Periodic constant depth sorting networks*

Marcin Kik Mirosław Kutyłowski Grzegorz Stachowiak

Institute of Computer Science, University of Wrocław,
ul. Przesmyckiego 20, PL-51-151 Wrocław, Poland,
email: kik,mirekk,gst@ii.uni.wroc.pl

Abstract. Comparator networks of constant depth can be used for sorting in the following way. The computation consists of a number of iterations, say t, each iteration being a single run through the comparator network. The output of a round j ($j < t$) is used as the input for the round $j+1$. The output of the round t is the output of the computation. In such a way, it is possible to apply a network with a small number of comparators for sorting long input sequences. However, it is not clear how to make such a computation fast.
Odd-Even Transposition Sort gives a periodic sorting network of depth 2, that sorts n numbers in $n/2$ iterations. The network of depth 8 proposed by Schwiegelshohn [8] sorts n numbers in $O(\sqrt{n}\log n)$ iterations. Krammer [5] modified the algorithm and obtained a network of depth 6 sorting in $O(\sqrt{n\log n})$ iterations.
For a fixed but arbitrary $k \in \mathcal{N}$, we present a periodic sorting network of depth $O(k)$ that sorts n input numbers in $O(k^2 \cdot n^{1/k})$ steps.

1 Introduction

Comparator networks are widely used for sorting sequences of numbers. A comparator network of depth d sorting n input numbers may be viewed as a set of n registers and a set of communication links between the registers divided into d so called levels. Each level can be represented by a directed graph of degree one, i.e. a single register may be connected to at most one other register at a given level. At step $i \leq d$ of the computation, all links of the ith level are used in parallel to perform comparison-exchange operations between the elements stored in different registers: For a given link (R, R'), if numbers x and x' are stored in R and R', respectively, then after the compare-exchange operation R stores $\min(x, x')$ and R' stores $\max(x, x')$.

A large number of sorting comparator networks has been proposed in the literature. The networks proposed by Batcher [2] are regarded as most successful in practice. They are very elegant in design and have depth $\frac{1}{2}(\log n)^2$. There has been a lot of effort to reduce the depth of sorting comparator networks to $\log n$. This was achieved by the famous AKS network of depth $O(\log n)$ [1]. However, the result is of purely theoretical importance, because of large constants involved.

* supported by KBN grant 2 1197 91 01 and Volkswagen Foundation, Project "Paralleles Rechnen: Theoretische und experimentelle Untersuchungen zu parallelen Rechnenmodellen und systemnahen Algorithmen", partially this work was done while the first and the second author visited Heinz–Nixdorf-Institut, Universität Paderborn

Except for relatively large depth, the networks of Batcher have yet another disadvantage. Every level is different from another. Therefore, if the sorting network is directly wired in a VLSI chip, then there is a large number of wires put into circuitry, making the chip large and expensive. If the comparator network is emulated by a set of processors, then there is a lot of overhead due to routing along many different paths at different times. One may try to overcome these difficulties by applying periodic sorting algorithms [3]. We may use a single comparator network (presumably of a small depth) repeatedly: after getting output that is still not sorted, we put it as an input to the network again. We stop once the output is sorted. The total computation time is therefore $\delta \cdot T$, where δ is the depth of the network and T is the number of iterations. Dowd *et al* [3] proposed a network of depth $\log n$ that sorts in $\log n$ iterations. All communication links of this network are those of an $\log n$-dimensional hypercube.

To achieve low cost sorting circuits for large inputs it would be desirable to reduce the depth of comparator networks used for periodic sorting to a constant while preserving a small number of iterations. The question whether such networks exist has been raised by Meyer auf der Heide [7]. For example, Odd-Even Transposition Sort leads to a network of depth 2 that sorts in $n/2$ iterations, i.e. n parallel steps [6]. Schwiegelshohn's network of depth 8 sorts n elements in $O(\sqrt{n}\log n)$ iterations [8]. Krammer [5] modified the construction of Schwiegelshohn and obtained a network of depth 6 sorting in $O(\sqrt{n\log n})$ iterations.

In this paper we prove the following theorem.

Theorem 1. *Let $k \in \mathcal{N}$ be an arbitrary constant. Let $n \in \mathcal{N}$. There is a comparator network of depth $O(k)$ that sorts n numbers in $O(k \cdot n^{1/k})$ iterations, i.e. in time $O(k^2 \cdot n^{1/k})$.*

The proof of Theorem 1 is constructive – we show how to build the required network using ε-halver networks of Ajtai, Komlós and Szemerédi. However, our network is not applicable directly in practice, since the constants seem to be not small enough and wiring due to expanders might be complicated.

Due to space limitations, some proofs in this paper are only sketched. More technical details can be found in [4].

2 Preliminaries

Definition 2.
A *comparator network* $N = (V, C_1, \ldots, C_d)$ for input sequences of n numbers consists of a set V of n registers $R_1, \ldots, R_n$ and d directed graphs $C_1, \ldots, C_d$, each of degree 1, with vertices in V. For $i \leq d$, C_i is called the ith *level* of N; d is called the *depth* of N. Network N works as follows. Initially, the input numbers are stored in the registers, the ith number in the register R_i. At step t ($t \leq d$), for each arc $(R_j, R_{j'}) \in C_t$ the numbers stored in R_j and $R_{j'}$ are compared. The minimum of them is put into R_j, the maximum into $R_{j'}$. (Since C_t has degree 1, there is at most one arc incident to a given vertex and the definition is unambiguous.) For an input sequence $\mathbf{x} = (x_1, \ldots, x_n)$, by the output of N on $\mathbf{x}$, $N(\mathbf{x})$, we mean the sequence $(x_{j_1}, \ldots, x_{j_n})$, where x_{j_i} denotes the contents of R_i after step d.

