Periodification Scheme: Constructing Sorting Networks with Constant Period

MIROSŁAW KUTYŁOWSKI AND KRZYSZTOF LORYŚ

University of Wrocław, Wrocław, Poland

AND

BRIGITTE OESTERDIEKHOFF AND ROLF WANKA

Paderborn University, Paderborn, Germany

Abstract. We consider comparator networks M that are used repeatedly: while the output produced by M is not sorted, it is fed again into M. Sorting algorithms working in this way are called *periodic*. The number of parallel steps performed during a single run of M is called its *period*, the sorting *time* of M is the total number of parallel steps that are necessary to sort in the worst case. Periodic sorting networks have the advantage that they need little hardware (control logic, wiring, area) and that they are adaptive. We are interested in comparator networks of a constant period, due to their potential applications in hardware design.

Previously, very little was known on such networks. The fastest solutions required time $O(n^\epsilon)$, where the depth was roughly $1/\epsilon$. We introduce a general method called *periodification scheme* that converts automatically an arbitrary sorting network that sorts n items in time $T(n)$ and that has layout area $A(n)$ into a sorting network that has period 5, sorts $\Theta(n \cdot T(n))$ items in time $O(T(n) \cdot \log n)$, and has layout area $O(A(n) \cdot T(n))$. In particular, applying this scheme to Batcher's algorithms, we get practical period 5 comparator networks that sort in time $O(\log^3 n)$. For theoretical interest, one may use the AKS network resulting in a period 5 comparator network with runtime $O(\log^2 n)$.

The authors were 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.

This is the full version of the results concerning the "periodification scheme" from the paper "Fast and Feasible Periodic Sorting Networks of Constant Depth." *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society Press, Los Alamitos, Calif., pp. 369–380. Some of the results presented here became a part of Ph.D. Dissertation of the third author [Oesterdiekhoff 1997].

Authors' addresses: M. Kutyłowski and K. Loryś, Institute of Computer Science, University of Wrocław, PL-51-151 Wroclaw, Poland, e-mail: {mirekk,lorys}@ii.uni.wroc.pl; B. Oesterdiekhoff and R. Wanka, Heinz Nixdorf Institute and Department of Mathematics and Computer Science, Paderborn University, D-33095 Paderborn, Germany, e-mail: {brigitte,wanka}@uni-paderborn.de.

Categories and Subject Descriptors: C.5.4 [**Computer System Implementation**]: VLSI Systems; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*sorting and searching*

General Terms: Algorithms, Design

Additional Key Words and Phrases: Comparator Network

1. *Introduction*

1.1. 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 an 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, then their positions are switched. It is helpful to assume that the items that are compared are switched also if they are equal. (This convention does not change the contents of the nodes, but influences the movements of single items and simplifies our analysis of these movements significantly.)

The computation of M consists of T parallel steps. For $i \leq T$, during step i 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 *period* of M. Odd-Even Transposition Sort [Knuth 1998] 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 period is 2.

We call a comparator network *monotonic*, if for every compare-exchange operation $[v_i, v_j]$ holds: $i < j$ (Knuth [1998] calls it a *standard* compare-exchange operation). Since every nonmonotonic sorting network can be converted easily into a monotonic one [Knuth 1998, p. 238], we confine our considerations to monotonic sorting networks.

Traditionally, comparator networks are considered as consisting of n vertical parallel wires connected by comparators. Each of the n input items moves through the wires. There is exactly one item on each wire. The items may be exchanged between wires by comparators: comparator $[i, j]$ originating at wire i and pointing 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 devices 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, if the network is to be practically implemented, it is reasonable to interleave the layers to save space.)

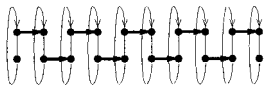

FIG. 1. An implementation of Odd-Even Transposition Sort.

This model is usually used to illustrate comparator networks. If a network has period c, then it consists of T/c identical parts.

1.2. DESIGN CRITERIA. Sorting is a problem studied already for decades. Nevertheless, new application environments such as routing packets in fast communication networks state new design goals. In the last case, sorting has to be done extremely fast, so only hardware solutions are feasible. In order to keep balance between speed and costs, the resulting circuits should have small layout and a clear geometrical structure. Of course, one must be aware of the AT^2-bound yielding a trade-off between speed and area. For this reason, we do not confine ourselves to the fastest networks, which necessarily have large layout area.

Comparator networks have advantage that their basic operations are very easy to implement. Another advantage is to use periodic comparator networks. In this case, even in the model with wires, 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 such a modification of Odd-Even Transposition Sort on Figure 1). In our setting (nodes communicating through the communication links) periodicity requires very little control logic at of each of the nodes. An additional advantage of periodic sorting networks is that they can work in an adaptive mode, that is, if no exchange occurs during a single cycle, then nothing will change during the following cycles. So the string must be already sorted and we may halt the execution.

The main problem with this approach is to design periodic comparator networks that have acceptable speed despite a small period. This turns out to be a nontrivial problem.

1.3. PREVIOUS WORK. The concept of periodic sorting has been introduced by Schröder [1983]. The most famous sorting networks such as Batcher's [1968] networks and the AKS network [Ajtai et al. 1983] are not periodic. However, there has been much effort to construct periodic comparator networks. Perhaps the most interesting construction in this area was the balanced sorting network with period $\log n$ that sorts n items in time $\log^2 n$ [Dowd et al. 1989]. The underlying graph of this network is the $\log n$-dimensional hypercube. Another standard periodic algorithm is Shearsort [Scherson et al. 1986; Sado et al. 1986]. On a $\sqrt{n} \times \sqrt{n}$-mesh, Shearsort 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 period equals $2\sqrt{n}$.

In order to get a constant period one might be tempted to generalize Odd–Even transposition sort by adding some "shortcut edges". This decreases the diameter of the underlying graph and gives a chance for a smaller sorting time. For instance, one may put n vertices on a mesh with a row major order and wrap-around edges between the first and the last column. In this case, the vertical edges may serve as "shortcut edges". However, for most such straightforward generalizations of Odd-Even transposition Sort the runtime equals $\Theta(n)$,

even on average, as shown by Savari [1993]. So despite "shortcut edges", no progress has been achieved! An ingenious modification of the schemes considered by Savari was proposed by Schwiegelshohn [1988]. He introduced a periodic comparator network with period 8 that sorts n items in time $O(\sqrt{n} \cdot \log n)$. The crucial point is to remove some edges to accelerate sorting. The modification is tiny and seems to go into a wrong direction, but the consequences are profound. The idea was followed a very technical and long analysis that has never been published in a complete form. In his Ph.D. dissertation, Krammer [1991] proposes a modification of the algorithm of Schwiegelshohn and suggests that it improves the runtime to $O(\sqrt{n \log n})$ while the underlying graph is still a mesh. The proposal is based on a standard trick used for reducing runtime of Shearsort. Actually, no proof or a proof idea in a crucial point is given in Krammer [1991].

Ierardi [1994] presents a period 4 network, which has a worst case running time of $\Theta(n)$, sorts in average time $O(\sqrt{n \log n})$. This result is clearly inferior to the construction of Schwiegelshohn published 6 years earlier.

For $k \in \mathbb{N}$, Kik et al. [1994] built a periodic comparator network with period $\Theta(k)$ that sorts n items in time $O(k^2 \cdot n^{1/k})$ and in fact is a certain generalization of Odd-Even Transposition Sort. Unfortunately, their construction is based on expander graphs and cannot be applied in practice.

1.4. The New Results. The result of Schwiegelshohn has been obtained by a detailed and technically involved analysis of routes of zeroes and ones. This leaves little hope for generalizations of this algorithm. We provide a considerably simpler approach that may be applied to a vast family of algorithms. For the sake of simplicity of presentation, we start by applying this method for the simplest case, namely, we revisit the case of the two-dimensional mesh:

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

We shall see that if the input contains a single column of 1's and 0's otherwise, then our network requires $\Omega(\sqrt{n} \cdot \log n)$ steps. Hence the above time bound is asymptotically 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.

It is easy to see that Odd–Even Transposition Sort is the only periodic sorting network with period 2. So the period of the network obtained in Theorem 1.1 cannot be further improved without loosing its good time performance.

The techniques that we develop in the proof of Theorem 1.1 are basic tools for a more general construction called *the periodification scheme*. It shows how to convert an arbitrary comparator sorting network into a periodic sorting network with period 5 with at most a slight loss of efficiency.

