Proc. 35th IEEE-FOCS, 1994, pp. 369–380

Fast and Feasible Periodic Sorting Networks of Constant Depth*†

Mirosław Kutyłowski‡
Krzysztof Loryś‡
Institute of Computer Science
University of Wrocław
PL-51-151 Wrocław, Poland
{mirekk,lorys}@ii.uni.wroc.pl

Brigitte Oesterdiekhoff§
Rolf Wanka§
Dept. of Mathematics & Computer Science
University of Paderborn
D-33095 Paderborn, Germany
{brigitte,wanka}@uni-paderborn.de

Abstract

A periodic comparator network has depth (or period) k, if for every $t > k$, the compare-exchange operations performed at step t are executed between exactly the same registers as at step $t - k$.

We introduce a general method that converts an arbitrary comparator network that sorts n items in time $T(n)$ and that has layout area A into a periodic sorting network of depth 5 that sorts $\Theta(n \cdot T(n))$ items in time $O(T(n) \cdot \log n)$ and has layout area $O(A \cdot T(n))$. This scheme applied to the AKS network yields a depth 5 periodic comparator network that sorts in time $O(\log^2 n)$. More practical networks with runtime $O(\log^3 n)$ can be obtained from Batcher's networks. Developing the techniques for the main result, we improve some previous results: Let us fix a $d \in \mathbb{N}$. Then we can construct a network of depth 3 based on a d-dimensional mesh sorting n items in time $O(n^{1/d} \cdot \log^{O(d)} n)$.

1 Introduction

The Model. Comparator networks can be defined in the following way: We have a digraph $G = (V, E)$, $V = \{v_1, \ldots, v_n\}$, and a fixed sequence $M = (S_1, \ldots, S_T)$ of matchings of G (and $E = S_1 \cup \cdots \cup S_T$). M is called *comparator network* and G its *underlying graph*. Each node stores exactly one item. Every edge $[v_i, v_j] \in E$ represents a communication link between the nodes v_i, v_j and may be used for *compare-exchange operations* only. During such a compare-exchange operation, the items stored at the nodes v_i and v_j are compared; if the item of v_i is greater than the item of v_j, they are switched. It is helpful to assume that the items that are compared are switched also if they are equal. (This convention does not influence the contents of the nodes, but influences the movements of single items and sometimes simplifies an analysis of these movements significantly.)

The computation of M consists of T parallel steps. For $i \leq T$, during the ith step all compare-exchange operations corresponding to the edges in S_i are executed simultaneously (no conflict arises, since S_i is a matching). The *computation time* is defined as the number of parallel steps executed, that is T. The *layout area* of M is the layout area of the graph G. M *sorts* in *time* T, if for each input, after executing the T parallel steps, the item of rank i is stored at node v_i, for $i \leq n$.

A comparator network $M = (S_1, \ldots, S_T)$ is called *periodic* if there is a number c such that $S_i = S_{i+c}$, for all $i \leq T - c$. The sequence of steps $S_1, \ldots, S_c$ is called a *cycle* and c is called the *depth* of M. If M is periodic, then it is completely described by its cycle and the total computation time.

There are at least three objectives in the design of sorting networks: low runtime, small layout area, and ease of implementation. Because of the AT^2-bound, there is a trade-off between speed and area. The ease of implementation, the number of parallel steps, and the communication pattern are crucial for the performance on real machines or VLSI circuits. Involved parallel algorithms that are very fast in theory might be slow on real machines because of constraints that are not taken into consideration during a theoretical analysis. Compare-exchange operations are easy to implement, but it remains to determine which compare-exchange operations have to be performed at a given time and

*partially supported by Volkswagen Stiftung, joint research project "Paralleles Rechnen: Theoretische und experimentelle Untersuchungen zu parallelen Rechnenmodellen und systemnahen Algorithmen" of University of Wrocław and Heinz Nixdorf Institute, University of Paderborn.

†An enlarged version of this paper is available through World Wide Web at `http://www.uni-paderborn.de/fachbereich/AG/agmadh/PapersPostscript/FastPeriodic_TechRep.ps`

‡partially supported by KBN grant 2 1197 91 01

§partially supported by DFG-Forschergruppe "Effiziente Nutzung massiv paralleler Systeme" and by Esprit Basic Research Action Nr. 7141 (ALCOM II)

to provide appropriate communication links between the nodes. This is easier if the algorithm is periodic. The depth should be kept small, since this saves control logic, simplifies timing considerations at the nodes, and reduces the number of communication links used. For these reasons studying periodic sorting networks of constant depths is particularly interesting.

Traditionally, comparator networks are considered as consisting of n vertical parallel wires connected by comparators. Each of n input items moves through the wires from top to bottom. There is exactly one item on each wire. The items may be exchanged between wires by comparators: comparator $[i, j]$ pointing from wire i to wire j places the greater of the items carried by wires i, j on wire j and the smaller one on wire i. The comparators are laid out as wires connecting the vertical wires. They are grouped into layers corresponding to parallel steps. Inside a single layer, a vertical wire can be connected to at most one comparator. Layers of the comparator network are usually laid out separately, layer $i+1$ below layer i, for every i. (However, specially if the network is to be practically implemented, it is reasonable to interleave the layers to save the space.) This model is usually used to illustrate comparator networks. If a network is periodic, then it leads to waste of hardware because of necessity of laying out the same connections repeatedly. However, one can modify the traditional model in the following way: The network consists of a single cycle with wrap-around edges. After executing the last step of a cycle the data are moved through wrap-around edges and fed as input to the network again (see Fig. 1). Because of this interpretation, the period of the algorithm is called depth of the periodic comparator network.

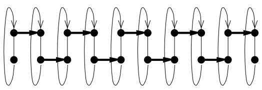

Figure 1: An implementation of Odd-Even Transposition Sort

Previous work. The concept of periodic sorting has been introduced by Schröder [11]. Odd-Even Transposition Sort [5] is the simplest periodic comparator network. It sorts n items in n steps; the underlying graph is a linear array of n nodes, and its depth is 2. The most famous sorting networks such as Batcher's networks [2] and the AKS network [1] are not periodic. However, there has been much effort to construct periodic comparator networks. Dowd et al. [3] proposed the balanced sorting network of depth $\log n$ that sorts n items in time $\log^2 n$. The underlying graph of this network is the $\log n$-dimensional hypercube.

To sort efficiently n items on a 2-dimensional mesh, we cannot generalize Odd-Even Transposition Sort in a straightforward way. The reason is that, even on average, we get runtime $\Theta(n)$ [9], which is no improvement relative to the original Odd-Even Transposition Sort. For the mesh architecture, the Shearsort algorithm has been proposed ([10] and [8]). It is periodic: on a $\sqrt{n} \times \sqrt{n}$-mesh, it performs $\lceil \log \sqrt{n} \rceil + 1$ cycles; each cycle consists of executing Odd-Even Transposition Sort on each row and afterwards on each column of the mesh. Thus, the depth equals $2\sqrt{n}$. An alternative solution for a two-dimensional mesh has been proposed by Schwiegelshohn [12]. He introduced a periodic comparator network of depth 8 that sorts n items in time $O(\sqrt{n} \log n)$. Krammer modified Schwiegelshohn's construction and reduced the depth to 6 and the runtime to $O(\sqrt{n \log n})$ [6].