A comparator network N is *monotonic* if $j < j'$ for every arc $(R_j, R_{j'}) \in C_t, t \leq d$. (All comparator networks considered in this paper are monotonic).

Definition 3. Let $N^1(\mathbf{x}) = N(\mathbf{x})$ and $N^{j+1}(\mathbf{x}) = N(N^j(\mathbf{x}))$ for $i \geq 1$. We say that network N sorts a sequence $\mathbf{x}$ in at most t iterations, if the sequence $N^t(\mathbf{x})$ is sorted.

To prove that a comparator network sorts in a stated time one can use well known 0–1 *Principle* (see [6]). Clearly, iterating computation of a comparator network is equivalent to a single computation on a comparator network with iterated layers.

Lemma 4. (0–1 Principle) *A comparator network correctly sorts every input sequence if and only if it correctly sorts every input sequence consisting of* 0*'s and* 1*'s.*

3 Construction of the network

In our construction we use ε-halver networks introduced in [1]. We recall their definition and basic properties.

Definition 5. Let $\varepsilon \geq 0$. A comparator network N is an *ε-halver* for inputs of size n if the following holds:

(i) The set of registers of N consists of two subsets V_1 and V_2 of cardinality $n/2$, and all comparison arcs point from V_1 to V_2.

(ii) Let the input for N contain k ones and $n-k$ zeroes. If $k \leq n/2$, then the output of N contains at most $\varepsilon \cdot k$ ones stored in V_1. If $k \geq n/2$, then the output of N contains at most $\varepsilon \cdot (n-k)$ zeroes stored in V_2.

We shall also say that N is a (V_1, V_2, ε)-halver.

Lemma 6. [1] *For each $\varepsilon > 0$ and $n \in \mathcal{N}$, there exists an ε-halver for inputs consisting of n numbers which has depth $O(1/\varepsilon \cdot \log(1/\varepsilon))$.*

The construction of ε-halver networks is based on expander graphs. In fact, ε-halver networks are basic components of the AKS sorting network.

Definition 7. Let $N = (V, C_1, \ldots, C_d)$ and $N' = (V', C'_1, \ldots, C'_{d'})$ be comparator networks. If $V \cap V' = \emptyset$ and $d = d'$, then by $N \cup N'$ we mean the network $(V \cup V', C_1 \cup C'_1, \ldots, C_d \cup C'_d)$. If $V = V'$, then $N|N'$ denotes the network $(V, C_1, \ldots, C_d, C'_1, \ldots, C'_{d'})$.

(ε, m)-blocks that we define below are key components of our sorting network.

Definition 8. An *(ε, m)-block* built on registers $R_1, \ldots, R_n$ is defined as follows. Let $l = \lceil \frac{n}{m} \rceil$. For $i < l$, let $K_i = \{R_{m\cdot(i-1)+1}, \ldots, R_{m\cdot i}\}$, and $K_l = \{R_{m\cdot(l-1)+1}, \ldots, R_n\}$. Let E_i be an $(K_i, K_{i+1}, \varepsilon)$-halver. (If K_l consists of less than m registers, then we add dummy registers to make K_l contain m registers, we take an $(K_{l-1}, K_l, \varepsilon)$-halver and remove all arcs pointing to the dummy registers; E_{l-1} consists of the arcs that remain.) Let $P = (E_1 \cup E_3 \cup E_5 \cup \cdots)$ and $N = (E_2 \cup E_4 \cup E_6 \cup \cdots)$.

Then the network $P|N$ is called an (ε, m)-block. P is called the first layer and N is called the second layer of the (ε, m)-block. In particular, a $(0,1)$-block is a network $(\{R_1, \ldots, R_n\}, C_1, C_2)$, where $C_1 = \{(R_1, R_2), (R_3, R_4), \ldots\}$ and $C_2 = \{(R_2, R_3), (R_4, R_5), \ldots\}$

We can imagine the registers of an (ε, m)-block to be arranged in an $m \times l$-matrix, with K_i being the ith column of the matrix. Between each pair of columns we put an ε-halver . During the first part of computation corresponding to P we apply ε-halversfor the pair of columns K_1 and K_2, K_3 and K_4, ... During the second part of the computation corresponding to N, we use ε-halversfor the pairs of columns K_2 and K_3, K_4 and K_5, ... The crucial property of (ε, m)-blocks is that in $O(l)$ iterations each element is moved to a column not far from the column of its final destination in the sorted sequence (to be proved in Section 4).

Now we complete the construction of our comparator network that sorts n numbers in $O(k^2 \cdot n^{1/k})$ iterations:

Definition 9. For $n, k \in \mathcal{N}$, let $I_{n,k}$ be a comparator network of the form

$$I_{n,k} = F_1|F_2|\cdots|F_k|F_{k+1}$$

where F_{k+1} is a $(0,1)$-block, and F_i is an $(\varepsilon, n^{(k-i)/k} \cdot (c \cdot \log n)^{i-1})$-block for $i = 1, \ldots, k$. (The constants c and ε $(0 < \varepsilon < \frac{1}{2})$ will be determined later.)

It follows from the definition that $I_{n,k}$ has depth $2\delta k + 2$, where $\delta = O(\frac{1}{\varepsilon} \cdot \log(\frac{1}{\varepsilon}))$ denotes the depth of the ε-halvers used in the construction.

4 The time bounds

This section is organized as follows. First we analyze computation of a single (ε, m)-block. In Subsection 4.1 we formulate the main lemma and make some straightforward observations. Subsections 4.2, 4.3, 4.4 are devoted to the proof of the main lemma. Finally, in Subsection 4.5 we use the main lemma to estimate the number of iterations required by the network $I_{n,k}$.