Theorem 1.2. *Let M be a monotonic* (*nonperiodic*) *comparator network of runtime* $T(n)$ *that sorts n items and has layout area* $A(n)$ *in the traditional model. Then there is a periodic comparator network with period* 5 *and layout area* $O(A(n) \cdot T(n))$ *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 compared with the original network.

The periodification scheme of Theorem 1.2 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 Thompson [1983]):

COROLLARY 1.3. *There is a practical periodic comparator network with period* 5 *and layout area* $O(n^2/\log^2 n)$ *that sorts* n *items in time* $O(\log^3 n)$.

The best runtime, at least theoretically, is achieved by the periodification scheme if we apply it to the AKS network.

COROLLARY 1.4. *There is a periodic comparator network with period* 5 *that sorts* n *items in time* $O(\log^2 n)$.

No lower time bound specific for sorting networks of constant period is known. So we do not know if the runtime given by Corollary 1.4 is optimal. Establishing precise runtime limits for sorting networks of a constant period seems to be a difficult mathematical problem.

The sorting networks that we construct are relatively simple in their design. Nevertheless, an analysis of runtime is technically challenging. 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.

Finally, let us remark that the periodification scheme is a general method for transforming sorting networks into another sorting networks preserving their relevant properties and extending them by an additional important property (periodicity). Another construction of this kind, namely Columnsort, has been proposed by Leighton [1985]. His procedure transforms an arbitrary sorting network into a sorting algorithm running on a certain processor network of degree 3.

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.

1.5. ORGANIZATION OF THE PAPER. In Section 1.5, we recall, for the readers convenience, some well-known facts on sorting networks. In Section 2, we present and analyze a network mentioned in Theorem 1.1. The reader may skip the details of the runtime analysis (Section 2.5) at the first reading. In Section 3, we present periodification scheme based on the case of a 2-dimensional mesh described in Section 2.

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

PROPOSITION 1.5 (0–1 PRINCIPLE [KNUTH 1998, P. 223]). *If a comparator network sorts all inputs consisting solely of* 0*'s and* 1*'s, then it sorts arbitrary inputs.*

PROPOSITION 1.6 ([KNUTH 1998, P. 240]). *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 at least once.*

By Proposition 1.6, a single cycle of a periodic sorting network must contain all comparators $[i, i + 1]$.

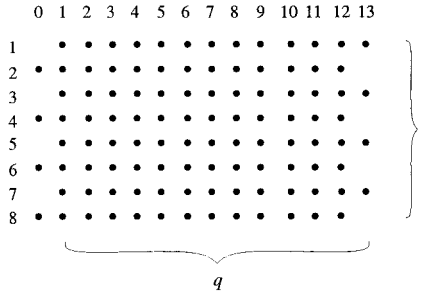

FIG. 2. Arrangement of the nodes for $p = 8$ and $q = 13$.

Let $\vec{x} = (x_1, \ldots, x_n)$, $\vec{y} = (y_1, \ldots, y_n)$. We say that $\vec{x} \leq \vec{y}$, if $x_i \leq y_i$ for every $i \leq n$. We say also that vector $\vec{x} = (x_1, \ldots, x_n)$ describes the contents of a network N with n nodes, if for $j \leq n$ the jth node of N contains x_j.

PROPOSITION 1.7. *Let N be a comparator network. Let $\vec{x}, \vec{y} \in \{0, 1\}^n$ be inputs for N, $\vec{x} \leq \vec{y}$. Let $\vec{x}_i$ ($\vec{y}_i$) describe the contents of N immediately after step i of N on input $\vec{x}$ ($\vec{y}$). Then $\vec{x}_i \leq \vec{y}_i$ for every step i of N.*

By applying Proposition 1.7 for every single step of a network, we get the following corollary:

COROLLARY 1.8. *Let N be a comparator network and $\vec{x} \in \{0, 1\}^n$ an arbitrary input to N. Let $\vec{x}_i$ describe the contents of N immediately after step i during the computation on input $\vec{x}$. Let $\vec{y}_0 = \vec{x}, \vec{y}_1, \vec{y}_2, \ldots$ be a sequence of vectors obtained in the following way: for every i, $y_{i+1}^{\rightarrow}$ is obtained from $\vec{y}_i$ by applying the ith step of network N and then replacing some number of 0's by 1's. Then $\vec{x}_i \leq \vec{y}_i$ for every i.*

PROPOSITION 1.9 [DE BRUIJN 1974]. *If a periodic comparator network M is monotonic and all compare-exchange operations of a single cycle of Odd–Even Transposition Sort are performed at every cycle of M, then M sorts every sequence of n items in at most n cycles.*

PROPOSITION 1.10. *Let M be a monotonic sorting network that sorts n items. Let M be considered in the traditional way: consisting of n vertical wires connected by comparators. If we remove the first k consecutive wires and the last ℓ consecutive wires and the comparators incident to these wires, then we get a sorting network M' that sorts $n - k - \ell$ items.*

2. *A Periodic Sorting Network with Period* 3

2.1. THE CONSTRUCTION OF THE NETWORK. Architecture of the periodic comparator network M with period 3 that sorts in time $O(\sqrt{n} \cdot \log n)$ 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 Figure 2. Formally, for even p and odd q, where $q = \Theta(p)$, 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 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*: $P_{i,j} < P_{i',j'} \Leftrightarrow (i < i') \vee (i = i' \wedge j < j')$.

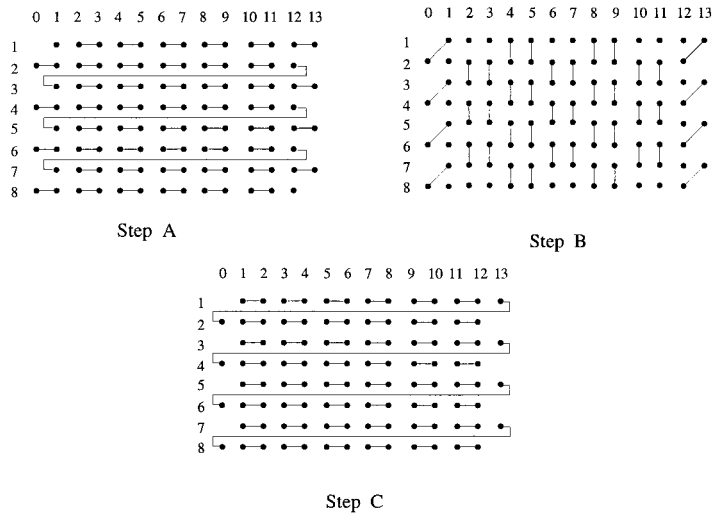

FIG. 3. The three parallel steps performed by M during one cycle.

Each cycle of the algorithm consists of the three parallel steps shown in Figure 3: Step A, Step B, and Step C, performed in this order. The orientation of the comparators is not shown since it is assumed that the network is monotonic. Steps A and C are called *horizontal* and Step B is called *vertical*. The horizontal steps ensure that, as required by Proposition 1.6, 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 C contain long edges connecting the border columns (so called "Schwiegelshohn" edges) *every second row*, similarly as in the case of the networks of Schwiegelshohn [1988] and Krammer [1991]. We cannot put all "Schwiegelshohn" edges into a single horizontal step since then even the average runtime of the resulting algorithm would be of order n [Savari 1993]. 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 moved into two different columns (see Figure 5). This effect is crucial for Schwiegelshohn's construction [Schwiegelshohn 1988].

2.2. GENERAL PROPERTIES OF M. In this section, we define basic notions that we use in the context of M and of the periodification scheme. Most of them are borrowed from the approach of Schwiegelshohn. We start with some general properties holding for all Schwiegelshohn-like networks; however, we state them for M. Nevertheless, we should keep in mind that we shall use them in Section 3.

Definition 2.1. *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.

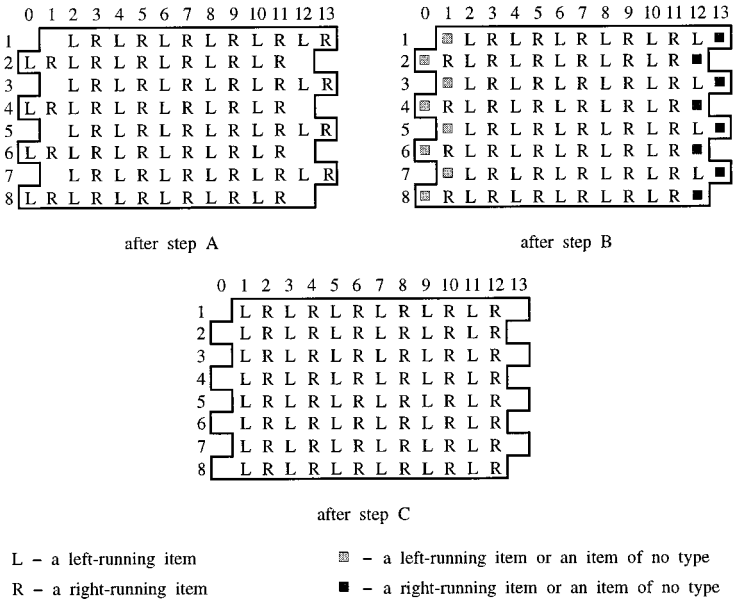

FIG. 4. R- and L-items after Step A, B, and C.