For $k \in \mathbb{N}$, Kik, Kutyłowski, and Stachowiak built a periodic comparator network of depth $\Theta(k)$ that sorts n items in time $O(k^2 \cdot n^{1/k})$. Unfortunately, their construction is based on expander graphs and is not applicable in practice.

The new results. We improve Schwiegelshohn's and Krammer's constructions by reducing the depth of the network to 3:

Theorem 1.1 *There is a periodic comparator network of depth 3 and layout area of a 2-dimensional mesh that sorts n items in time* $O(\sqrt{n} \cdot \log n)$.

It is easy to see that if the input contains a single column of ones and zeroes otherwise, then our network requires $\Omega(\sqrt{n} \cdot \log n)$ steps. Hence the above time bound is tight. The underlying graph of the network is only slightly different from the two-dimensional mesh, so its VLSI implementation is straightforward. Current experiments show that the algorithm might be interesting for practical applications. We present a complete proof of Theorem 1.1 and simplify the approach of Schwiegelshohn [12] and Krammer [6] (these papers contain informal proofs without details).

Since every periodic sorting network of depth 2 requires at least n steps, Odd-Even Transposition Sort is time-optimal for depth 2, and the depth 3 network obtained in Theorem 1.1 cannot be further improved. We generalize the construction from Theorem 1.1 for multi-dimensional meshes:

Theorem 1.2 *Let a* $d \in \mathbb{N}$ *be fixed. There is a periodic comparator network of depth 3 that sorts n items in time* $O(n^{1/d} \cdot \log^{O(d)} n)$. *The underlying graph of this network is a d-dimensional mesh with some extra edges and has the layout area proportional to the layout area of the mesh.*

Note that the runtimes of these networks have been achieved by the networks of Kik, Kutyłowski, and Stachowiak [4], but the depth of our network is only 3 (which is optimal) and the layout area of is close to optimal. The layout area of the networks from [4] is huge due to the use of expander graphs.

We call a comparator network *monotonic* if for every compare-exchange operation $[v_i, v_j]$ performed holds: $i < j$ (Knuth [5] calls it a standard compare-exchange operation). Since every non-monotonic sorting network can be converted easily into a monotonic one ([5, p. 239]), often only monotonic sorting networks are considered.

The techniques that we develop in the proofs of Theorems 1.1 and 1.2, can be used in a more general construction called *the periodification scheme.* It shows how to convert an arbitrary comparator sorting network into a periodic sorting network of depth 5 with at most slight loss of efficiency.

Theorem 1.3 *Let M be a monotonic (non-periodic) comparator network of runtime $T(n)$ that sorts n items and has layout area A in the traditional model. Then there is a periodic comparator network of depth 5 and layout area $O(A \cdot T(n))$ that sorts $\Theta(n \cdot T(n))$ items in time $O(T(n) \cdot \log n)$.*

Note that for certain functions T (for instance, $T = n^{\epsilon}$), the above transformation even improves the runtime.

The periodification scheme is a general method for transforming sorting comparator networks into another sorting comparator networks preserving their relevant properties and extending them by an additional important property (periodicity). Another construction of this kind (Columnsort) has been proposed by Leighton [7]. His procedure transforms an arbitrary sorting network into a sorting algorithm running on a certain processor network of degree 3.

The periodification scheme of Theorem 1.3 is constructive and generates no big overhead. Hence by applying it to practical networks, we get practical networks. Applying it to Batcher's sorting networks, we get the following corollary (compare [13]):

Corollary 1.4 *There is a practical periodic comparator network of depth 5 and layout area $O(n^2/\log^2 n)$ that sorts n items in time $O(\log^3 n)$.*

Theoretically, the best runtime is achieved by the periodification scheme if we apply it to the AKS network.

Corollary 1.5 *There is a periodic comparator network of depth 5 that sorts n items in time $O(\log^2 n)$.*

The sorting networks that we construct are relatively simple in their design. Nevertheless, proving upper bounds for sorting times requires a quite detailed and sometimes tiresome technical analysis. This is not a surprise, since it is known that even slight modifications in such constructions might result in a disastrous increase of the time required to sort.

Due to the number of technical details, some proofs are only sketched. Since we try to simplify the presentation as much as possible, we do not attempt to optimize the constants hidden by the big O's.

1.1 Preliminaries

First we recall some easy or known properties of comparator networks.

Proposition 1.6 (0-1-Principle [5]) *If a comparator network sorts all inputs consisting solely of 0's and 1's, then it sorts arbitrary inputs.*

Proposition 1.7 *A monotonic comparator network that sorts all inputs must perform compare-exchange operations between each two consecutive nodes according to the ordering of the underlying graph.*

Proposition 1.8 *Let N be a comparator network. Let $\vec{x}, \vec{y} \in \{0,1\}^n$ be inputs for N such that $x_i \le y_i$ for $i \le n$. Then for arbitrary t and $j \le n$, if during a computation on $\vec{x}$, immediately after step t, the jth node of N contains a 1, then the same holds for the computation on $\vec{y}$.*

Proposition 1.9 [11] *If a periodic comparator network M is monotonic and performs all compare-exchange operations of Odd-Even Transposition Sort at every cycle, then M sorts every sequence of n items in at most n cycles.*

2 A periodic sorting network of depth 3

2.1 The construction of the network

The construction of the periodic comparator network M of depth 3 that sorts in time $O(\sqrt{n}\log n)$ is very simple: The underlying graph G_M of M is a two-dimensional mesh in which the nodes are arranged in a way portrayed by Fig. 2. More formally, for even p and odd q, G_M consists of the nodes $P_{i,j}$ for $i \in \{1,\ldots,p\}$, $j \in \{1,\ldots,q\}$ if i is odd, and $i \in \{1,\ldots,p\}$, $j \in \{0,\ldots,q-1\}$ if i is even. For a given i, the ith row of G_M consists of the nodes of the form $P_{i,j}$, and the ith column, called K_i, consists of the

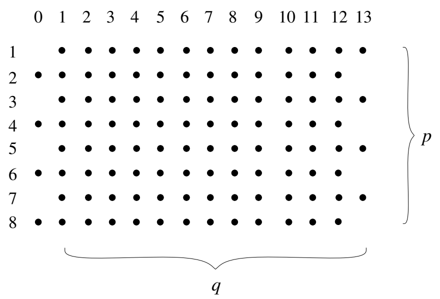

Figure 2: Arrangement of the nodes

nodes of the form $P_{j,i}$. The two leftmost columns and the two rightmost columns are called *border* columns; the remaining columns are called *internal* columns. The nodes of M are ordered according to the row-major ordering: $P_{i,j} \prec P_{i',j'} \iff (i < i') \vee (i = i' \wedge j < j')$.