4.1 Main lemma

Let B be an (ε, m)-block consisting of registers $R_1, \ldots, R_n$, $l = \lceil \frac{n}{m} \rceil$. By $K_i = K_i(B, m)$ we denote the ith column of B of height m, i.e., $K_i = \{R_{m(i-1)+1}, \ldots, R_{mi}\}$, for $i < l$, and $K_l = \{R_{m(l-1)+1}, \ldots, R_n\}$. We shall use this notation throughout the whole section.

Definition 10. Let $C = X_1|B|X_2$ and let the registers of C store 0's and 1's, only. Then we say that network C is at most (p, m)-dirty, if there is j such that the columns $K_1, \ldots, K_j$ contain only zeroes, and the columns $K_{j+p+1}, \ldots, K_l$ contain only ones.

Lemma 11. (Main Lemma) *Let B' be a comparator network such that $B' = X_1|B|X_2$ for some monotonic comparator networks X_1, X_2 and the (ε, m)-block B. There are constants $\alpha \in \mathcal{N}$ and $\beta \in \mathcal{R}$, such that if initially B' stores 0's and 1's and is at most (p, m)-dirty, then after $\alpha \cdot p$ iterations, B' is at most $(\beta \cdot \log(p \cdot m), m)$-dirty.*

It is convenient to reduce Main Lemma using three following observations:

Fact 12 *It suffices to prove Main Lemma for $l = p$ and for the case when the last column of B consists of m registers.*

The first claim is obvious. The second claim can be proved by adding lacking registers containing ones to the last column.

Fact 13 *If Main Lemma holds for the case when the number of ones is an odd multiple of m, then it holds in general.*

Sketch of the proof. We need the following easy claim:
Claim. [5] If $\mathbf{x}_1 \subseteq \mathbf{x}_2$ and F is a comparator network, then $F(\mathbf{x}_1) \subseteq F(\mathbf{x}_2)$.
Let $\mathbf{x}$ be an arbitrary input to B' and its input $\mathbf{x}$ contain s ones. Consider $\mathbf{x}_1, \mathbf{x}_2$ such that: $\mathbf{x}_1 \subseteq \mathbf{x} \subseteq \mathbf{x}_2$; for some odd r, $\mathbf{x}_1$ contains $r \cdot m$ ones and $\mathbf{x}_2$ contains $(r+1) \cdot m$ ones. By the claim, $(B')^{\alpha l}(\mathbf{x}_1) \subseteq (B')^{\alpha l}(\mathbf{x}) \subseteq (B')^{\alpha l}(\mathbf{x}_2)$. We replace zeros in $\mathbf{x}_1$ that are ones in $\mathbf{x}$ by $1/3$ and zeros in $\mathbf{x}$ that are ones in $\mathbf{x}_2$ by $2/3$ obtaining $\mathbf{x}'$. Assume $(B')^{\alpha l}(\mathbf{x}_i)$ are $(\beta \log(lm), m)$-dirty. It is easy to see, that the area between columns consisting of zeros and columns consisting of ones in $(B')^{\alpha l}(\mathbf{x}')$ contains at most $2\beta \log(lm) + 1$ columns. It means, that $(B')^{\alpha l}(\mathbf{x})$ is at most $(2\beta \log(lm) + 1, m)$-dirty.□

4.2 Dominance relation

We consider the number of 1's in each column K_i during the computation on inputs consisting of 0's and 1's. This motivates the definitions that follow.

Definition 14. Let $\mathbf{a} = (a_1, \ldots, a_l), \mathbf{b} = (b_1, \ldots, b_l)$, where $a_i, b_i \in [0, m]$ ($a_i, b_i \in \mathcal{R}$), for $i = 1, 2, \ldots, l$. For $i \leq l$, let $\mathrm{head}_i(\mathbf{a}) = \sum_{j=1}^{i} a_j$. We say that $\mathbf{a}$ *dominates* $\mathbf{b}$ (denoted by $\mathbf{a} \succeq \mathbf{b}$), if $\mathrm{head}_l(\mathbf{a}) = \mathrm{head}_l(\mathbf{b})$ and $\mathrm{head}_k(\mathbf{a}) \leq \mathrm{head}_k(\mathbf{b})$ for every $k < l$.

Intuitively, a_i, b_i denote the number of ones in K_i. If $\mathbf{a}$ dominates $\mathbf{b}$, then the sequence of 0's and 1's corresponding to $\mathbf{a}$ is closer to the sorted sequence than the sequence corresponding to $\mathbf{b}$. Equivalently, for $\mathbf{a}, \mathbf{b} \in \{0,1\}^l$, we could define: $\mathbf{a} \succeq \mathbf{b}$ if $\mathbf{a}$ and $\mathbf{b}$ contain the same number of 1's, and for every i, the ith one in $\mathbf{b}$ occurs no later than the ith one in $\mathbf{a}$ (cf. [5]). The following properties follow directly from the definition:

Fact 15 *The relation $\succeq$ is a partial order.*

Fact 16 *Let X be an arbitrary monotonic network on the registers $R_1, \ldots, R_n$. Consider an input consisting of 0's and 1's; let e_i denote the number of 1's in K_i and $\mathbf{e} = (e_1, \ldots, e_l)$. Let x_i denote the number of 1's in K_i after applying X to the input and $\mathbf{x} = (x_1, \ldots, x_l)$. Then $\mathbf{x} \succeq \mathbf{e}$.*

By Fact 16, by applying the network from Main Lemma we get a sequence not worse in sense of $\preceq$ than a sequence produced by block B only. So, we can restrict our considerations to prove Main Lemma for $B' = B$.