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

Obviously, every internal column is alternately an L-column and an R-column, and the changes occur every horizontal step. The situation at the border columns is more complicated. The locations of right- and left-running items are shown by Figure 4.

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

Fact 2.2. 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. Similarly, 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 0's 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 1's that become right-running at this step. Similarly, L-columns "move left".

Let $w(K_i, t)$ denote the number of 1's in column K_i immediately after step t. Since horizontal and vertical steps of M are interleaved, we introduce additional notation: let h_t denote the tth horizontal step and v_t denote the tth vertical step of the algorithm. The following facts result obviously from our considerations above:

Fact 2.3. If K_i, $i < q - 1$, is an R-column immediately after step h_t, 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, $i > 1$, is an

L-column immediately after step h_t, 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 2.4. If K_i, $i + k < q$, 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 $w(K_{i+k}, h_{t+k}) \geq x$. Similarly, if K_j, $j > k$, 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 $w(K_{j-k}, h_{t+k}) \leq x$.

Fact 2.5. Suppose that K_i is an R-column immediately after step h_t. Then $w(K_j, h_t) \leq w(K_i, h_t)$ provided that $j < i$, $i - j \leq 2t - 1$ and K_j is an L-column immediately after step h_t (i.e., i and j are of different parities).

Intuitively, the R-column and the L-column that are at K_i and K_j after step h_t have already "met" in the past. Therefore, by Facts 2.3 and 2.4, the L-column has no more ones than the R-column.

PROOF. Let $t' = (i - j)/2$ and $s = j + t'$. Then $t' < t$ and $i = (s + 1) + t'$. Immediately 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 $w(K_s, h_{t-t'}) \leq w(K_{s+1}, h_{t-t'})$. On the other hand, by Fact 2.4, $w(K_{s+1}, h_{t-t'}) \leq w(K_{s+1+t'}, h_t) = w(K_i, h_t)$ and $w(K_s, h_{t-t'}) \geq w(K_{s-t'}, h_t) = w(K_j, h_t)$. Hence, $w(K_i, h_t) \geq w(K_j, h_t)$. □

2.3. MORE SPECIFIC PROPERTIES OF M. While R-columns "move to the right" and L-columns "move to the left", the vertical steps make efforts to sort these columns. (Of course, this process cannot be completed because of the new 1's that join this column.) In order to sort these columns steps of Odd–Even Transposition Sort are performed. 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 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 Corollary 1.8 we get the following fact:

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

Now we discuss what happens at the border columns; as already mentioned, this is crucial for achieving a good runtime. The reader may inspect how a high tower of 1's contained in an R-column reaching the right border is broken into two parts by the Steps A and B (see Figure 5).

Fact 2.7. If step t is a C-step and $w(K_1, t) \leq x$, then $w(K_1, t + 2) \leq x + p/2$.

A more precise estimation of $w(K_1, t + 2)$ is possible, but what we state in Fact 2.7 suffices for our purposes. Note that Fact 2.7 shows that even if the right

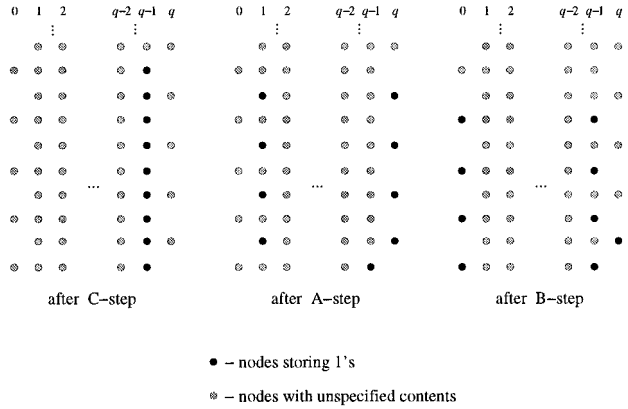

FIG. 5. Breaking up a tower of 1's at the right border.

border contains only 1's, then the new R-columns formed at the left border take only a limited number of 1's from the right border. This prevents the effect of a single high column traveling around the network without loosing its height. (Compare Savari [1993] for some networks on the mesh that exhibit this behavior and therefore require long runtimes.)

PROOF OF FACT 2.7. Let $x < p/2$, since otherwise the fact is trivial. For each even i, it is easy to see 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 K_1; at least $p/2 - x$ of them are stored at the nodes $P_{i,1}$, for i even. Hence, K_1 contains at least $p/2 - x$ zeros after step $t + 2$, that is, at most $p/2 + x$ ones. □

2.4. LOWER BOUND. We show that M requires $\Omega(\sqrt{n} \cdot \log n)$ steps in the worst case. Namely, we consider an input for which K_2 contains $2^s - 1 = \Theta(p)$ ones in the lowest nodes, $(2^s - 1 < q/2)$, and 0's otherwise. The 1's move together to the right and in $\Theta(q)$ steps reach column K_{q-1}, say at step h_{t_0}. It is easy to see on Figure 6 that after step h_{t_0+4}, a new R-column formed at K_2 contains $\lfloor 0.5(2^s - 1) \rfloor = 2^{s-1} - 1$ ones. By Fact 2.6, after $\Theta(q)$ steps an R-column with a foot of height $2^{s-1} - 1$ arrives at the right border. Continuing in the same way we see that after $\Theta(m \cdot q)$ steps there is a column inside M that contains at least $2^{s-m+1} - 1$ ones. Since $2^s - 1 < q/2$, all ones must be moved to the bottom row before the contents of M become sorted. So, if there is a column containing at least two 1's, the contents of M are still not sorted. It follows immediately that the input considered here requires $\Omega(s \cdot q) = \Omega(q \cdot \log p)$ steps.

2.5. RUNTIME ANALYSIS. In this section, we show that M sorts in $O(\sqrt{n} \cdot \log n)$ steps. In order to simplify our analysis we assume that $q > 13p$ and $q - 1$ is divisible by 4. (However, we believe that M performs well also if we leave out these assumptions.)

By the 0–1 Principle (Proposition 1.5), it suffices to consider only sequences consisting of 0's and 1's. As in Schwiegelshohn [1988], our goal is to establish the following fact:

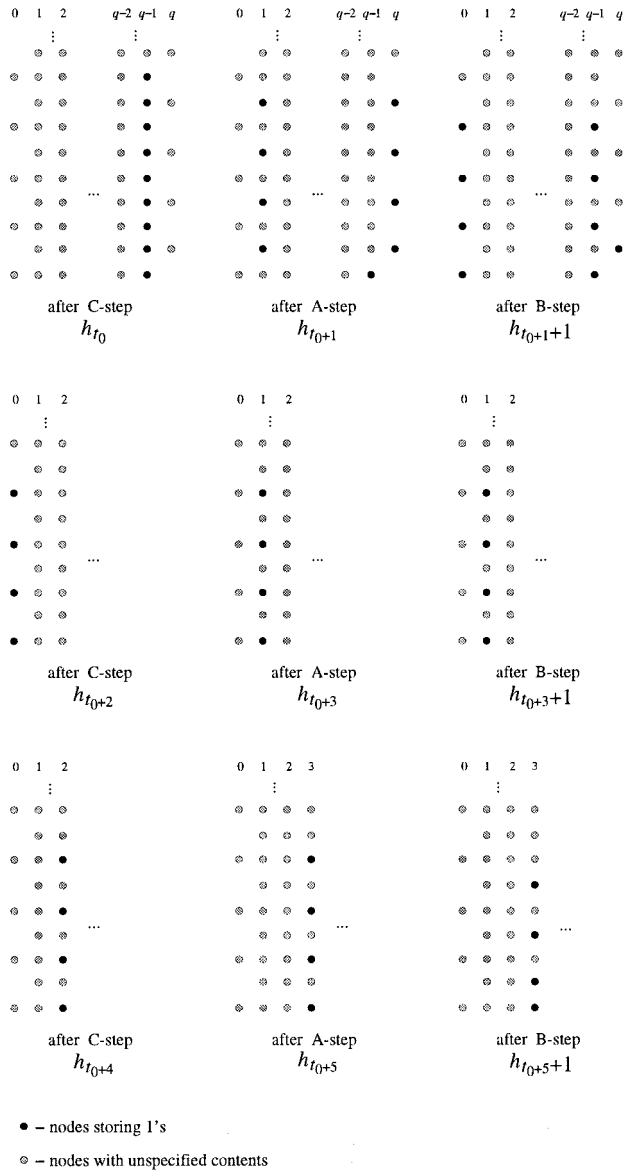

FIG. 6. Breaking up a tower of nine 1's at the right border.