Each cycle of the algorithm consists of three parallel steps: step A, step B, and step C, performed in this order. The edges corresponding to the compare-exchange operations performed during each of these steps are depicted by Fig. 3 (orientation of the comparators is not depicted since it is assumed that the network is monotonic). It is obvious that the layout area of M is proportional to the layout area of a regular 2-dimensional mesh. Steps A and C are called *horizontal* and step B is called *vertical.* The horizontal steps ensure that, as required by Proposition 1.7, there is a compare-exchange operation between every two consecutive nodes of G_M according to the row-major ordering. Note that the horizontal steps A and B contain long edges connecting the border columns (so called "Schwiegelshohn edges") *every second row*, similarly as in the case of the networks of [12] and [6]. We cannot put all "Schwiegelshohn edges" into a single horizontal step since the resulting algorithm would have average runtime of order n [9]. The crucial phenomenon for the performance of the algorithm on inputs consisting of 0's and 1's is the way in which high towers of 1's (0's) coming to the right (left) border are broken up into two parts residing in two different columns (see Fig. 4). This effect is the basis for Schwiegelshohn's construction [12].

2.2 Runtime analysis

In this section, we show that M sorts in $O(\sqrt{n}\log n)$ steps. In order to simplify our analysis of behavior of M, we assume that $q \geq 13p$, p is even, and $q-1$ is divisible by 4. For practical implementations, the side lengths of the mesh should be chosen very carefully.

By the 0-1-Principle (Proposition 1.6), it suffices to consider only sequences consisting of 0's and 1's.

Lemma 2.1 *There exist constants c and d such that after executing $c \cdot q$ steps, the nodes in the $d \cdot p$ bottom rows of G_M contain only ones or the nodes in the $d \cdot p$ top rows of G_M contain only zeros.*

By Lemma 2.1, it immediately follows that after executing $O(q \cdot \log p)$ cycles, the bottom rows of G_M contain only ones, the top rows of G_M contain only zeros and between them, there are $O(1)$ rows that contain zeros and ones. By Proposition 1.9, the contents of G_M becomes sorted after $O(q)$ additional cycles. Hence, Theorem 1.1 follows.

The rest of this section is devoted to the proof of Lemma 2.1. First we introduce some notions which are closely related to the notions used in [6].

Definition 2.2 Horizontal (vertical) edges *of M are the edges of G_M that connect the nodes of the same row (of the same column). The remaining edges are* slanted edges *used at step B and* "Schwiegelshohn edges" *connecting the left and the right border columns of G_M.*

An item α is called right-running (left-running)*, if during the last horizontal step α was involved in a compare-exchange operation through a horizontal edge and placed at the right (left) end of the edge.*

A column that contains only right-running items is called R-column*, a column that contains only left-running items is called* L-column*.*

Obviously, every internal column is alternately an L-column and an R-column, and the changes occur every horizontal step.

According to our convention for performing compare-exchange operation on equal items, we immediately get the following property:

Fact 2.3 *Each right-running* 1 *remains right-running until it reaches the right border columns. Similarly, each left-running* 0 *remains left-running until it reaches the left border columns. A left-running* 1 *becomes right-running if it is compared with a right-running* 0*. A right-running* 0 *becomes left-running if it is compared with a left-running* 1*.*

Intuitively, we may think that the R-columns move one position to the right at each horizontal step. Indeed, during a horizontal step, all right-running 1's of an R-column K_i move to the column K_{i+1}. Some of the right-running zeroes in K_i also move to K_{i+1}, however some of them meet left-running 1's, stay in K_i and become left-running. Their place in K_{i+1} is taken over by the ones that become right-running at this step. Similarly, L-columns "move left".

Let $w(K_i, t)$ denote the number of ones in column K_i immediately after step t. Horizontal and vertical

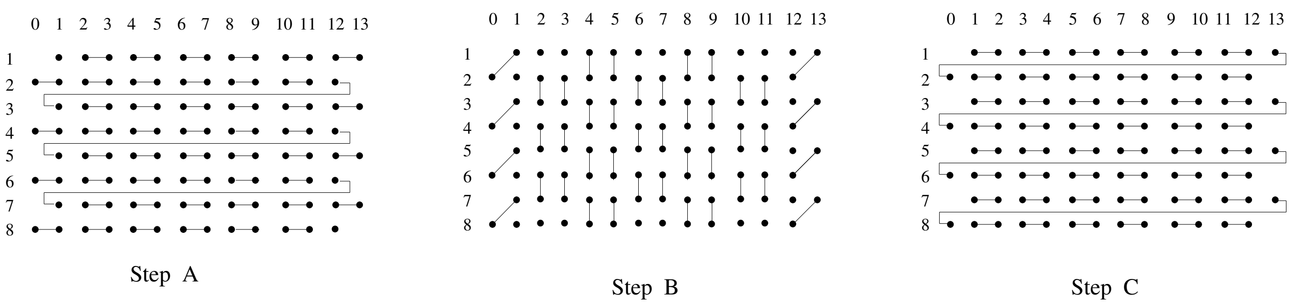

Figure 3: The three parallel steps performed by M during one cycle

steps of M are interleaved. Therefore we introduce the following notation: h_t denotes the tth horizontal step and v_t denotes the tth vertical step of the algorithm. The following facts result obviously from our considerations above:

Fact 2.4 *If K_i is an R-column immediately after step h_t, $i+1 < q-1$, then $w(K_i, h_{t+1}) \leq w(K_i, h_t) \leq w(K_{i+1}, h_{t+1})$. Similarly, if K_i is an L-column immediately after step h_t, $i-1 > 1$, then $w(K_{i-1}, h_{t+1}) \leq w(K_i, h_t) \leq w(K_i, h_{t+1})$.*

By a straightforward induction on k, we obtain the following fact:

Fact 2.5 *If K_i is an R-column immediately after step h_t and $w(K_i, h_t) = x$, then K_{i+k} is an R-column immediately after step h_{t+k} and $w(K_{i+k}, h_{t+k}) \geq x$, provided that $i+k < q-1$. Similarly, if K_j is an L-column immediately after step h_t and $w(K_j, h_t) = x$, then K_{j-k} is an L-column immediately after step h_{t+k} and $w(K_{j-k}, h_{t+k}) \leq x$, provided that $j-k > 1$.*

While an R-column "moves to the right", the vertical steps are used to sort its contents. (Of course, sorting may not be completed because of the new ones that join this column.) R-columns are sorted by Odd-Even Transposition Sort. Note that between consecutive vertical steps, an R-column moves two positions to the right. Hence, to make the things work properly, the vertical edges of the columns at distance 2 must represent *different* steps of Odd-Even Transposition Sort. This is the reason for our somewhat strange design of step B.

If all k lowest nodes of a column contain 1's, then we say that this column has a *foot of height k*. Since p steps of Odd-Even Transposition Sort suffice to sort every input of size p, by Proposition 1.8 we get the following fact:

Fact 2.6 *Suppose that after step v_t, K_i is an R-column and $w(K_i, v_t) \geq x$. If $r \geq p$ and $i + 2r \leq q-1$, then after step v_{t+r}, column K_{i+2r} contains a foot of height x.*

Fact 2.7 *Suppose that K_i is an R-column after step h_t and $w(K_i, h_t) = x$. Then $w(K_j, h_t) \leq x$ for each $j \leq i$ such that K_j is an L-column after step h_t.*

Proof. For each such column K_j consider s and t' that satisfy $j = s - t'$ and $i = (s+1) + t'$. After step $h_{t-t'}$, K_{s+1} is an R-column and K_s is an L-column. Hence they are compared at step $h_{t-t'}$ and therefore $w(K_s, h_{t-t'}) \leq w(K_{s+1}, h_{t-t'})$. Now it suffices to apply Fact 2.5 for $k = t'$. □

Now we discuss what happens at the border columns; as already mentioned, behavior of M in this part is crucial for achieving good runtime. The reader may inspect how a high tower of ones contained in an R-column reaching the right border is broken into two parts by the steps A and B (see Fig. 4).