Definition 17. Let B' be the network defined in Main Lemma. Let the input of the network B' consist of 0's and 1's. Then $a_{2t,i}$ denotes the number of ones in the column K_i of (ε, m)-block B immediately after iteration t of B', and $a_{2t+1,i}$ denotes the number of ones in K_i after applying the first layer P of B at iteration $t+1$ of B'. Let $\mathbf{a}_t = (a_{t,1}, \ldots, a_{t,l})$.

The vectors $\mathbf{a}_t$ might be very difficult to determine. Even if we fix ε-halvers used to build B, there are many different distributions of 0's and 1's corresponding to the same vector $\mathbf{a}_0$, and each of them influences $\mathbf{a}_t$ in some way. Therefore, we look for simpler vectors $\mathbf{b}_t$ such that $\mathbf{a}_t$ dominates $\mathbf{b}_t$ for each t. The key point is to define $\mathbf{b}_{t+1}$ from $\mathbf{b}_t$ without loosing the property that $\mathbf{a}_{t+1}$ dominates $\mathbf{b}_{t+1}$. To do this we first define pair of functions $(f_1^\varepsilon, f_2^\varepsilon)$ that estimates how a single ε-halver works.

Definition 18. Let $\varepsilon \in [0, \frac{1}{2})$. We define

$$f_1^\varepsilon(x) = \begin{cases} \varepsilon x & \text{for } x \leq m, \\ m - (1-\varepsilon)(2m - x) & \text{for } x > m, \end{cases}$$

$$f_2^\varepsilon(x) = \begin{cases} (1-\varepsilon)x & \text{for } x \leq m, \\ m - \varepsilon(2m - x) & \text{for } x > m, \end{cases}$$

Fact 19 *Let us have a single (V_1, V_2, ε)-halver H and an input $\mathbf{x}$ to H containing x ones. Then $H(\mathbf{x})$ contains not more than $f_1^\varepsilon(x)$ ones in V_1 and not less than $f_2^\varepsilon(x)$ ones in V_2.*

Proof. Follows from the definition of ε-halver .□

In order to define the sequence $\{\mathbf{b}_t\}_{t \geq 0}$ we also need to define on vectors in l-dimensional hypercube $[0, m]^l$ two operations corresponding to layers P and N of (ε, m)-block.

Definition 20. Let $\mathbf{b} = (b_1, b_2, b_3, \ldots) \in [0, m]^l$. We define:

$$P^\varepsilon(\mathbf{b}) = (f_1^\varepsilon(b_1 + b_2), f_2^\varepsilon(b_1 + b_2), f_1^\varepsilon(b_3 + b_4), f_2^\varepsilon(b_3 + b_4), f_1^\varepsilon(b_5 + b_6), \ldots)$$

and

$$N^\varepsilon(\mathbf{b}) = (b_1, f_1^\varepsilon(b_2 + b_3), f_2^\varepsilon(b_2 + b_3), f_1^\varepsilon(b_4 + b_5), f_2^\varepsilon(b_4 + b_5), f_1^\varepsilon(b_6 + b_7), \ldots).$$

The correspondence between $P^\varepsilon, N^\varepsilon$ and layers P, N of (ε, m)-block is given by the following lemma.

Lemma 21. *Let $\mathbf{b}$ be the sequence of numbers of ones in in columns of an (ε, m)-block for an arbitrary input $\mathbf{x}$. Let $\mathbf{b}^P$ and $\mathbf{b}^N$ be sequences of numbers of ones in columns after applying a single layer P or N respectively to $\mathbf{x}$. Then*

$$P^\varepsilon(\mathbf{b}) \preceq \mathbf{b}^P \qquad \text{and} \qquad N^\varepsilon(\mathbf{b}) \preceq \mathbf{b}^N.$$

Proof. Follows from fact 19.□

We need also another technical lemma whose proof we leave to the reader.

Lemma 22. *If* $\mathbf{a} \preceq \mathbf{b}$, *then*

$$P^\varepsilon(\mathbf{a}) \preceq P^\varepsilon(\mathbf{b}) \qquad \textit{and} \qquad N^\varepsilon(\mathbf{a}) \preceq N^\varepsilon(\mathbf{b})$$

To estimate vector $\mathbf{a}_t$ according to relation $\preceq$ we need some starting vector $\mathbf{b}_0$ which is dominated by any vector $\mathbf{a}_0$ representing an input sequence with a given number s of ones.

Definition 23. Let $m|s$. Then $\mathbf{b}_0 = (b_1, \ldots, b_l)$ is the *worst vector for* s *ones*, if $b_i = m$ for $i \leq s/m$, and $b_i = 0$, otherwise.

From now on, we consider only inputs to network B' that contain 0's and 1's, only, and contain exactly $r \cdot m$ ones for some fixed but arbitrary odd number r.

Definition 24. For $i \in \mathcal{N}$, let $\mathbf{b}_i^\varepsilon$ be defined as follows: $\mathbf{b}_0^\varepsilon = \mathbf{b}_0$, $\mathbf{b}_{i+1}^\varepsilon = N^\varepsilon(\mathbf{b}_i^\varepsilon)$, if $i+1$ is odd, and $\mathbf{b}_{i+1}^\varepsilon = P^\varepsilon(\mathbf{b}_i^\varepsilon)$, if $i+1$ is even.

Lemma 25. ${\mathbf{b}^\varepsilon}_t \preceq \mathbf{a}_t$, *for every* t.

For $\varepsilon = 0$, we get vectors $\mathbf{b}_i^0$ that are closely related to the sequences of 0's and 1's occurring during Odd-Even Transposition Sort. Namely, let $\tilde{b_i^0}$ be the sequence obtained from $\mathbf{b}_i^0$ by replacing every m by 1. Then $\tilde{b_i^0}$ is the sequence obtained after i steps of Odd-Even Transposition Sort, when started with $\tilde{b_0}$. So we can reformulate well known properties of Odd-Even Transposition Sort in the following way.