LEMMA 2.8 (KEY LEMMA). *There exist constants c and d such that after executing $c \cdot q$ steps, either*

(*a*) *the $d \cdot p$ bottom rows of G_M contain only* 1*'s, or*
(*b*) *the $d \cdot p$ top rows of G_M contain only* 0*'s.*

By Lemma 2.8, after executing $c \cdot q$ steps, $d \cdot p$ consecutive bottom or top rows can be removed from M without disturbing the computation on the remaining part of the network. Let this remaining part be called M'. Note that M' has the same structure as M (maybe, we have to leave one more row of M in

M'), and differs from M only by its height. The contents of M become sorted exactly when the contents of M' become sorted. For M', we may apply Lemma 2.8 as well. By iterating this process, we see that after $O(q \cdot \log p)$ steps the contents of M become sorted with the exception of $O(1)$ rows lying above a group of rows containing only 1's and below a group of rows containing only 0's. By Proposition 1.9, $O(q)$ additional steps suffice to sort these rows. Hence $O(q \cdot \log p) + O(q) = O(q \cdot \log p)$ steps suffice to sort the contents of M and Theorem 1.1 follows.

In the proof of Lemma 2.8, 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 $(pq)/2$ ones.

Let $g = q - 4p$. We shall prove that after $O(q)$ steps, $\Omega(p)$ 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, every time when K_g is an R-column after step h_g. Therefore, we have to show that K_g *always* contains sufficiently many 1's. It is a little surprising that such a property really holds. Even more surprising and therefore useful is that we do not need to know much about the structure of M to derive this result. Let us remark that at this point we start an analysis that is very much different from the approach of Schwiegelshohn.

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

Proof. Let $w(K_g, h_t) = x$ and K_g be an R-column immediately after step h_t, $t \geq g$. We shall estimate the total number of 1's in G_M by an expression depending on x. Then, we shall see that $x \geq c_1 p$ follows from the assumption that there are at least $(pq)/2$ ones in G_M.

It is crucial to observe how many 1's are in G_M on the left side of K_g. Let $t_0 = t - g$. For $2 \leq i \leq g$, let S_i consist of the columns 2 through i with their contents immediately after step h_{t_0+i}. By Fact 2.4, $w(K_i, h_{t_0+i}) \leq w(K_{i+g-i}, h_{t_0+i+g-i}) = w(K_g, h_t) = x$. It means that the rightmost column of S_i contains at most x ones. Let us consider the number of ones in the columns inside S_i. For the L-columns it is easy, since they have "met" the R-column that is the rightmost column of S_i at the moment. More formally, since $i \leq 2(t_0 + i) - 1$, by Fact 2.5, each L-column inside S_i contains at most x ones. In particular, it follows that $w(K_1, t') \leq x$ if t' is a C-step and $t' > h_{t_0}$. Hence by Fact 2.7, $w(K_1, t' + 2) \leq x + p/2$ (note that step $t' + 2$ is the next B-step after step t').

Now we estimate the number of 1's in R-columns of S_i, and hence the total number of 1's in S_i. Let $w(S_i)$ denote the total number of 1's in S_i. Clearly, $w(S_2) \leq x$ as already seen. In order to estimate $w(S_g)$, 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 1's in each of the columns 2 through $i + 1$. So we only have to consider what happens during step h_{t_0+i+1}. By the construction of M, the total number of 1's in the columns 2 through $i - 1$ 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 1's in the columns 3 through $i - 1$ does not change. S_{i+1} contains R-column K_{i+1} 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, K_{i+1} only. By Fact 2.3, $w(K_i, h_{t_0+i+1}) \leq w(K_i, h_{t_0+i})$. On the other hand,

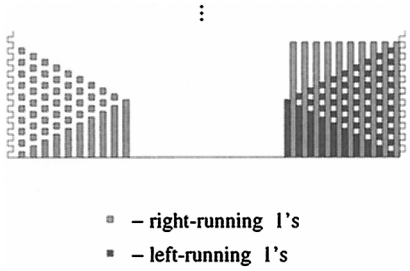

FIG. 7. Sorting L- and R-columns on sides of M.

$w(K_{i+1}, h_{t_0+i+1}) \leq x$. So at the right border of S_{i+1} at most x "new" ones occur. Hence, if step h_{t_0+i+1} is an A-step, then

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

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

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

By inequalities (1), (2) and $w(S_2) = x$, it is easy to derive that $w(S_g) \leq (g - 2) \cdot p/4 + (3g - 4) \cdot x/2$. Taking into account that there are at most $4p^2 - p/2$ ones in the nodes on the right side of K_g and at most $(3/2)p$ ones inside columns K_0 and K_1, we may easily see that the total number of 1's in G_M is less than $(1/4)pq + (3/2)qx + 3p^2$. We have assumed that the total number of 1's in G_M is at least $(pq)/2$. Since $q \geq 13p$, it may be derived that $x \geq ((1/6) - (2/13))p$. □

By Fact 2.6 and Fact 2.9, we get immediately:

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

Fact 2.10 guarantees that every time after step h_q, there are high feet coming to the right border all the time. We shall see that these feet suffice to fill the bottom part of G_M with 1's in reasonable time.

We say that a word $w = w_1 w_2 \cdots w_p \in \{0, 1, *\}^p$ 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 every time after step h_q, have a feet of height $c_1 p$. The 1's of each such a foot are split into two groups: half of the 1's move to the left border through the "Schwiegelshohn" edges and begin their movement to the right; the remaining 1's are "reflected" at the right border and become left-running (see Figure 6). 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 1's are sorted inside the columns by execution of vertical steps (see Figure 7). Below, we examine this phenomenon more closely. Let c_2 be a constant such that $c_2 p \leq \lfloor (c_1 p - 1)/2 \rfloor$. For $j \in \mathbb{N}$, let $\bar{j} = \lfloor j/2 \rfloor$. Let $w_i = (*)^{p - 2c_2 p + \bar{i}} (1*)^{c_2 p - \bar{i}} (1)^{\bar{i}}$. By inspection of Figure 6, we get the following fact.

Fact 2.11. Let $t_0 \geq q$ and t_0 be even. The word w_2 describes column K_2 immediately after C-step h_{t_0+4} and column K_{q-2} immediately after C-step h_{t_0+2}.

The R-column (L-column) that resides in K_2 (K_{q-2}) immediately after step h_{t_0+4} (h_{t_0+2}) moves to the right (left) during the following steps. Its contents during this movement are described as follows (see Figure 7).

Fact 2.12. If $t_0 \geq q$, t_0 even and $2 \leq i \leq 2c_2p$, then w_i describes

(i) column K_i immediately after step h_{t_0+i+2}.
(ii) column K_{q-i} immediately after step h_{t_0+i}.

PROOF

(i) The proof is by induction on i. The case for $i = 2$ holds because of Fact 2.11. We assume that the property claimed holds for i and prove it for $i + 1$.

First, let us assume that i is even. Then h_{t_0+i+2} is a C-step and $h_{t_0+(i+1)+2}$ is the consecutive A-step. During step h_{t_0+i+3} all 1's are moved from K_i to K_{i+1} without changing their vertical positions. Hence, w_i describes K_{i+1} immediately after step h_{t_0+i+3}, too. On the other hand $\bar{\imath} = \overline{i+1}$, if i is even, hence $w_i = w_{i+1}$ and the property claimed holds for $i + 1$.