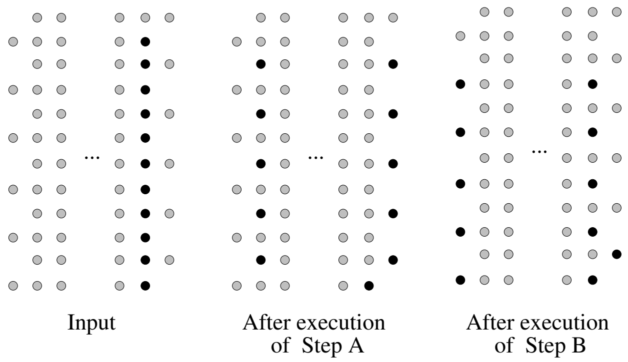

Figure 4: Breaking up a tower of ones at the right border

Fact 2.8 *If step t is a C step and $w(K_1, t) \leq x$, then $w(K_1, t+2) \leq x + \frac{p}{2}$.*

Fact 2.8 shows that if there is a high tower of ones reaching the right border and simultaneously an L-column with only few ones reaching the left border, then after a few steps, the number of ones in these columns will be balanced to some extent.

Proof of Fact 2.8. Let $x < \frac{p}{2}$, since otherwise the fact is trivial. For each even i, it is easy to check for each even i that if after step t the node $P_{i,1}$ contains a zero, then after subsequent step B (i.e., step $t+2$), node $P_{i-1,1}$ contains a zero. After step t, there are at least $p-x$ zeros in the first column; at least $\frac{p}{2}-x$ of them are stored at the nodes $P_{i,1}$, for i even. Hence, there are at least $\frac{p}{2}-x$ zeros in column K_1 after step $t+2$, that is, at most $\frac{p}{2}+x$ ones. □

In the proof of Lemma 2.1, we shall use some counting arguments. Since G_M is symmetric, we may always reverse the roles of 0's and 1's. Hence, we may assume that the input contains at least $\frac{1}{2}pq$ ones.

Let $g = q-4p$. We shall prove that after $O(q)$ steps, $O(p)$ bottom rows of G_M will be filled with 1's. To fill this region with 1's, we use the 1's that occur in K_g, when K_g is an R-column after step h_g. Therefore, we have to show that K_g *always* contains enough ones.

Lemma 2.9 *There is a constant c_1 such that if the column K_g is an R-column after horizontal step h_t and $t \geq g$, then $w(K_g, h_t) \geq c_1 \cdot p$.*

Proof. Let x be the number of ones in the R-column K_g after step h_t. We shall estimate the total number of ones in G_M by an expression that depends on x. We shall see that the inequality $x \geq c_1 \cdot p$ follows from the assumption that there are at least $\frac{1}{2}pq$ ones in G_M.

It is crucial how many ones are on the left side of K_g. Let $t_0 = t-(g-1)$. For $i \leq g-1$, let S_i consist of the columns 2 through $i+1$ with their contents immediately after step h_{t_0+i}. Note that the rightmost column of S_i is the R-column that reaches K_g at step h_t. Hence, by Fact 2.5, the rightmost column of S_i contains at most x ones. So by Fact 2.7, each L-column inside S_i contains at most x ones. In particular, for $t > h_{t_0}$, $w(K_1, t) \leq x$ if t is a C step. Hence by Fact 2.8, $w(K_1, t+2) \leq x + \frac{p}{2}$ (note that $t+2$ is the next B step after step t).

Let $w(S_i)$ denote the total number of ones in S_i. In order to estimate $w(S_{g-1})$ we consider how $w(S_i)$ grows with i. A vertical step that might be executed between steps h_{t_0+i} and h_{t_0+i+1} does not influence the number of ones in the columns 2 through $i+2$. So we only have to consider what happens during step h_{t_0+i+1}. By the construction of M, the total number of ones in the columns 2 through i does not change during step h_{t_0+i+1}, if this is an A step. If step h_{t_0+i+1} is a C step, then at this step the number of ones in the columns 3 through i does not change. S_{i+1} contains R-column K_{i+2} that is not in S_i. Therefore to estimate $w(S_{i+1}) - w(S_i)$, it suffices to consider the contents of the columns K_2, K_{i+1}, K_{i+2} only. By Fact 2.4, $w(K_{i+1}, h_{t_0+i+1}) \leq w(K_{i+1}, h_{t_0+i})$. On the other hand, $w(K_{i+2}, h_{t_0+i+1}) \leq x$. So at the right border, the number of ones increases by at most x. It follows that if step h_{t_0+i+1} is an A step, then

$$w(S_{i+1}) \leq w(S_i) + x. \tag{1}$$

Now we assume that step h_{t_0+i+1} is a C step. We have seen that $w(K_1, h_{t_0+i+1}-1) \leq x + \frac{1}{2}p$. Hence the number of ones in K_2 may increase during step C by at most $x + \frac{1}{2}p$. So

$$w(S_{i+1}) \leq w(S_i) + 2x + \tfrac{1}{2}p. \tag{2}$$

By inequalities (1) and (2), it is easy to derive that $w(S_{g-1}) \leq (g-1)(\frac{1}{4}p + \frac{3}{2}x)$. Taking into account at most $4p^2 - \frac{1}{2}p$ ones in the nodes on the right side of K_g and at most $\frac{3}{2}p$ ones in columns K_0 and K_1, we see that the total number of ones is less than $\frac{1}{4}pq + \frac{3}{2}qx + 3p^2$. We have assumed that the total number of ones is at least $\frac{1}{2}pq$. Since $q \geq 13p$, it follows that $x \geq c_1 \cdot p$ for some constant c_1. □

By Fact 2.6 and Lemma 2.9, we get immediately:

Fact 2.10 *There exists a constant c_1 such that at any step h_t, for $t \geq q$, each R-column K_j for $j \geq q-2p$ contains a foot of height $c_1 \cdot p$.*

Fact 2.10 guarantees that after step h_q, there are high feet coming to the right border all the time. We shall see that these feet alone suffice to fill the bottom part of G_M with ones in reasonable time.