Lemma 26. *For* $t \geq l-1$, $\mathbf{b}_t^0 = (0, \ldots, 0, m, \ldots, m)$. *For* $t < l-1$, $\mathbf{b}_t^0$ *is a sequence containing only* 0*'s in positions* 1 *through* $(l-r)-(l-1-t)$ *and containing only* m*'s in positions* $(l-r+1)+(l-1-t)$ *through* l.

Proof. The lemma follows from the fact that Odd-Even Transposition Sort sorts $\tilde{b_0^0}$ in exactly $l-1$ steps and that at time step t every element must be at a distance not larger than $l-1-t$ from its final destination. □

4.3 Convex estimates

Recall that for $\alpha_1, \ldots, \alpha_k \geq 0$ such that $\sum_{i=1}^k \alpha_i = 1$, the vector $\alpha_1\mathbf{e}_1 + \cdots + \alpha_k\mathbf{e}_k$ is a *convex combination of* $\mathbf{e}_1, \ldots, \mathbf{e}_k$. First, we prove the following property of the operators N^ε and P^ε on convex combinations:

Lemma 27. *Let* $\alpha_1, \ldots, \alpha_k \geq 0$, $\sum_{i=1}^k \alpha_i = 1$ *and* $\mathbf{e}_1, \ldots, \mathbf{e}_k \in [0,m]^l$. *Then*

$$\sum_{i=1}^k \alpha_i N^\varepsilon(\mathbf{e}_i) \preceq N^\varepsilon(\sum_{i=1}^k \alpha_i \mathbf{e}_i) \qquad \textit{and} \qquad \sum_{i=1}^k \alpha_i P^\varepsilon(\mathbf{e}_i) \preceq P^\varepsilon(\sum_{i=1}^k \alpha_i \mathbf{e}_i)$$

Proof. The proof of the lemma follows from the claim:
Claim. *Let* $\alpha, \beta \geq 0, \alpha+\beta = 1$, *and* $x, y \in [0, 2m]$. *Then* $f_1^\varepsilon(\alpha x + \beta y) \leq \alpha f_1^\varepsilon(x) + \beta f_1^\varepsilon(y)$.
Since verification of the claim is elementary and based on the definition of the function f_1^ε, we skip it. □

Lemma 28. *Let $0 \leq i \leq l-2$. If i is even, then*

$$N^\varepsilon(\mathbf{b}_i^0) = N^\varepsilon(\mathbf{b}_{i+1}^0) = \varepsilon\mathbf{b}_i^0 + (1-\varepsilon)\mathbf{b}_{i+1}^0.$$

If i is odd, then

$$P^\varepsilon(\mathbf{b}_i^0) = P^\varepsilon(\mathbf{b}_{i+1}^0) = \varepsilon\mathbf{b}_i^0 + (1-\varepsilon)\mathbf{b}_{i+1}^0.$$

Moreover, $P^\varepsilon(\mathbf{b}_0^0) = \mathbf{b}_0^0$; $P^\varepsilon(\mathbf{b}_{l-1}^0) = \mathbf{b}_{l-1}^0$, if l is even; $N^\varepsilon(\mathbf{b}_{l-1}^0) = \mathbf{b}_{l-1}^0$, if l is odd.

The proof of Lemma 28 is easy and we leave it to the reader.

We may apply Lemma 28 as follows. Assume that $\mathbf{b} \succeq \alpha_0\mathbf{b}_0^0 + \cdots + \alpha_{l-1}\mathbf{b}_{l-1}^0$. Then we get:

$$N^\varepsilon(\mathbf{b}) \succeq N^\varepsilon(\alpha_0\mathbf{b}_0^0 + \cdots + \alpha_{l-1}\mathbf{b}_{l-1}^0) \succeq \alpha_0 N^\varepsilon(\mathbf{b}_0^0) + \cdots + \alpha_{l-1}N^\varepsilon(\mathbf{b}_{l-1}^0) =$$
$$\varepsilon(\alpha_0+\alpha_1)\mathbf{b}_0^0 + (1-\varepsilon)(\alpha_0+\alpha_1)\mathbf{b}_1^0 + \varepsilon(\alpha_2+\alpha_3)\mathbf{b}_2^0 + (1-\varepsilon)(\alpha_2+\alpha_3)\mathbf{b}_3^0 + \cdots.$$

Similarly,

$$P^\varepsilon(\mathbf{b}) \succeq \alpha_0\mathbf{b}_0^0 + \varepsilon(\alpha_1+\alpha_2)\mathbf{b}_1^0 + (1-\varepsilon)(\alpha_1+\alpha_2)\mathbf{b}_2^0$$
$$+\varepsilon(\alpha_3+\alpha_4)\mathbf{b}_3^0 + (1-\varepsilon)(\alpha_3+\alpha_4)\mathbf{b}_4^0 + \cdots.$$

Definition 29.
Convex estimate $\alpha_0^t\mathbf{b}_0^0 + \cdots + \alpha_{l-1}^t\mathbf{b}_{l-1}^0$ of $\mathbf{b}_t^\varepsilon$ is defined as follows:

- for $t = 0$, the convex estimate equals $\mathbf{b}_0^0$ (i.e. $\alpha_0^0 = 1$ and $\alpha_i^0 = 0$ for $i \neq 0$),
- if t is even, then the convex estimate of $\mathbf{b}_{t+1}^\varepsilon$ equals $\alpha_0^t N^\varepsilon(\mathbf{b}_0^0) + \cdots + \alpha_{l-1}^t N^\varepsilon(\mathbf{b}_{l-1}^0)$, that is,

$$\alpha_i^{t+1} = \begin{cases} \varepsilon(\alpha_i^t + \alpha_{i+1}^t) & \text{if } i \text{ is even, } 0 \leq i < l-1, \\ (1-\varepsilon)(\alpha_{i-1}^t + \alpha_i^t) & \text{if } i \text{ is odd, } 0 \leq i \leq l-1, \\ \alpha_{l-1}^t & \text{if } i = l-1, l-1 \text{ is even,} \end{cases}$$