Now let us assume that i is odd. Then between steps h_{t_0+i+2} and h_{t_0+i+3} a vertical step is executed. It changes positions of 1's in K_i before these 1's are moved to K_{i+1} during step h_{t_0+i+3}. Note that inside K_i, there are vertical edges between the nodes (we count them from the bottom): 1st and 2nd, 3rd and 4th, ..., if $\bar{\imath}$ is even, and between the nodes 2nd and 3rd, 4th and 5th, ..., if $\bar{\imath}$ is odd. Hence, there are always edges between the nodes $\bar{\imath} + 1$ and $\bar{\imath} + 2$, $\bar{\imath} + 3$ and $\bar{\imath} + 4$, ... According to the description of K_i, before executing the vertical step, there are 1's at the nodes $\bar{\imath} + 2$, $\bar{\imath} + 4 \cdots$ (the nodes $\bar{\imath} + 1$, $\bar{\imath} + 3 \cdots$ are of unspecified contents). Therefore, all 1's above row $\bar{\imath}$ are moved one position downwards during the vertical step. Thereby, the lowest 1 joints the foot of 1's and K_i is described afterwards by the word $w = (*)^{p-2c_2p+\bar{\imath}+1}(1*)^{c_2p-\bar{\imath}-1}(1)^{\bar{\imath}+1}$. After executing step h_{t_0+i+3}, the word w describes the contents of K_{i+1}. Note that $\bar{\imath} + 1 = \overline{i+1}$, if i is odd. Hence, $w = w_{i+1}$ and the property claimed holds for $i + 1$.

(ii) The proof is similar. Additionally, we use the fact that each R-column K_{q-j} for $j \leq 2p$ has a foot of height c_1p. Therefore, if during a horizontal step such an R-column K_{q-i-1} is compared with an L-column K_{q-i} described by the word w_i, then immediately after this step L-column K_{q-i-1} is described by w_i. □

By Fact 2.12, the left-running 1's arriving at column K_{q-2c_2p} after step h_{q+2c_2p} form feet of height c_2p. Fact 2.12 also implies that each R-column arriving at K_{2c_2p} after step h_{q+2c_2p+2} has a foot of height c_2p. These feet move to the right (see Figure 8). Finally, after step h_{2q-2c_2p} each R-column K_j, $j \geq 2c_2p$, has a foot of height c_2p. Then the feet of height c_2p of L-columns appearing at column K_{q-2c_2p} can move further to the left up to the column K_{2c_2p} (see Figure 9). These feet fill the middle-bottom part of the network in $q - 4c_2p$ horizontal steps (see Figure 10). Thereby, we have proved the following property:

Fact 2.13. Starting from step h_{t_1}, $t_1 = 3q - 6c_2p$, each column K_i, $2c_2p \leq i \leq q - 2c_2p$, has a foot of height c_2p.

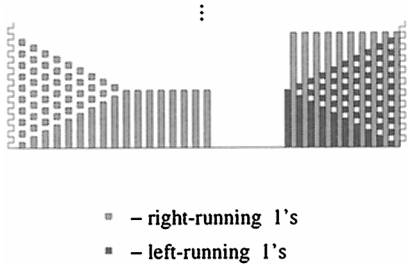

FIG. 8. Filling feet of R-columns in the middle of M.

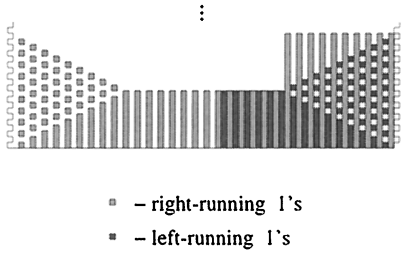

FIG. 9. Filling feet of L-columns in the middle of M.

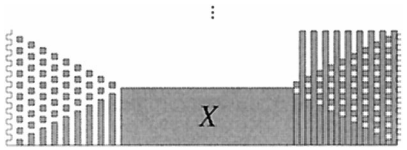

FIG. 10. The situation after step h_{t_1}.

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 2.8, we have to show that X expands to the both borders during the next $O(p)$ steps.

Fact 2.14. After step h_{t_1} region X reaches the left border of the network in $O(p)$ steps, that is, each column K_j, $j < 2c_2p$, 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 $\ell < 2c_2p$ and K_ℓ has not a foot of height c_2p at the considered moment of the computation. First, we assume that $\ell \geq 4$.

Case 1. ℓ *is odd*. By Fact 2.12, after the subsequent Step A, the R-columns K_ℓ and $K_{\ell-2}$ are described by the words w_ℓ and $w_{\ell-2}$, respectively. The vertical edges of the subsequent Step B transform the columns K_ℓ and $K_{\ell-2}$ so that now they are described by

$$w' = (*)^{p-2c_2p+\bar{\ell}+1}(1*)^{c_2p-\bar{\ell}-1}(1)^{\bar{\ell}+1}$$

and

$$u' = (*)^{p-2c_2p+\overline{\ell-2}+1}(1*)^{c_2p-\overline{\ell-2}+1}(1)^{\overline{\ell-2}+1},$$

respectively. Note that $\bar{\ell} = \overline{\ell-2} + 1$. Hence $u' = w_\ell$. The next Step C does not change the contents of the c_2p lowest nodes of K_ℓ, since all corresponding nodes of column $K_{\ell+1}$ contain 1's. Since the 1's from column $K_{\ell-2}$ are moved one

position to the right, the words w_ℓ and w' describe now the columns $K_{\ell-1}$ and K_ℓ, respectively. Then for each $i \le c_2 p$, either $P_{i,\ell-1}$ or $P_{i,\ell}$ contains a 1. These two nodes are compared during the next Step A. Hence, a foot of height $c_2 p$ is formed in K_ℓ.

Case 2. *ℓ is even*. After subsequent Step C, the R-columns K_ℓ and $K_{\ell-2}$ are described by the words w_ℓ and $w_{\ell-2}$, respectively. Similarly as in Case 1, after Step A, these words describe the columns K_ℓ and $K_{\ell-1}$, respectively. Then Step B transforms the columns so that now they are described by

$$(*)^{p-2c_2p+\bar{\ell}+1}(1*)^{c_2p-\bar{\ell}-1}(1)^{\bar{\ell}+1}$$

and

$$(*)^{p-2c_2p+\overline{\ell-2}+1}(1*)^{c_2p-\overline{\ell-2}-1}(1)^{\overline{\ell-2}+1},$$

respectively. Since $\bar{\ell} = \overline{\ell-2} + 1$, for each $i \le c_2 p$, at least one of the nodes $P_{i,\ell-1}$ and $P_{i,\ell}$ contains a 1. Hence after the next Step C, column K_ℓ has a foot of height $c_2 p$.

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 $c_2 p$ (the feet are no longer broken at the right border, since the 1's in the first columns prevent the ones from the right border to go through Schwiegelshohn edges). Such a left-running foot moves to the left until it meets X. Then, it expands 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 by Odd-Even Transposition Sort on their way to the right and left border, respectively. Now we show that, if we replace Odd-Even Transposition Sort by other algorithms, then we may obtain much faster networks, as claimed in Theorem 1.2. The main idea of the construction 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 $\Omega(p)$, then we still have to prove that X expands quickly to the right and left border of the network. The problem is that the proof of this phenomenon in Section 2 depends heavily on the properties of Odd-Even Transposition Sort. In Section 3.1, we design a periodic network with period 5 that avoids this problem no matter which (nonperiodic) algorithm has been used. In Section 3.2, we discuss some features of the construction. In Section 3.3, we prove an upper bound of the runtime of the algorithm.

Since our construction is general, it can be tuned a little bit once we decide upon which network undergoes the periodification scheme. We discuss some of such possibilities in the conclusions section.

3.1. Construction of the Networks. Let N be a (nonperiodic) monotonic sorting network, which sorts p items in T steps. We show how to convert N into

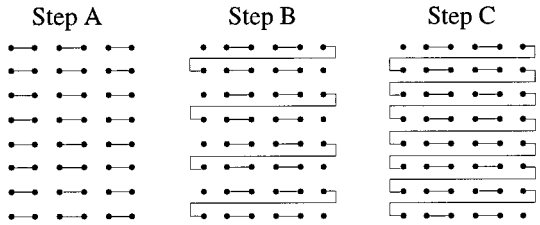

FIG. 11. The horizontal steps for $p = 8$ and $q = 6$.

a periodic network $\mathcal{P}_N$. The nodes of $\mathcal{P}_N$ are arranged in a $p \times q$-rectangle, where p is even and 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, A, C, D. The Steps A, B, and C, called *horizontal*, are depicted by Figure 11. With few exceptions, Step D, called *vertical*, performs only vertical comparisons, that is, comparisons connecting nodes inside the same column. Step D depends on N and its crucial components are *embeddings* of N into $\mathcal{P}_N$. Intuitively, 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 layer j of N and put them into column i_j (see Figure 12).

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 $1 \leq k_1 \leq k_2 \leq p$, there is an edge between the nodes P_{k_1,i_j} and P_{k_2,i_j}, if and only if $[k_1, k_2]$ is a comparator of N at step j (during the nonperiodic computation of N).