We say that a word $w = w_1 w_2 \ldots w_p \in \{0, 1, *\}^p$ describes column K_j at some instant if for $i \leq p$ node $P_{i,j}$ stores number w_i whenever $w_i \in \{0, 1\}$. By Fact 2.10, the R-columns coming to the right border after step h_q have feet of height $c_1 p$. The ones of each such a foot are split into two groups: half of the ones move to the left border through the "Schwiegelshohn edges" and begin their movement to the right; the remaining ones are "reflected" at the right border and become left-running. By Fact 2.10, they remain left-running at least up to the moment when they reach column K_{q-2p}. In the meantime, the mentioned groups of left- and right-running ones are sorted inside the columns by the vertical steps. Let c_2 be a constant such that $c_2 p \leq \lfloor \frac{c_1 p - 1}{2} \rfloor$. For $j \in \mathbb{N}$, let $\bar{\jmath} = \lfloor (j+1)/2 \rfloor$ and $\tilde{\jmath} = \lfloor (j-1)/2 \rfloor$. A more detailed but easy inspection shows the following fact (see also Fig. 5):

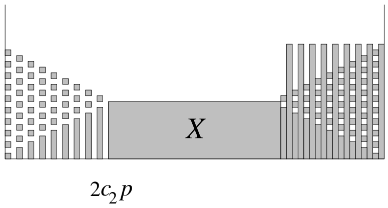

Figure 5: The situation after step h_{3q}; only some of the ones are depicted: those indicated by Fact 2.11 on the left side, those of X in the middle, and the feet of the R-columns and the reflected ones on the right side

Fact 2.11 *Let $2 \leq i \leq c_1p-2$. Then for each $j \geq 0$, immediately after step h_{q+i+2j}, the R-column K_i is described by the word $(*)^{p-2c_2p+\bar{i}}(1*)^{c_2p-\bar{i}}(1)^{\bar{i}}$.*
Let $2 \leq i \leq c_1p$. Then for each $j \geq 0$ immediately after step h_{q+i+2j} the L-column K_{q-i} is described by the word $(*)^{p-2c_2p+\tilde{i}}(1*)^{c_2p-\tilde{i}}(1)^{\tilde{i}}$.

By Fact 2.11 the left-running ones arriving at column K_{q-c_1p} after step h_{q+c_1p} form feet of height c_2p. Fact 2.11 also implies that after step h_{2q} every R-column K_j for $j \geq c_1p-2$ has a foot of height c_2p. Then the feet of height c_2p of L-columns appearing at column K_{q-c_1p} can move further to the left up to the column K_{c_1p-2}. These feet fill the middle-bottom part of the network in $O(q)$ steps:

Fact 2.12 *Starting from step h_{3q}, between the columns K_{2c_2p} and K_{q-2c_2p} every column has a foot of height c_2p.*

The feet of height c_2p located in the middle of G_M form "an area of high feet" that will be called region X. To finish the proof of Lemma 2.1 we have to show that X quickly extends itself to the both borders during the next $O(p)$ steps.

Lemma 2.13 *In $O(p)$ steps after step h_{3q}, region X reaches the left border of the network, i.e., each column K_j, for $j \leq 2c_2p$, contains a foot of height c_2p.*

Proof. We show that every $O(1)$ cycles, a new column joins X on the left side. Let l be the maximal number such that $l \leq 2c_2p$ and K_l has not got a foot of height c_2p at the considered moment of the computation. First we assume that $l \geq 4$.

Case 1: *l is odd.* By Fact 2.11 after subsequent step B, the R-columns K_l and K_{l-2} are described by the words $w = (*)^{p-2c_2p+\bar{l}}(1*)^{c_2p-\bar{l}}(1)^{\bar{l}}$ and $u = (*)^{p-2c_2p+\bar{l}-1}(1*)^{c_2p-\bar{l}+1}(1)^{\bar{l}-1}$, respectively. The next step C does not change the contents of c_2p lowest nodes of K_l, since all corresponding nodes of column K_{l+1} contain ones. Since the ones from column K_{l-2} are moved one position to the right, the words u and w describe now the columns K_{l-1} and K_l, respectively. Then for each $i < c_2p$, either $P_{i,l-1}$ or $P_{i,l}$ contains a 1. These two nodes are compared during the next step A. Hence a foot of height c_2p is formed.

Case 2: *l is even.* After subsequent step C, the columns K_l and K_{l-2} are R-columns and are described by the words $w = (*)^{p-2c_2p+\bar{l}}(1*)^{c_2p-\bar{l}}(1)^{\bar{l}}$ and $u = (*)^{p-2c_2p+\bar{l}-1}(1*)^{c_2p-\bar{l}+1}(1)^{\bar{l}-1}$, respectively. Similarly as in Case 1, after step A, these words describe the columns K_l and K_{l-1}, respectively. Then step B transforms the columns so that now they are described by $w' = (*)^{p-2c_2p+\bar{l}+1}(1*)^{c_2p-\bar{l}-1}(1)^{\bar{l}+1}$ and $u' = (*)^{p-2c_2p+\bar{l}}(1*)^{c_2p-\bar{l}}(1)^{\bar{l}}$, respectively. Then for each $i < c_2p$, at least one of the nodes $P_{i,l-1}$ and $P_{i,l}$ contains a 1. Hence after the next step C, column K_l has a foot of height c_2p.

Similar considerations show that $O(1)$ cycles suffice to include the columns K_0, K_1, K_2 and K_3 to region X provided that K_4 is already in X. □

Once X reaches the left border, every left-running column emerging at the right border contains a foot of height c_2p (the feet are no longer broken at the right border, since there are 1's in K_0). Such a left-running foot moves to the left until it meets X. Then it extends X by one column. It follows that X reaches the right border in $O(p)$ steps.

3 The periodification scheme

An important property of the networks described in Section 2 is that R-columns and L-columns are being sorted on their way to the right and left border, respectively. Now we show that if we replace Odd-Even Transposition Sort that has been previously used by other algorithms, then we may obtain much faster networks, as claimed in Theorem 1.3. The main idea of the algorithm remains the same as in Section 2, but there is a new technical problem that causes deep modifications of the design. Namely, if we proceed as in Section 2 and prove that there is a region X consisting of middle columns of the network in which every column contains a foot of height $O(p)$, then we still have to prove that X quickly extends itself to the right and left borders of the network. The problem is that the proof of this phenomenon in Section 2 very heavily depends on the properties of Odd-Even Transposition Sort. In Subsection 3.1, we design a periodic network of depth 6 that avoids this problem no matter which (non-periodic) algorithm has been used. In Subsection 3.2, we discuss some features of the construction. In Subsection 3.3, we prove an upper bound of the runtime of the algorithm. Finally in Subsection 3.4, we explain how to reduce the depth to 5.

3.1 Construction of the networks

Let N be a (non-periodic) monotonic sorting network, which sorts p items in T steps. The nodes of the network $\mathcal{P}_N$ that we construct are arranged in a $p \times q$-rectangle, where an even q, $q = \Theta(T)$, is chosen appropriately. As before, the nodes are ordered according to the row-major ordering and $P_{i,j}$ denotes the node from the ith row and the jth column. $\mathcal{P}_N$ performs cyclically steps A, B, D, A, C, E. The steps A, B, and C called *horizontal* are depicted by Fig. 6. With

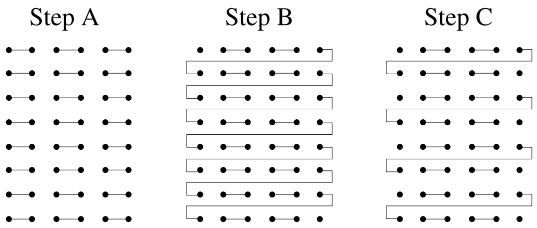

Figure 6: The horizontal steps

few exceptions, the steps D and E, called *vertical*, perform only vertical comparisons, i.e. comparisons along the edges inside the same column. These edges depend on N. To determine them, we define embeddings of N into $\mathcal{P}_N$. Intuitively speaking, in order to embed N into columns $i_1, i_2, \ldots, i_T$, we draw N in the traditional way with its "wires" laid horizontally: for $j \leq T$ we take all comparators from the level j of N and put them into column i_j (see Fig. 7).