- if t is odd, then the convex estimate of $\mathbf{b}_{t+1}^\varepsilon$ equals $\alpha_0^t P^\varepsilon(\mathbf{b}_0^0) + \cdots + \alpha_{l-1}^t P^\varepsilon(\mathbf{b}_{l-1}^0)$, that is,

$$\alpha_i^{t+1} = \begin{cases} \alpha_0^t & \text{if } i = 0, \\ \varepsilon(\alpha_i^t + \alpha_{i+1}^t) & \text{if } i \text{ is odd, } 0 \leq i < l-1, \\ (1-\varepsilon)(\alpha_{i-1}^t + \alpha_i^t) & \text{if } i \text{ is even, } 0 \leq i \leq l-1, \\ \alpha_{l-1}^t & \text{if } i = l-1, l-1 \text{ is odd.} \end{cases}$$

By Lemma 27, for every t, $\mathbf{b}_t^\varepsilon$ dominates its convex estimate. The coefficients α_j^t are relatively easy to compute. Moreover, one can see that the coefficients of $\mathbf{b}_i^0$, for a small i, quite fast begin to decrease exponentially when t grows. If they become very small, then we shall see that $\mathbf{b}_t^\varepsilon$ is "almost sorted".

4.4 Flow of coefficients

It would be tiresome to estimate the coefficients α_i^t directly. Instead, we use a method based on the concept of *flow of coefficients*:

Definition 30. For $i, t \in \mathcal{N}$, $i \leq l-1$, let $\mathrm{flow}_t(i) = \alpha_i^t - \alpha_i^{t-1}$, if t and i have the same parity, and $\mathrm{flow}_t(i) = 0$, otherwise.

Intuitively, $\mathrm{flow}_t(i)$ describes increase of α_i at expense of α_{i-1} at step t. Indeed, if t and i have the same parity, then $\alpha_i^t = (1-\varepsilon)\cdot(\alpha_i^{t-1}+\alpha_{i-1}^{t-1})$, $\alpha_{i-1}^t = \varepsilon\cdot(\alpha_i^{t-1}+\alpha_{i-1}^{t-1})$, hence $\alpha_i^t + \alpha_{i-1}^t = \alpha_i^{t-1} + \alpha_{i-1}^{t-1}$. So $\alpha_{i-1}^t = \alpha_{i-1}^{t-1} - \mathrm{flow}_t(i)$.

Lemma 31. *Let $i < l$ and let $t+1$ and i have the same parity. Then for $1 < i < l-1$,*

$$\mathrm{flow}_{t+1}(i) = (1-\varepsilon)\cdot\mathrm{flow}_t(i-1) + \varepsilon\cdot\mathrm{flow}_t(i+1).$$

For $i = 1$,

$$\mathrm{flow}_{t+1}(1) = \varepsilon\cdot\mathrm{flow}_t(2).$$

For $i = l-1$,

$$\mathrm{flow}_{t+1}(l-1) = (1-\varepsilon)\cdot\mathrm{flow}_t(l-2).$$

Proof. By the construction of convex estimates, $\alpha_i^{t-1}/\alpha_{i-1}^{t-1} = (1-\varepsilon)/\varepsilon$. Hence

$$(1-\varepsilon)\alpha_{i-1}^{t-1} = \varepsilon\alpha_i^{t-1}. \tag{1}$$

Let $\mathrm{flow}_t(i-1) = x$ and $\mathrm{flow}_t(i+1) = y$. So $\alpha_{i-1}^t = \alpha_{i-1}^{t-1} + x$ and $\alpha_i^t = \alpha_i^{t-1} - y$. Hence, $\alpha_i^{t+1} = (1-\varepsilon)(\alpha_{i-1}^{t-1} + x + \alpha_i^{t-1} - y) = \alpha_i^{t-1} + (1-\varepsilon)x - (1-\varepsilon)y$ (the second equality follows from (1)). So $\mathrm{flow}_{t+1}(i) = \alpha_i^{t+1} - (\alpha_i^{t-1} - y) = (1-\varepsilon)x + \varepsilon y$.

The formulas for $\mathrm{flow}_{t+1}(1)$ and $\mathrm{flow}_{t+1}(l-1)$ can be obtained in a similar way. □

Fact 32 $\mathrm{flow}_t(i) \geq 0$ *for every t, i.*

To get an upper bound on flows we define $\mathrm{upflow}_t(i)$ for $i \in \mathcal{Z}$. Namely, for $t = 1$ we put $\mathrm{upflow}_1(1) = 1$, and $\mathrm{upflow}_1(i) = 0$ for $i \neq 1$. (We put $\mathrm{upflow}_1(1) = 1$ instead of $\mathrm{upflow}_1(1) = 1-\varepsilon$ in order to simplify some expressions.) To define upflow_{t+1} from upflow_t, we use a formula based on the formula for flows given in Lemma 31:

$$\mathrm{upflow}_{t+1}(i) = (1-\varepsilon)\cdot\mathrm{upflow}_t(i-1) + \varepsilon\cdot\mathrm{upflow}_t(i+1),$$

if $t+1$ and i have the same parity, and $\mathrm{upflow}_{t+1}(i) = 0$ otherwise. It easily follows from the definition that $\mathrm{upflow}_t(i) \geq 0$ for any t, i, and $\sum_{i\in\mathcal{Z}} \mathrm{upflow}_t(i) = 1$.