In $\mathcal{P}_N$, we distinguish regions X_L^2, Z_L, X_L^1, Y, X_R^1, Z_R, X_R^2 depicted by Figure 13. Each of the regions X_L^i, X_R^i for $i = 1, 2$ consists of $4T$ columns. Four copies of N are embedded into each of these regions. Two copies are used for sorting R-columns and the other two for sorting L-columns. The copies for sorting R-columns are embedded from left to right and the copies for sorting L-columns from right to left. That is, if we index the columns of such a group from left to right by the numbers $1, 2, \ldots, 4T$, then the first copy of N is embedded into the columns $1, 5, \ldots, 4T - 3$ and the second one is embedded into the columns $3, 7, \ldots, 4T - 1$. The other two copies for sorting L-columns are embedded into the columns $4T - 2, 4T - 6, \ldots, 2$ and $4T, 4T - 4, \ldots, 4$. The parts Z_L and Z_R consist of four columns each and contain *slanted* edges depicted by Figure 14. 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.)

Step D performs all comparisons corresponding to the vertical edges introduced by the embeddings of N. Additionally, it performs comparisons corresponding to the slanted edges inside Z_L and Z_R presented at Figure 14 and may perform arbitrary vertical comparisons inside Y.

3.2. REMARKS ON THE CONSTRUCTION. There are some elements in the design of $\mathcal{P}_N$ that need to be commented before we go into technical details. The first interesting feature is the shape of Step C. 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 play a crucial role once there is a "region X" of high

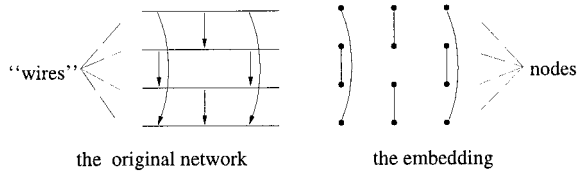

FIG. 12. Embedding of a network.

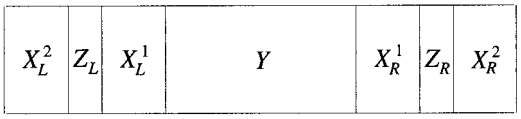

FIG. 13. Partitioning of the nodes of $\mathcal{P}_N$.

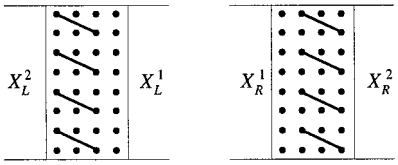

FIG. 14. The edges inside Z_L and Z_R.

feet in the middle of $\mathcal{P}_N$. Each such an R-column with a high foot expands X by one column to the left once it arrives next to X on its left side.

Since for $\mathcal{P}_N$ 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 C. In the situation when every R-column going to the right border of $\mathcal{P}_N$ contains a foot of height x, we can no longer guarantee that every L-column generated on the left border contains at least $\lfloor x/2 \rfloor$ ones. This is guaranteed only for *every second* L-column. This problem is solved as follows: Consider an L-column that contains at least $\lfloor x/2 \rfloor$ ones in the x lowest nodes when it is generated at the right border. After moving through X_R^2 it contains a foot of height $\lfloor x/2 \rfloor$. Then this column moves to Z_R. In Z_R, the foot of this column is splitted into two parts during Step D–every second 1 is sent to the next L-column in Z_R (for which we had no guarantee about a number of 1's contained). Thus, in X_R^1 each L-column contains at least $\lfloor x/4 \rfloor$ ones. Z_L acts symmetrically on the R-columns on the left side of $\mathcal{P}_N$.

The role of Y is only to increase the width of $\mathcal{P}_N$ in order to apply counting arguments (as in the proof of Fact 2.9).

3.3. ANALYSIS OF THE ALGORITHM. The key point in the analysis is, as before, the following lemma:

LEMMA 3.2 (KEY LEMMA). *There exist constants c and d such that after executing $c \cdot q$ cycles either*

(*a*) *the bottom $d \cdot p$ rows of $\mathcal{P}_N$ contain only* 1's, *or*
(*b*) *the top $d \cdot p$* rows of $\mathcal{P}_N$ *contain only* 0's.

Recall that N is monotonic. If we remove consecutive bottom (top) rows containing 1's (0's) only and all incident edges, then we obtain a network $\mathcal{P}_{N'}$ and a corresponding embedded comparator network N'. By Proposition 1.10, N' is a sorting network. Therefore, we may apply Lemma 3.2 repeatedly to show that after executing $O(q \log p)$ cycles 0's fill the top rows of M, 1's fill the bottom rows of M and these rows are separated by $O(1)$ rows containing both 0's and 1's. By Proposition 1.9, the next $O(q)$ cycles suffice to complete sorting. Hence, $\mathcal{P}_N$ sorts in time $O(q \log p) = O(T \log p)$.

The rest of this section is devoted to the proof of Lemma 3.2. We have to follow almost the same sequence of technical facts as in Section 2. When possible, we only sketch the necessary changes.

Since the network is symmetric, we may assume that at least half of the nodes contain 1's. Under this assumption, we show that after $c \cdot q$ cycles the bottom $d \cdot p$ rows of $\mathcal{P}_N$ contain only 1's (i.e., part (a) of Lemma 3.2 holds).

Fact 3.3. When Step D is executed, the slanted edges inside Z_L connect R-columns and the slanted edges inside Z_R connect L-columns. Every R-column (L-column) is incident to the slanted edges during Step D exactly once during its movement through Z_L (Z_R).

PROOF. The columns incident to slanted edges are K_{4T+1}, K_{4T+3} (inside Z_L) and K_{q-4T-2}, K_{q-4T} (inside Z_R). Immediately after Step C, the odd numbered columns are R-columns and the even numbered columns are L-columns (see Figure 11). So at this moment, K_{4T+1}, K_{4T+3} are R-columns. Also, since q is even, K_{q-4T-2}, K_{q-4T} are L-columns at this moment.

For the second part of Fact 3.3, observe that there are four horizontal steps between two consecutive Steps D. Hence, every moving column once influenced by the slanted edges leaves Z_R or Z_L before the next Step D is executed. □

Fact 2.2 remains valid for $\mathcal{P}_N$. Let K_{z_l} be the right-most column of Z_L and K_{z_r} be the leftmost column of Z_R. Fact 2.3 should be reformulated in the following way.

Fact 3.4. If K_i is an R-column immediately after step h_t, $i < q$ and $i \geq z_l$, then $w(K_i, h_{t+1}) \leq w(K_i, h_t) \leq w(K_{i+1}, h_{t+1})$. Similarly, if K_j is an L-column immediately after step h_t, $j > 1$ and $j \leq z_r$, then $w(K_{j-1}, h_{t+1}) \leq w(K_j, h_t) \leq w(K_j, h_{t+1})$.

The assumptions that $i \geq z_l$ and $j \leq z_r$ are necessary, since for the proof, we require that between horizontal steps h_t and h_{t+1} the number of 1's in K_i remains unchanged. Obviously, performing Step D on K_i with $i = z_l, z_r$ may violate this property. Concerning an R-column K_i, this cannot happen, if $i \geq z_l$, by Fact 3.3. Similarly, while concerning an L-column K_j, it suffices to assume that $j \leq z_r$. Otherwise, the proof of Fact 3.4 is the same as the proof of Fact 2.3. Similarly as in Section 2, we get immediately the next fact by induction on k.

Fact 3.5. If K_i is an R-column immediately after step h_t, $w(K_i, h_t) = x$, $i + k \leq q$ and $i \geq z_l$, then K_{i+k} is an R-column immediately after step h_{t+k} and $w(K_{i+k}, h_{t+k}) \geq x$. Similarly, if K_j is an L-column immediately after step h_t, $w(K_j, h_t) = x$, $j - k \geq 1$ and $j \leq z_r$, then K_{j-k} is an L-column immediately after step h_{t+k} and $w(K_{j-k}, h_{t+k}) \leq x$.

Fact 2.6 has to be modified as follows:

Fact 3.6

(i) Let K_s be the first or the third column of X_Z^i, $i \in \{1, 2\}$, $Z \in \{L, R\}$. If K_s is an R-column immediately after step v_t and $w(K_s, v_t) \geq x$, then immediately after step v_{t+T-1}, column K_{s+4T-4} contains a foot of height x.

(ii) Let K_s be column $4T$ or $4T - 2$ of X_Z^i, $i \in \{1, 2\}$, $Z \in \{L, R\}$. Suppose that immediately after step v_t, the column K_s is an L-column and K_s contains x ones in the y lowest registers. Assume further that for $t' \geq v_t$ each R-column in X_Z^i contains a foot of height y. Then immediately after step v_{t+T-1} column K_{s-4T+4} contains a foot of height x.

