    $\text{rules}_{C_s^j}((j,z)) := a_i \rightarrow a'_t$;

Algorithm 2: Generation of the sequence of P communication graphs when passing from data layout $\mathcal{D}_{s-1}$ to $\mathcal{D}_s$, with $0 < s \leq \lceil k/m \rceil - 1$.

for $j \leftarrow 0$ **to** $P-1$ **do**
  $\text{rules-endcomm}((j,j)) := \{a'_i \rightarrow a_i \mid 0 \leq i \leq n-1\}$;

Algorithm 3: Generation of the rules associated to the identity graph which rewrite back the auxiliary symbols a'_t when passing from any data layout $\mathcal{D}_{s-1}$ to $\mathcal{D}_s$, with $0 < s \leq \lceil k/m \rceil - 1$.

We assume that the sequence denoted by $SimOM$ is the sequence of pairs $[graph, rules]$ which simulates the omega network of size n, OM_m ($n = 2^m$). Its construction was indicated in Section 4 and is expressed algorithmically below.

$SimOM \leftarrow \lambda$;
for $t \leftarrow 1$ **to** $m = \log n$ **do**
  forall $p \leftarrow 0$ **to** $P-1$ **in parallel do**
    $\text{rules}_{t,1}((p,p)) \leftarrow \{a_i \rightarrow s_i \mid i = 0, 1, \ldots n-1\}$;
    $\text{rules}_{t,2}((p,p)) \leftarrow \{s_i s_j \rightarrow t_i t_j, s_i \rightarrow t_j, s_j \rightarrow t_j \mid i = 0, 1, \ldots, n-1, i < j = bc_t(i)\}$;
    $\text{rules}_{t,3}((p,p)) \leftarrow \{t_i \rightarrow a_i \mid i = 0, 1, \cdots n-1\}$;
  $SimOM \leftarrow SimOM \cdot [Id, \text{rules}_{t,1}] \cdot [Id, \text{rules}_{t,2}] \cdot [Id, \text{rules}_{t,3}]$;

Algorithm 4: Generation of the sequence $SimOM$ which simulates the omega network of size n.

We can now give the algorithm which generates the whole sequence R_μ guiding the computation.

$R_\mu \leftarrow \lambda$;
for $s \leftarrow 1$ **to** $\lceil k/m \rceil - 1$ **do**
  $R_\mu \leftarrow R_\mu \cdot SimOM$;
  for $j \leftarrow 0$ **to** $P - 1$ **do**
    $R_\mu \leftarrow R_\mu \cdot [C_s^j, \text{rules}_{C_s^j}]$;
  $R_\mu \leftarrow R_\mu \cdot [Id, \text{rules-endcomm}]$;
$R_\mu \leftarrow R_\mu \cdot SimOM$;

Algorithm 5: Generation of the sequence R_μ which guides the computation.

5.1 Computation complexity

Observe that the length of the sequence $SimOM$ is $3 \log n$. As we have $\frac{\log N}{\log n}$ data layouts, and that in each data layout $3 \log n$ steps are needed for $SimOM$ and another $P + 1$ steps are needed for communication, the length of R_μ is $3 \log N + \frac{N \log N}{n \log n}$. A sorting network can be obtained by a serial connection of $\log N$ omega networks, hence our model can sort in time $O(\log^2 N + \frac{N \log^2 N}{n \log n})$. Note that when $n = N$ all computation is local, and the complexity is the best possible, $O(\log^2 N)$. When $n = 2$ the complexity increases to $O(N \log^2 N)$.

References

1. A. Aggarwal, A.K. Chandra, M. Snir, "Communication Complexity of PRAMs", *Theoretical Computer Science,* vol. 71, no.1, pp. 3 - 28, Mar. 1990.
2. M. Ajtai, J. Komlos, and E. Szemeredi, "An $O(N \log N)$ Sorting Network", *Proc. 15th Ann. ACM Symp. Theory of Computing,* pp. 1-9, May 1983.
3. A. Alexandrov, M. Ionescu, K.E. Schauser, C. Scheiman, "LogGP: Incorporating Long Messages into the LogP model", *Journal of parallel and distributed computing,* vol. 44, no. 1, pp. 71-79, 1997.
4. 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, 2005.
5. K.E. Batcher, "Sorting networks and their applications", *Proc. AFIPS Spring Joint Comput. Conf.,* vol. 32, pp. 307-314, Apr. 1968.
6. G. Bilardi, "Merging and Sorting Networks with the Topology of the Omega Network", *IEEE Transactions on Computers,* vol. 38, no. 10, pp. 1396-1403, Oct. 1989.
7. R. Ceterchi, C. Martín-Vide, "Dynamic P Systems", *LNCS*, vol. 2597, pp. 146 - 186, 2003.
8. R. Ceterchi, M.J. Pérez Jiménez, "On two-dimensional mesh networks and their simulation with P systems", *LNCS*, vol. 3365, pp. 259-277, 2005.
9. R. Ceterchi, M.J. Pérez Jiménez, A.I. Tomescu, "Simulating the Bitonic Sort Using P Systems", *G. Eleftherakis et al. (Eds.): WMC8 2007, LNCS,* vol. 4860, pp. 172-192, 2007.

10. D.E. Culler, R.M. Karp, D.A. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, and T. von Eicken, "LogP: Towards a Realistic Model of Parallel Computation", *Proc. Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming,* pp.1-12, May 1993.
11. M. Dowd, Y. Perl, M. Saks, L. Rudolph, "The balanced sorting network", *Proc. Second annual ACM symp. on Principles of distributed computing,* pp. 161-172, 1983.
12. M. Dowd, Y. Perl, M. Saks, L. Rudolph, "The periodic balanced sorting network", *JACM,* vol. 36. no. 4, pp. 738-757, 1989.
13. M.F. Ionescu, "Optimizing Parallel Bitonic Sort", *Tech. Report TRCS96-14,* Dept. of Comp. Sci., Univ. of California, Santa Barbara, July 1996.
14. M.F. Ionescu, K.E. Schauser, "Optimizing parallel bitonic sort", *Proc. 11th Int'l Parallel Processing Symp.,* pp. 303-309, 1997.
15. D.E. Knuth, *The art of computer programming*, volume 3: sorting and searching, second ed. Redwood City, CA: Addison Wesley Longman, 1998.
16. C. Kruskal, L. Rudolph, M. Snir. "A complexity theory of efficient parallel algorithms", *Theoretical Computer Science,* vol.71, no.1, pp. 95 - 132, Mar. 1990.
17. J.D. Lee, K.E. Batcher, "Minimizing Communication in the Bitonic Sort", *IEEE Trans. on Parallel and Distributed Systems,* vol. 11, no. 5, pp. 459-474, May 2000.
18. F. Leighton, "Tight Bounds on the Complexity of Parallel Sorting," *IEEE Trans. Computers,* vol. 34, no. 4, pp. 344-354, Apr. 1985.
19. M.S. Paterson, "Improved Sorting Networks with $O(\log N)$ Depth," *Algorithmica,* vol. 5, pp. 75-92, 1990.
20. A.I. Tomescu, "Optimal Data Layouts for Omega Networks", *manuscript.*