Lemma 33. *For every $t, i \in \mathcal{N}$, $i < l-1$,*

$$\mathrm{flow}_t(i) \leq \mathrm{upflow}_t(i).$$

Lemma 34. *If t and i have the same parity, $-t+2 \leq i \leq t$, then*

$$\mathrm{upflow}_t(i) = \binom{t-1}{\frac{t+i}{2}-1}\varepsilon^{\frac{t-i}{2}}(1-\varepsilon)^{\frac{t+i}{2}-1}. \tag{2}$$

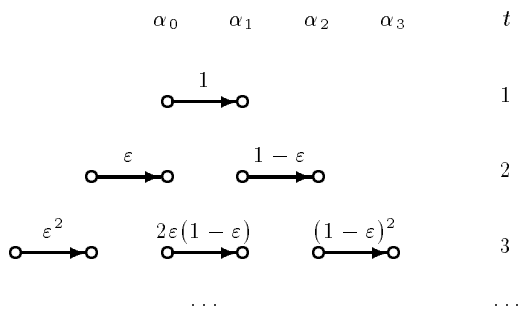

Fig. 1. Upper bounds for the flows

Lemma 34 follows directly from the formulas defining upflows (see also Fig. 1). For the rest of the proof we fix constants $d, c_0 \in \mathcal{R}$ such that

- $d > \frac{1}{1-2\varepsilon}$ and dl is an even natural number,
- $c_0 = \frac{(d+1)\varepsilon}{(d-1)(1-\varepsilon)}$.

Corollary 35. *If $0 < i < l$ and i is even, then*

$$\frac{\mathrm{upflow}_{dl}(i)}{\mathrm{upflow}_{dl}(i+2)} \leq c_0 < 1. \tag{3}$$

Proof. The first inequality may be easily obtained using equality (2). The second one follows from the assumption that $d > \frac{1}{1-2\varepsilon}$. We leave the details to the reader. □

Corollary 36. *If i is even, $0 < i < l$, then* $\mathrm{upflow}_{dl}(i) \leq c_0^{\lceil (l-i)/2 \rceil}$. *Hence,* $\mathrm{flow}_{dl}(i) \leq c_0^{\lceil (l-i)/2 \rceil}$.

Proof. By inequality (3), we get $\mathrm{upflow}_{dl}(i) \leq c_0^j \cdot \mathrm{upflow}_{dl}(i+2j)$, for $i + 2j \leq l + 1$. Let $s = \lceil (l-i)/2 \rceil$. Then $i + 2s \leq l + 1$. The sum of all upflows is 1, so $\mathrm{upflow}_{dl}(i+2s) \leq 1$. Hence $\mathrm{upflow}_{dl}(i) \leq c_0^s$. The second part of the lemma follows from Lemma 33. □

Lemma 37. *There is a constant c_1 such that for every odd i, $0 < i < l$,*

$$\alpha_{i-1}^{dl-2} + \alpha_i^{dl-2} < c_1 \cdot c_0^{(l-i)/2}.$$

The proof requires a bit labour, but is elementary, so we omit it.

We are now looking for an η such that $\sum_{i=0}^{l-\eta} \alpha_i^{dl-2} < \frac{1}{ml}$, i.e., up to $\alpha_{l-\eta}^{dl-2}$, the coefficients α_i^{dl-2} are very small (which means that the vectors $\mathbf{b}_i^0$ contribute very little to the convex estimation of $\mathbf{b}_{dl-2}^{\varepsilon}$).

Lemma 38. *There is a constant $c_2 > 0$ such that $\sum_{i=0}^{l-\eta} \alpha_i^{dl-2} < \frac{1}{ml}$ for $\eta = c_2 \cdot \log(m \cdot l)$.*

Proof. We consider η such that $l-\eta$ is odd. Then

$$\sum_{i=0}^{l-\eta} \alpha_i^{dl-2} = \sum_{j=0}^{(l-\eta-1)/2} x_{2j+1}^{dl-2} \leq c_1 \cdot c_0^{\eta/2} \cdot \sum_{s=0}^{\infty} c_0^s = \tfrac{c_1}{1-c_0} \cdot c_0^{\eta/2}.$$

For an $\eta = O(\log(ml))$ appropriately chosen (namely, for $\eta > \frac{2}{-\log c_0}(\log(ml) + \log \frac{c_1}{1-c_0})$), the last expression is smaller than $\frac{1}{ml}$. □

Proof of Main Lemma (Lemma 11). Recall that

$$\mathbf{a}_t \succeq \mathbf{b}_t^{\varepsilon} \succeq \sum_{i=0}^{l-1} \alpha_i^t \cdot \mathbf{b}_i^0.$$

It follows that for every $h \leq l$, $\mathrm{head}_h(\mathbf{a}_t) \leq \mathrm{head}_h(\sum_{i=0}^{l-1} \alpha_i^t \cdot \mathbf{b}_i^0)$. Let $\eta = c_2 \cdot \log(m \cdot l)$ and $h = l - r - \eta - 1$. Then

$$\mathrm{head}_h(\mathbf{a}_{dl-2}) \leq \sum_{i=0}^{l-1} \alpha_i^{dl-2} \cdot \mathrm{head}_h(\mathbf{b}_i^0).$$

By Lemma 26, $\mathbf{b}_i^0$ contains only zeroes up to position $(l-r)-(l-1-i)$. For $i > l-\eta$, $(l-r)-(l-1-i) > h$, hence $\mathrm{head}_h(\mathbf{b}_i^0) = 0$. So

$$\mathrm{head}_h(\mathbf{a}_{dl-2}) \leq \sum_{i=0}^{l-\eta} \alpha_i^{dl-2} \cdot \mathrm{head}_h(\mathbf{b}_i^0) \leq ml \cdot \sum_{i=0}^{l-\eta} \alpha_i^{dl-2}.$$

Hence by Lemma 38, $\mathrm{head}_h(\mathbf{a}_{dl-2}) < 1$. Since $a_{t,i} \in \mathcal{N}$ for every t and i, it follows that $a_{dl-2,1}, \ldots, a_{dl-2,l-r-\eta-1} = 0$.