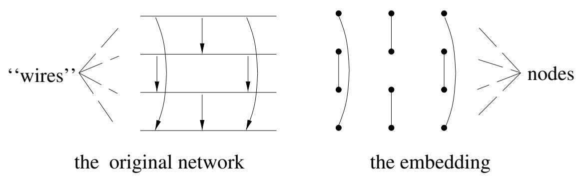

Figure 7: Embedding of a network

Definition 3.1 *We say that N is* embedded *into columns $i_1, i_2, \ldots, i_T$, if for every $j = 1, \ldots, T$ and $1 \leq k_1 \leq k_2 \leq p$, if the nodes k_1 and k_2 of N are compared at step j (during the non-periodic computation of N), then there is an edge between the nodes P_{k_1,i_j} and P_{k_2,i_j}.*

The network N is embedded into $\mathcal{P}_N$ into many different places. We divide the nodes of $\mathcal{P}_N$ into groups shown by Fig. 8. The groups X_L^i, X_R^i for $i = 0, 1, 2$

X_L^2	Z_L	X_L^1	X_L^0	Y	X_R^0	X_R^1	Z_R	X_R^2

Figure 8: Partitioning of the nodes of $\mathcal{P}_N$

are all identical. Each of them consists of $2T$ columns and contains two embedded copies of N. One copy is used for sorting R-columns and the second one for sorting L-columns. If we index the columns of such a group from left to right by the numbers $1, 2, \ldots, 2T$, then the first copy of N is embedded into the columns $1, 3, \ldots, 2T-1$ and the second one is embedded into the columns $2T, 2T-2, \ldots, 2$. Z_L and Z_R consist of four columns each and contain edges depicted by Fig. 9. Group Y has width $\Theta(T)$ and may contain arbitrary vertical edges. (We do not need such edges for our runtime analysis, however they may improve real performance of the network.)

Steps D and E both perform all comparisons corresponding to the vertical edges introduced by embeddings of N. Additionally, step D performs comparisons corresponding to the edges inside Z_L and Z_R presented at Fig. 9. Steps D and E may perform arbitrary vertical comparisons inside Y.

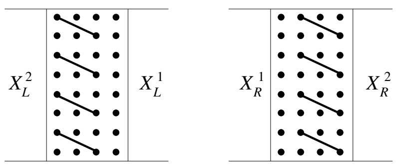

Figure 9: The edges inside Z_L and Z_R

3.2 Remarks on the construction

There are some unexpected elements in the design of $\mathcal{P}_N$. The first one is the shape of step B. It contains **all** Schwiegelshohn edges. The idea is to let every second R-column move from the right to the left border. These R-columns are used once there is a "region X" of high feet in the middle of $\mathcal{P}_N$. Each such R-column with a high foot meeting X extends X by one column to the left.

Since now there is only one step that performs breaking columns at the borders, there is no need for ragged sides of the network and therefore the nodes form an ordinary rectangle.

There is an additional problem caused by step B. In the situation when every R-column going to the right border of $\mathcal{P}_N$ contains a foot of height k, we can no longer guarantee that every L-column generated contains at least $k/2$ ones. This is guaranteed only for **every second** L-column, because of step B. However, the following may be observed: Consider an L-column that contains at least $k/2$ ones in k lowest nodes when it is generated at the right border. After moving through X_R^2 it contains a foot of height $k/2$. Then this column moves to Z_R. Everything is synchronized so that this L-column is the second leftmost column in Z_R when step D is executed. During step D the foot of this column of height $k/2$ is split into two parts – every second 1 is sent to the next L-column (for which we had no guarantee about a number of ones contained). The edges inside Z_R and the sequence of moves are designed so that the slanted edges inside Z_R do not affect the R-columns. Z_L acts similarly on the R-columns on the left side of $\mathcal{P}_N$.

The role of Y is only to make $\mathcal{P}_N$ wide enough in order to apply counting arguments (as in Lemma 2.9).

3.3 Analysis of the algorithm

Lemma 3.2 *There exist constants c and d such that after executing $c \cdot q$ cycles the bottom $d \cdot p$ rows contain only ones or the top $d \cdot p$ rows contain only zeroes.*

Since M is monotonic, if we remove a number of bottom (top) rows containing 1's (0's) only (and the edges pointing to these rows), then we get a network $\mathcal{P}_{N'}$ for some other sorting network N'. Hence by Lemma 3.2 and Proposition 1.9, $\mathcal{P}_N$ sorts in time $O(q \log p) = O(T \log p)$.

Sketch of the proof of Lemma 3.2. Since the network is symmetric, we may assume that at least half of the nodes contain ones. We have to follow almost the same sequence of technical facts as in Section 2. We only sketch the necessary changes.

The Facts 2.3, 2.4 and 2.7 remain valid for $\mathcal{P}_N$ and no changes in the proofs are necessary. Fact 2.5 holds if K_{i+k} is on the left side of Z_R and K_{j-k} is on the right side of Z_L (this assumption is necessary to disregard the slanted edges inside Z_L and Z_R). Fact 2.6 has to be modified in the following way:

Fact 3.3 *Let K_i be the leftmost column of X_Z^i, $i \in \{0,1,2\}$, $Z \in \{L,R\}$. Suppose that after step v_t, K_i is an R-column and $w(K_i, v_t) \geq x$. If $r \geq T$ and $i + 2r \leq q-1$, then after step v_{t+r}, column K_{i+2r} contains a foot of height x.*

Fact 3.3 follows immediately from the construction of $\mathcal{P}_N$ and Proposition 1.8. Fact 2.8 is also valid, again following immediately from the construction. Let K_g be the first column of X_R^0. The proof of Lemma 2.9 can be repeated with slight changes: now only every second R-column is generated during step C. During step B the number of ones at the left border of S_i may increase by p. Therefore the estimation of the number of ones in S_i must be changed accordingly.

By Lemma 2.9 and Fact 3.3 every R-column arriving on the left side of X_R^1 after some step $t_0 = O(q)$ contains a foot of height at least $k = \Omega(p)$. These feet are not destroyed while they move to the right through Z_R, because the slanted edges inside Z_R are applied to L-columns only.