Fact 3.6 follows immediately from the construction of $\mathcal{P}_N$ and Corollary 1.8. The only difference between Facts 2.6 and 3.6 is that previously we could start at any column K_i. This was possible since the consecutive vertical steps perform Odd-Even Transposition Sort even if we start sorting at K_i. Now, in order to perform the steps 1, 2, 3, ... of N (in this order) we have to start at the beginning of some embedded copy of N.

At Step B, only $p/2$ ones may come through "Schwiegelshohn" edges to the right border. Hence, we get immediately the following property:

Fact 3.7. If step t is a B-step, then $w(K_1, t) \leq w(K_1, t - 1) + p/2$.

Let K_g be the first column of X_R^1.

Fact 3.8. There is a constant c_1 such that if K_g is an R-column immediately after horizontal step h_t, $t \geq g$, then $w(K_g, h_t) \geq c_1 p$.

PROOF. The idea of the proof is essentially the same as in the case of Fact 2.9. The main difference is that during Step C the number of 1's in K_1 may increase by p. Only during Step B this increase is bounded by $p/2$. Additionally, some complications are caused by the slanted edges of Z_L, Z_R and the definition of the S_i's must be changed accordingly. Below we sketch the details.

Let $w(K_g, h_t) = x$ and K_g be an R-column immediately after step h_t, $t \geq g - z_l$. Let $t_1 = t - g + z_l$.

Let S_i consist of columns K_1 through K_{z_l+i} immediately after step h_{t_1+i}. The number of 1's in S_i, denoted by $w(S_i)$ may be estimated as follows:

$$w(S_0) \leq z_l \cdot p$$

(since S_0 contains $z_l \cdot p$ nodes);

$$w(S_{i+1}) \leq w(S_i) + x \qquad \text{if} \quad h_{t_1+i+1} \text{ is an A-step}$$

(indeed, during an A-step no new 1's come through "Schwiegelshohn" edges to K_1; on the right side the number of 1's increases by at most x, since $w(K_{z_l+i+1}, h_{t_1+i+1}) \leq x$ and $w(K_{z_l+i}, h_{t_1+i+1}) \leq w(K_{z_l+i}, h_{t_1+i})$);

$$w(S_{i+1}) \leq w(S_i) + x + \frac{p}{2} \qquad \text{if} \quad h_{t_1+i+1} \text{ is a B-step}$$

(additionally, we have to take into account the increase of the number of 1's in K_1, which is at most $p/2$ by Fact 3.7);

$$w(S_{i+1}) \leq w(S_i) + x + p \qquad \text{if} \quad h_{t_1+i+1} \text{ is a C-step}$$

(we estimate the increase of the number of 1's in K_1 by p). Step D does not influence the number of 1's in S_i. Observe that every second horizontal step is an A-step, every fourth horizontal step is a B-step, and every fourth horizontal step is a C-step. It follows that

$$w(S_{g-z_l}) \leq z_l \cdot p + x \cdot (g - z_l) + \left\lceil \frac{g - z_l}{4} \right\rceil \cdot \frac{3}{2} p.$$

Since S_{g-z_l} comprises of the columns K_1 through K_g, the total number of 1's in $\mathcal{P}_N$ is at most $w(S_{g-z_l}) + (q - g) \cdot p$. Recall that we have assumed that the total number of 1's is at least $(pq)/2$. Therefore, it follows that $x \geq c_1 \cdot p$ for some constant c_1 provided that $q \geq c_2 \cdot T$, where c_2 is an appropriately large constant. We may assume that Y is wide enough so that this assumption is satisfied. □

By Facts 3.6 and 3.8, each R-column arriving on the left side of Z_R after some step $t_2 = O(q)$, $t_2 = q - O(T) - O(1)$, contains a foot of height $x = \Omega(p)$. By Fact 3.3, such a foot is not destroyed while an R-column moves to the right through Z_R; obviously, it is not destroyed while the R-column moves through X_R^2. Therefore we have proved the following fact:

Fact 3.9. There is a $t_3 = O(q)$, $t_3 = t_2 + O(T) + O(1)$, such that

if $t \geq t_3$, then immediately after step t every R-column inside $Z_R \cup X_{\mathrm{R}}^2$ has a foot of height x.

At the right border, R-columns are affected alternately by the Steps B and C. In the case of a C-step, the foot of K_q is moved to K_1 (except the one from the lowest node). This gives rise to a new R-column on the left border that contains a foot of height $x - 1$. In the case of Step B only one half of the 1's contained in the foot of height x of K_q is moved to K_1. Also, since these 1's are moved to every second of the lowest x registers in K_1, they do not form a foot. However, such an R-column created in K_1 moves to the right through X_{L}^2. By Fact 3.6, when this R-column arrives at Z_L, the mentioned 1's form a foot of height $\lfloor x/2 \rfloor$. Therefore, there is a $t_4 = O(q)$, $(t_4 = t_3 + O(T))$, such that

if $t \geq t_4$, then immediately after step t every R-column entering Z_{L} has a foot of height $\lfloor x/2 \rfloor$.

Observe that Step D cannot destroy feet of height $\lfloor x/2 \rfloor$ after step t_4. Indeed, the slanted edges effect columns both with feet of height $\lfloor x/2 \rfloor$. Therefore, there is a $t_5 = O(q)$, $t_5 = t_4 + O(1)$, such that

if $t \geq t_5$, then immediately after step t every R-column leaving Z_{L} has a foot of height $\lfloor x/2 \rfloor$.

(Note that after step t each R-column in Z_{L} has also a foot of height $\lfloor x/2 \rfloor$.) In $O(q)$ steps the first R-column that leaved Z_{L} after step t_5 reaches Z_{R}. Then

every R-column outside X^2_{L} contains a foot of height $\lfloor x/2 \rfloor$, since by Fact 3.9 R-columns in $Z_{\mathrm{R}} \cup X^2_{\mathrm{R}}$ contain feet of height x. This property remains true afterwards, since all R-columns emerging from Z_{L} have feet of height $\lfloor x/2 \rfloor$. So there is a $t_6 = O(q)$, $(t_6 = t_5 + O(q))$, such that

> if $t \geq t_6$, then immediately after step t every R-column outside X^2_{L} has a foot of height $\lfloor x/2 \rfloor$.

Now let us consider what happens on the right side of $\mathcal{P}_N$. After step t_3, every second L-column generated at K_q has $\lfloor x/2 \rfloor$ ones in x lowest registers. These 1's remain left-running at least until they reach X^1_R, because every R-column inside $Z_{\mathrm{R}} \cup X^2_{\mathrm{R}}$ contains a foot of height x. Therefore, by Fact 3.6, it follows that

> if $t \geq t_4 = t_3 + O(T)$, then immediately after step t every second L-column arriving in Z_R has a foot of height $\lfloor x/2 \rfloor$.

Network $\mathcal{P}_N$ is designed so that when an L-column with feet $\lfloor x/2 \rfloor$ mentioned above enters Z_R, then Step D is executed when it is the third from the left column of Z_R. Indeed, the L-column is generated at the right side of the network at Step B. Then, the L-column moves two positions to the left, that is, to the third column from the right, before Step D is executed for the next time. Then between two consecutive steps D, the L-column moves 4 positions to the left. Since the number of columns in X^2_R is divisible by 4, the L-column occurs at the third from the right column of Z_{R}, when Step D occurs. Then, the foot of the L-column is splitted and a half of the ones of the foot is sent into the L-column next to the right. In this way, we sent 1's into an L-column for which we had no guarantee about the number of 1's contained. Hence,

> if $t \geq t_5 = t_4 + O(1)$, then immediately after step t every L-column leaving Z_{R} contains at least $\lfloor x/4 \rfloor$ ones.

Applying Lemma 3.6 for the L-columns that leave Z_{R} with at least $\lfloor x/4 \rfloor$ ones after step t_6, we see that these L-columns arrive in Y with feet of height $\lfloor x/4 \rfloor$. Therefore, there is a $t_7 = O(q)$, $(t_7 = t_6 + O(T))$, such that

> if $t \geq t_7$, then immediately after step t every L-column arriving in Y has a foot of height $\lfloor x/4 \rfloor$.