The proof that $a_{dl-2,h}, \ldots, a_{dl-2,l} = m$, for $h = (l-r+1)+\eta+1$, is similar. (We change the directions of all comparators and obtain an (ε, m)-block that works in opposite direction. Ones in this (ε, m)-block behave exactly like zeroes in the original (ε, m)-block.) It follows that after iteration $(dl-2)/2$, the contents of the network is at most $(2\eta+2, m)$-dirty. Since $\eta = O(\log(m \cdot l))$, Main Lemma follows. □

4.5 Analysis of the algorithm

We shall show that the network $I_{n,k}$ sorts in $\alpha k n^{1/k} + (c \log n)^k$ iterations, where the constants α, β are from Main Lemma while $c = \beta + 1$ is a parameter from the definition of $I_{n,k}$. By 0-1 Principle, it suffices to consider inputs consisting of 0's and 1's. We divide the iterations into $k+1$ phases. Each of the first k phases consists of $\alpha n^{1/k}$ iterations; the last phase consists of the last $(c \log n)^k$ iterations. In order to analyze phase i, for $i \leq k$, we use the fact that $I_{n,k}$ is of the form $X_1|F_i|X_2$, where X_1, X_2 are some monotonic networks. Let m_i denote the size of the columns of F_i, that is, $m_i = n^{(k-i)/k} \cdot (c \log n)^{i-1}$ and $m_{k+1} = 1$.

Claim 39 *For $i \leq k$, at the beginning of phase i, the network $I_{n,k}$ is at most $(n^{1/k}, m_i)$-dirty. At the beginning of phase $k+1$, $I_{n,k}$ is at most $((c \log n)^k, 1)$-dirty.*

Proof. The proof is by induction on i. By definition, every input sequence is at most $(n^{1/k}, n^{(k-1)/k})$-dirty. Hence the lemma holds for $i = 1$. By Main Lemma, if the input of phase i is at most $(n^{1/k}, m_i)$-dirty, then the output of phase i is at most $(\beta \log(n^{1/k} \cdot m_i), m_i)$-dirty. Since $n^{1/k} \cdot m_i \leq n$, the output is at most $(\beta \log n, m_i)$-dirty, as well. This means that the number of registers from the first register containing a 1 to the last register containing a 0 is at most $m_i \cdot \beta \log n$. For $i < k$, these registers are contained in at most $\lceil \frac{m_i \cdot \beta \log n}{m_{i+1}} \rceil + 1 = \lceil n^{1/k} \frac{\beta}{\beta+1} \rceil + 1$ adjacent columns of (ε, m_{i+1})-block F_{i+1}. We may assume that n is sufficiently large so that $\lceil n^{1/k} \frac{\beta}{\beta+1} \rceil + 1 \leq n^{1/k}$. Hence the input of phase $i+1$ is at most $(n^{1/k}, m_{i+1})$-dirty. If $i = k$, then $m_i \cdot \beta \log n \leq (c \log n)^k$, hence the output of stage k is at most $((c \log n)^k, 1)$-dirty. □

The output of phase k is at most $((c \log n)^k, 1)$-dirty. $I_{n,k} = X | F_{k+1}$, where X is some monotonic network, so by Lemma **??**, after $\lceil (c \log n)^k / 2 \rceil$ iterations at stage $k+1$ the sequence stored by $I_{n,k}$ is sorted.

The total time for sorting a sequence of n elements on $I_{n,k}$ is the depth of the network multiplied by the number of necessary iterations, that is, at most $(2\delta k + 2)(k \alpha n^{1/k} + (c \log n)^k) = O(k^2 \cdot n^{1/k})$, where $\delta = O(\frac{1}{\varepsilon} \cdot \log \frac{1}{\varepsilon})$ is the depth of an ε-halver. This completes the proof of Theorem 1. □

Acknowledgment

We thank Friedhelm Meyer auf der Heide for presenting us the problem of periodic sorting networks of constant depth. Many ideas leading to the solution presented in the paper have been contributed by Friedhelm Meyer auf der Heide, Juraj Hromkovič, Krzysztof Loryś and Rolf Wanka.
Added in proof: Very recently, in Wrocław and Paderborn, the results of this paper have been significantly improved giving more practical solutions.

References

1. M. Ajtai, J. Komlós and E. Szemerédi. Sorting in $c \log n$ parallel steps. *Combinatorica* **3** (1983) 1-19.
2. K. E. Batcher. Sorting networks and their applications. In *AFIPS Conf. Proc. 32*, pp. 307–314, 1968.
3. M. Dowd, Y. Perl, M. Saks, and L. Rudolph. The periodic balanced sorting network. *J. ACM* **36** (1989) 738–757.
4. M. Kik, M. Kutyłowski, G, Stachowiak. *Periodic constant depth sorting networks.* Technical Report, Heinz–Nixdorf–Institut, Universität Paderborn, September 1993.
5. J. G. Krammer. *Lösung von Datentransportproblemen in integrierten Schaltungen.* Ph.D. Dissertation, Technical University Munich, 1991.
6. F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes* (Morgan Kaufmann, San Mateo, 1992).
7. F. Meyer auf der Heide. *Personal communication*, 1991.
8. U. Schwiegelshohn. A shortperiodic two-dimensional systolic sorting algorithm. In *IEEE International Conference on Systolic Arrays*, pp. 257-264, 1988.