At the right border, R-columns are affected alternately by steps B and C. Ones of every second R-column are entirely moved to the left border (except for the one in the lowest row). The remaining R-columns are split and half of their foot elements are reflected, i.e., they become left-running 1's at the right border. Thus from a certain step t_1 ($t_1 = O(q)$), every second L-column in X_R^2 will have at least $\frac{k}{2}$ ones in its k lowest nodes. These ones form a foot of height $\frac{k}{2}$ when they arrive in Z_R. Moreover, each of these feet, while moving through Z_R, is located in the second leftmost column when step D is executed. Then such a foot is split into two parts, and half of the ones is sent to the next L-column which could contain no 1 at all. This splitting the feet guarantees that after $O(q)$ steps each L-column arriving in X_R^1 has at least $\frac{k}{4}$ ones. Hence every L-column arriving at the left border of X_R^1 has a foot of height $\frac{k}{4}$.

Every R-column arising at the left border after step t_1 contains at least $\frac{k}{2}$ ones in k lowest nodes. This R-column contains a foot of height at least $\frac{k}{2}$ when it arrives at the right border of X_L^2. Therefore after $O(q)$ steps, each R-column to the right of X_L^2 has a foot of height $\frac{k}{2}$. Then the ones from the lowest $\frac{k}{4}$ nodes of L-columns move freely to the left as long as they are outside X_L^2. Thus after $O(q)$ steps each column inside $Z_L X_L^1 X_L^0 Y X_R^0$ has a foot of height $\frac{k}{4}$. Afterwards the "area of high feet" expands itself to the left by at least one column during each cycle. Indeed, the R-columns arising at the left border after step B have feet of height k. They move to the right; every time such a column reaches the "area of high feet" this area expands itself by one columns to the left. As soon as the whole X_L^2 is incorporated into the area of high feet, each L-column arising at the right border has a foot of height $\frac{k}{4}$. These L-columns move to the left and expand the area of high feet to the right by one column every horizontal step. Thus after $O(q)$ cycles all columns have a foot of height $\frac{k}{4}$. □ Lemma 3.2

3.4 Reducing the depth to 5

We substitute two vertical steps by one. Now each cycle consists of steps A,B,F,A,C, where F is the only vertical step. Step F performs the same comparisons inside Z_L and Z_R as step E. Since step F is executed every fourth horizontal step, we must modify connections inside X_L^i and X_R^i $(i = 0, 1, 2)$ so that all R-(L-)columns are transformed by consecutive steps of network N. Each X_L^i and X_R^i consists of $4T$ columns and contains 4 embedded copies of N. Two of them are embedded into columns $1, 5, \ldots, 4T-3$ and $3, 7, \ldots, 4T-1$ and sort R-columns. The other two copies are embedded into columns $4T-2, 4T-6, \ldots, 2$ and $4T, 4T-4, \ldots, 4$ and sort L-columns. Note that still every *second* L-column is split while moving through Z_R. Indeed, there are two L-columns in Z_R, when step F is executed: the second leftmost column (which is to be split) and the rightmost one. Since Z_R consists of 4 columns and there are 4 horizontal steps in a cycle, step F must be executed exactly once when an L-column moves through Z_R.

4 Periodic networks based on multi-dimensional meshes

In this section, we show how the construction described in Section 2 can be directly extended to multi-dimensional meshes. Due to the large number of technical details, we confine ourselves to a sketchy discussion of the topic.

The main idea is to connect several copies of the comparator network based on d-dimensional meshes in such a way that, in the resulting $(d+1)$-dimensional mesh, they play a similar role as the columns in the 2-dimensional case. The computation consists of three steps performed repeatedly; two of them correspond to the horizontal steps of the 2-dimensional case; the third step is used to sort the "columns", i.e. d-dimensional submeshes. Since the third step does not compare nodes lying in two different d-dimensional submeshes (except for the leftmost and rightmost submeshes), it makes sense to speak about L- and R-"columns". Each "column" implements a single step of the sorting algorithm on the d-dimensional mesh. The choice of steps implemented by particular "columns" is done so that when an L- or R-"column" moves through the network, then the consecutive steps of the sorting algorithm on the d-dimensional mesh are applied. This general idea can be implemented provided that we can deal with the following two problems. To prove a counterpart of Lemma 2.13 we must design special edges in the rightmost and the leftmost "columns" that cause the region X of "high feet" to expand itself to the left and right borders. In Section 3, we solved this problem by letting every second R-column move unbroken to the left border. However, this caused increase of the depth of the network to 5. The second problem is that when we reduce the number of "dirty rows" containing 0's and 1's to a constant, then the "dirty rows" should form a d-dimensional mesh of the same structure for which the sorting time can be derived.

In order to simplify our considerations, we describe only the case of $d = 3$ (however, it does not reveal all technicalities required for the general case). We place the nodes of M, the network being constructed, in a $r \times (p+1) \times (q+1)$-mesh and index them by triples of integers (k, j, i) with $1 \le k \le r$, $0 \le j \le p$ and $0 \le i \le q$. On the side walls, the front wall and the back wall, only a half of the nodes are used by the network M (see Fig. 10).

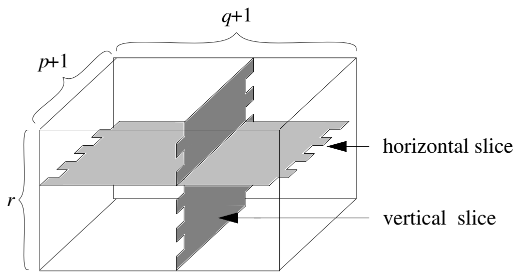

Figure 10: Network M

A *vertical i-slice* is the set of all nodes of M with the last coordinate equal i (see Fig. 10). A *horizontal k-slice* is the set of all nodes of M with the first coordinate equal k. The nodes of M are placed so that the horizontal slices look like depicted by Fig. 11. The

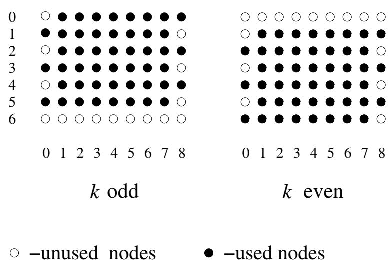

Figure 11: Nodes of a horizontal slice k

nodes are ordered lexicographically with the first coordinate most important, i.e. all nodes of the horizontal $(k-1)$-slice precede the nodes of the horizontal k-slice, and inside a horizontal slice the nodes are ordered as in the 2-dimensional case. We assume that q and p are odd, r is even, and $q = \Omega(p \log r)$, $p = \Omega(r)$ are appropriately chosen.