Since all R-columns outside X^2_{L} have feet of height $\lfloor x/2 \rfloor$, the L-columns with feet of height $\lfloor x/4 \rfloor$ move intact through $Z_{\mathrm{L}} \cup \mathrm{X}^1_{\mathrm{L}} \cup \mathrm{Y}$. In $O(q)$ steps after step t_7, the first such L-column reaches the left end of Z_{L}, say at step t_8. Afterwards all L-columns in $Z_{\mathrm{L}} \cup X^1_{\mathrm{L}} \cup \mathrm{Y}$ have feet of height $\lfloor x/4 \rfloor$, since new L-columns arriving at Y have such feet. On the other hand, all R-columns in $Z_{\mathrm{L}} \cup X^1_{\mathrm{L}} \cup Y$ have already feet of height $\lfloor x/2 \rfloor$. Therefore,

> if $t \geq t_8 = t_7 + O(q)$, then immediately after step t every column in $Z_{\mathrm{L}} \cup X^1_{\mathrm{L}} \cup Y$ has a foot of height $\lfloor x/4 \rfloor$.

Since every second R-column created in K_1 has a foot of height x and since the 1's from the $\lfloor x/4 \rfloor$ lowest registers of such an R-column cannot move out of X^2_{L}, these feet fill the $\lfloor x/4 \rfloor$ rows of X^2_{L} in $O(T)$ steps. Hence, there is a $t_9 = O(q)$, $(t_9 = t_8 + O(T))$, such that

if $t \geq t_9$, then immediately after step t every column in $X_{\mathrm{L}}^2 \cup Z_{\mathrm{L}} \cup X_{\mathrm{L}}^1 \cup Y$ has a foot of height $\lfloor x/4 \rfloor$.

After step t_9, column K_1 contains always a foot of height $\lfloor x/4 \rfloor$. Then, at Steps B and C, the $\lfloor x/4 \rfloor$ lowest registers of K_q retain 1's. In this way, L-columns with feet of height $\lfloor x/4 \rfloor$ are created. They move to the left and once the first such L-column reaches the end of X_{R}^1, all L-columns in $X_{\mathrm{R}}^1 \cup Z_{\mathrm{R}} \cup X_{\mathrm{R}}^2$ have feet of height $\lfloor x/4 \rfloor$. All R-columns in $X_{\mathrm{R}}^1 \cup Z_{\mathrm{R}} \cup X_{\mathrm{R}}^2$ have feet of height $\lfloor x/2 \rfloor$, as previously observed, so all columns in $X_{\mathrm{R}}^1 \cup Z_{\mathrm{R}} \cup X_{\mathrm{R}}^2$ have feet of height $\lfloor x/4 \rfloor$. All other columns have such feet already for every step $t \geq t_9$. Hence, finally,

if $t \geq t_{10} = t_9 + O(T)$, then immediately after step t every column in $\mathcal{P}_N$ has a foot of height $\lfloor x/4 \rfloor$.

Since $t_{10} = O(q)$ and $x = \Omega(p)$, this concludes the proof of Lemma 3.2. □

4. *Conclusions and Final Remarks*

The best sorting time that we have obtained using periodic networks with a constant period is $O(\log^2 n)$. There is no known lower bound for sorting time on periodic comparator networks with constant period other than $\Omega(\log n)$, which holds for arbitrary comparator networks. Despite many efforts no progress has been made until now to close this gap and it seems to be a very difficult mathematical problem.

The periodification scheme can be modified so that we get networks with period 3. The idea is to generalize construction for the two-dimensional mesh in a little different way. Instead of embedding copies of N to cope with broken feet that occur at the borders of $\mathcal{P}_N$, we embed special networks that sort sequences obtained by breaking feet. Generally, there are many ways to achieve this goal. For instance, we may apply a network from Merge Sort algorithm. This is efficient in the sense that the network that we embed has logarithmic depth and need only few columns on the sides of $\mathcal{P}_N$ [Oesterdiekhoff 1997]. The price we have to pay for it is an increase of the layout area so that we cannot guarantee the bound from Theorem 1.2. This is not a fault of Merge Sort, since due to the AT^2 bound, any fast algorithm of this kind needs a large layout area. For Batcher's sorting networks, which themselves require a large layout area, it does not matter and we improve Corollary 1.3 by replacing period 5 by period 3. Similarly, there is a tailored version of periodification scheme for multi-dimensional meshes yielding networks of period 3 (see Loryś et al. [1994] and Oesterdiekhoff [1997] and a forthcoming journal version for technical details).

Periodification has shown to be a useful technique in other contexts. It is hidden in some way in a construction of periodic merging networks of a constant period [Kutyłowski et al. 1998]. Recently, Stachowiak [2000] came up with an idea of a correction network that is based on the periodification paradigm.

ACKNOWLEDGMENTS. 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 some ideas that finally have been implemented by our algorithms. We thank Endre Szemerédi for remarks concerning an early version of the paper. We also thank anonymous referees for their remarks.

REFERENCES

AJTAI, M., KOMLÓS, J., AND SZEMERÉDI, E. 1983. Sorting in $c \cdot \log n$ parallel steps. *Combinatorica 3*, 1–19.

BATCHER, K. E. 1968. Sorting networks and their applications. In *AFIPS Conference Proceedings*, vol. 32. AFIPS Press, Atlantic City, N.J., pp. 307–314.

DE BRUIJN, N. G. 1974. Sorting by means of swapping. *Disc. Math. 9*, 333–339.

DOWD, M., PERL, Y., SAKS, M., AND RUDOLPH, L. 1989. The periodic balanced sorting network. *J. ACM 36*, 738–757.

IERARDI, D. 1994. 2d-Bubblesorting in average time $O(\sqrt{N \lg N})$. In *Proceedings of the 6th ACM Symposium on Parallel Algorithms and Architectures (SPAA)* ACM, New York, pp. 36–45.

KIK, M., KUTYŁOWSKI, M., AND STACHOWIAK, G. 1994. Periodic constant depth sorting networks. In *Proceedings of the 11th Symposium on Theoretical Aspects of Computer Science (STACS)*. Lecture Notes in Computer Science, vol. 775. Springer-Verlag, Berlin, Germany, pp. 201–212.

KNUTH, D. E. 1998. *The Art of Computer Programming*. Vol. 3. *Sorting and Searching*. Addison-Wesley, Reading, Mass.

KRAMMER, J. 1991. *Lösung von Datentransportproblemen in integrierten Schaltungen*. Dissertation, TU München. Munich, Germany.

KUTYŁOWSKI, M., LORYŚ, K., AND OESTERDIEKHOFF, B. 1998. Periodic merging networks. *Theory Comput. Syst. 31*, 5, 551–578.

LEIGHTON, T. 1985. Tight bounds on the complexity of parallel sorting. *IEEE Trans. Comput. 34*, 344–354.

LORYŚ, K., KUTYŁOWSKI, M., OESTERDIEKHOFF, B., AND WANKA, R. 1994. Fast and feasible periodic sorting networks of constant depth. In *Proceedings of the 35th IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society Press, Los Alamitos, Calif., pp. 369–380.

OESTERDIEKHOFF, B. 1997. On periodic comparator networks. Dissertation, University of Paderborn, Paderborn, Germany.

SADO, K., AND IGARASHI, Y. 1986. Some parallel sorts on a mesh-connected processor array and their time efficiency. *J. Parall. Distrib. Comput. 3*, 398–410.

SAVARI, S. A. 1993. Average case analysis of five two-dimensional bubble sorting algorithms. In *Proceedings of the 5th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*. ACM, New York, pp. 336–345.

SCHERSON, I. D., SEN, S., AND SHAMIR, A. 1986. Shear-sort: A true two-dimensional sorting technique for VLSI networks. In *Proceedings of the IEEE International Conference on Parallel Processing*. IEEE Computer Society Press, Los Alamitos, Calif., pp. 903–908.

SCHRÖDER, H. 1983. Partition sorts for VLSI. In *Proceedings of the 13th GI-Jahrestagung*. vol. 73 of *Informatikfachberichte*. Springer-Verlag, New York, pp. 101–116.

SCHWIEGELSHOHN, U. 1988. A short periodic two-dimensional systolic sorting algorithm. In *International Conference on Systolic Arrays* (San Diego, Calif., May 25–27). K. Bromley, S.-Y. Kung, E. Swartzlander, eds. Computer Science Press, Baltimore, Md., pp. 257–264.

STACHOWIAK, G. 2000. Fibonacci correction networks algorithm theory. In *SWAT 2000*, M. M. Halldörsson, ed. Lecture Notes in Computer Science, vol. 1851. Springer-Verlag, Berlin, Germany, pp. 535–548.

THOMPSON, C. D. 1983. The VLSI complexity of sorting. *IEEE Trans. Comput. C-32*, 12, 1171–1184.

RECEIVED SEPTEMBER 1997; REVISED DECEMBER 1999; ACCEPTED JANUARY 2000