The algorithm performs parallel steps G, H and J periodically. It is convenient to consider a 2-dimensional layout of the nodes of M obtained by putting horizontal slices into a $pr \times (q+1)$ rectangle, so that the node $P_{k,j,i}$ is placed at the position $((k-1)p + j, i)$ (see Fig. 12). This layout and ordering of the nodes

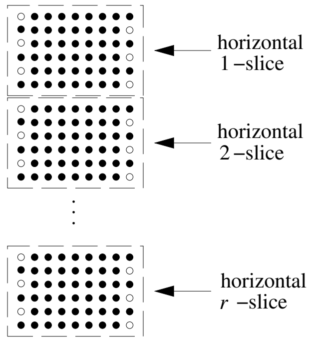

Figure 12: Representing M as a 2-dimensional mesh

is the same as for the 2-dimensional mesh. Steps G and J correspond to steps A and C, respectively, of the algorithm on the 2-dimensional mesh. Step H is more complex and it implements all comparisons performed by the algorithm working on d-dimensional meshes on vertical slices. We consider three groups of vertical slices, called $G_d, G_{\text{easy}}, G_{\text{border}}$ playing different roles during the computation:

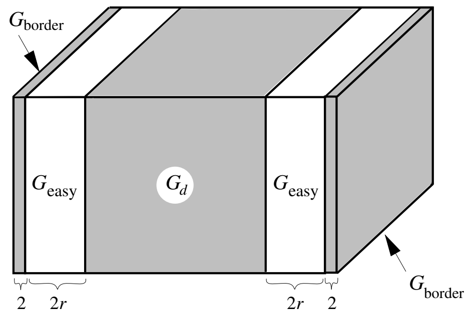

Figure 13: Classification of vertical slices

- G_{border} consists of the two leftmost and two rightmost slices; these slices provide all slanted edges for the 2-dimensional layout that would be used by the algorithm on the 2-dimensional mesh during step B;
- G_{easy} consists of vertical slices $2, \ldots, 2r+2$ and $q - 2r - 2, \ldots, q - 2$; the arrangement of edges inside these slices is shown by Fig. 14; for $i \equiv 1$ or $2 \bmod 4$ the i-slices have edges of type I, the remaining slices have edges of type II;
- G_d consists of the remaining vertical slices; the edges inside these slices are arranged as for steps A, B or C (see Fig. 3); for $i \equiv 0$ or $1 \bmod 6$ the arrangement is as for step A, for $i \equiv 2$ or $3 \bmod 6$ it corresponds to step B, and for $i \equiv 4$ or $5 \bmod 6$ it corresponds to step C.

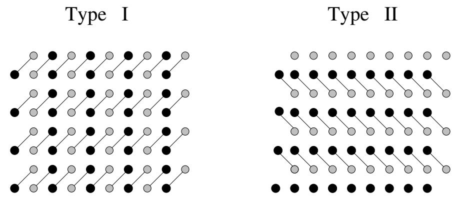

Figure 14: The edges inside vertical slices of G_{easy}. The black nodes represent 1's from a broken R-slice before and after applying the first slice of type I.

The analysis of the sorting time is similar to the analysis for the 2-dimensional mesh. Assuming that at least half of the nodes contain 1's, one can prove that after $O(q)$ steps, each of the $c_1 p \log r$ rightmost R-slices of G_d contains $c_2 pr$ ones, for some constants c_1, c_2, just like in Section 2. While moving to the right, the R-slices are transformed by step H. We choose c_1 so that $c_1 p \log r \geq 2T$, where T is the time required to sort a $r \times p$-mesh. Therefore, each R-"column" entering G_{easy} has at least $c_2 r$ lowest rows filled up with ones. After reflecting at the right border, each newly created L-slice contains 1's in every second node inside $c_2 r$ bottom rows. The comparators of G_{easy} have been specially designed to deal with such sequences: after applying a slice of type I for the first time, every second row up to row $c_2 r$ contains 1's only. Applying consecutive slices of types II and I makes the rows with 1's fall down (see Fig. 14). It is not hard to see that these rows fill bottom horizontal slices just as 1's filled bottom rows of the 2-dimensional mesh in Section 2. Continuing as in Section 2, we can finally prove that after $O(q)$ steps at least $O(r)$ horizontal slices contain no zeros (provided that at least half of the input items are 1's). It follows that after $O(q \log r)$ steps, bottom horizontal slices contain 1's only, top horizontal slices contain 0's only, and between them there is a constant number (say c_3) of horizontal slices that may contain both 0's and 1's. We call these slices *dirty*.

Now we look at the 2-dimensional layout of M restricted to the dirty slices. It is almost the 2-dimensional mesh considered in Section 2. The difference is that $2r+2$ leftmost and rightmost columns do not contain connections which correspond to Odd-Even Transposition Sort. Additionally, there are columns in the middle of the mesh that contain comparators implementing vertical comparisons in vertical slices. However, using Proposition 1.9, one can rewrite the proof of Section 2 to deal with such a mesh and prove that the time required for sorting dirty slices is $O(q\log(c_3 p))$. It follows that the total time required for sorting is $O(q\log(pr))$. For $q=\Theta(n^{1/3}\log^{2/3})$, $p,r=\Theta(n^{1/3}/\log^{1/3}n)$, we get $O(n^{1/3}\log^{5/3}n)$.

Problem

The best sorting time that we have obtained using periodic networks of constant depth is $O(\log^2 n)$. There is no known lower bound for sorting time on periodic comparator networks of constant depth other than $\Omega(\log n)$, which holds for arbitrary comparator networks. It would be interesting to close this gap.

Acknowledgment

We are grateful to Friedhelm Meyer auf der Heide for presenting us the problem of periodic sorting with constant depth networks. He also pointed to many ideas that finally have been implemented by our algorithms. We thank Endré Szemerédi for remarks concerning an early version of the paper.

References

[1] M. Ajtai, J. Komlós, and E. Szemerédi. Sorting in $c\cdot\log n$ parallel steps. *Combinatorica*, 3:1–19, 1983.

[2] Kenneth E. Batcher. Sorting networks and their applications. In *AFIPS Conf. Proc. 32*, pages 307–314, 1968.

[3] Martin Dowd, Yehoshua Perl, Michael Saks, and Larry Rudolph. The periodic balanced sorting network. *Journal of the ACM*, 36:738–757, 1989.

[4] Marcin Kik, Mirosław Kutyłowski, and Grzegorz Stachowiak. Periodic constant depth sorting networks. In *Proc. of the 11th STACS*, pages 201–212, 1994.

[5] Donald E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching.* Addison-Wesley, Reading, MA, 1973.

[6] Josef Krammer. *Lösung von Datentransportproblemen in integrierten Schaltungen.* Dissertation, TU München, 1991.

[7] Tom Leighton. Tight bounds on the complexity of parallel sorting. *IEEE Transactions on Computers*, 34:344–354, 1985.

[8] Kazuhiro Sado and Yoshihide Igarashi. Some parallel sorts on a mesh-connected processor array and their time efficiency. *Journal of Parallel and Distributed Computing*, 3:398–410, 1986.

[9] Serap A. Savari. Average case analysis of five two-dimensional bubble sorting algorithms. In *Proc. of the 5th ACM-SPAA*, pages 336–345, 1993.

[10] Isaac D. Scherson, Sandeep Sen, and Adi Shamir. Shear-sort: A true two-dimensional sorting technique for VLSI networks. In *Proc. of the IEEE International Conference on Parallel Processing*, pages 903–908, 1986.

[11] Heiko Schröder. Partition sorts for VLSI. In I. Kupka, editor, *Proc. of the 13th GI-Jahrestagung*, volume 73 of *Informatikfachberichte*, pages 101–116. Springer-Verlag, 1983.

[12] Uwe Schwiegelshohn. A shortperiodic two-dimensional systolic sorting algorithm. In *International Conference on Systolic Arrays*, pages 257–264, 1988.

[13] Clark D. Thompson. The VLSI complexity of sorting. *IEEE Transactions on Computers*, C-32(12):1171–1184, 1983.