Improving the Average Delay of Sorting*

Andreas Jakoby[1] **Maciej Liśkiewicz**[1]
Rüdiger Reischuk[1] **Christian Schindelhauer**[2]

[1] Inst. für Theoretische Informatik, Universität zu Lübeck, Germany
jakoby/liskiewi/reischuk@tcs.mu-luebeck.de
[2] Dept. of Computer Science, Universität Freiburg, schindel@informatik.uni-freiburg.de

Abstract. In previous work we have introduced an average-case measure for the time complexity of Boolean circuits – that is the delay between feeding the input bits into a circuit and the moment when the results are ready at the output gates – and analysed this complexity measure for prefix computations. Here we consider the problem to sort large integers that are given in binary notation. Contrary to a *word comparator sorting circuit* C where a basic computational element, a comparator, is charged with a single time step to compare two elements, in a *bit comparator circuit* C' a comparison of two binary numbers has to be implemented by a Boolean subcircuit `CM` called *comparator module* that is built from Boolean gates of bounded fanin. Thus, compared to C, the depth of C' will be larger by a factor up to the depth of `CM`.
Our goal is to minimize the average delay of bit comparator sorting circuits. The worst-case delay can be estimated by the depth of the circuit. For this worst-case measure two topologically quite different designs seems to be appropriate for the comparator modules: a tree-like one if the inputs are long numbers, otherwise a linear array working in a pipelined fashion. Inserting these into a word comparator circuit we get bit level sorting circuits for binary numbers of length m for which the depth is either increased by a multiplicative factor of oder $\log m$ or by an additive term of order m.
We show that this obvious solution can be improved significantly by constructing efficient sorting and merging circuits for the bit model that only suffer a constant factor time loss on the average if the inputs are uniformly distributed. This is done by designing suitable hybrid architectures of tree compaction and pipelining. These results can also be extended to classes of nonuniform distributions if we put a bound on the complexity of the distributions themselves.

1 Introduction

For circuits, depth is normally used to measure the time a computation takes. This is a worst case estimation. In [JRS94] we have defined an average-case measure for the time complexity of circuits called **delay**. It has been observed that in many cases critical paths of a given circuit, e.g. paths between input and output gates of maximal length,

* Supported by DFG research grant Re 672/3.

have no influence on the final output. Hence, the output values of the circuit for some inputs can be obtained much earlier.

The average delay of basic functions like OR, ADDITION, PARITY, and THRESHOLD has been estimated precisely. These are special instances of the parallel prefix problem that has been investigated in detail in [J98]. In many cases we have found circuit designs that are exponentially faster on average than the optimal circuits for the worst-case [JRS94,JRSW94,JRS95]. On the other hand, we could show lower bounds saying that for certain functions, e.g. PARITY, the average delay remains asymptotically the same as in the worst case. A similar result holds for the problem to sort n bits that has worst-case complexity $\Theta(\log\, n)$. For the worst case, the lower bound follows from a simple counting argument, the upper bound has been established by a nontrivial construction of Ajtai, Komlos and Szemeredi [AKS83].

The delay of a sorting circuit may be smaller than its depth as can be seen in Fig. 1 showing a sorting circuit C_3 for 3 elements. The first picture shows the circuit consisting of 3 comparators A, B, C. Its depth is 3, too, since the line in the middle marked with input y goes through each comparator. The pictures show the flow of the inputs through the circuit starting with time $t = 0$ when all inputs are at the left end. However, for the given input vector $(1, 0, 0)$, on the critical path in the middle there does not occur a delay of 3. The reason is as follows: already in the first time step the lower 0 can be passed through comparator B to its upper output line although the second input for B has not arrived yet. No matter, what kind of bit this will be, comparator B can be set to an X that switches the inputs because we can be sure that a 0 must occur at the upper output. In the second phase this allows comparator C to do its job since both its inputs are already there and comparator B to finish its work by passing the input 1 on its upper line to the lower output line. Still, this saving in the computation time has no asymptotic effect as we have shown in [JRS94].

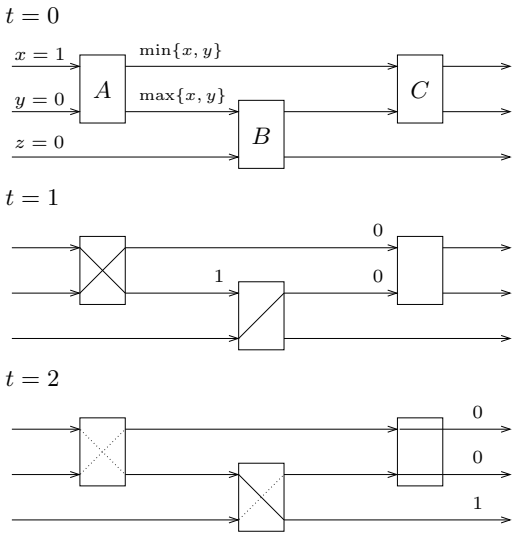

Fig. 1. *The flow of inputs in a computation of* C_3.

Fact 1 *The average delay of a sorting circuit over an arbitrary finite basis with gates of bounded fanin that sorts* n *uniformly distributed bits is at least* $\Omega(\log\, n)$.

Thus, sorting n elements requires logarithmic time - even on the average. The complexity of sorting seems to be settled. But what happens if we do not have to sort single bits, but long binary numbers. Obviously, the depth has to increase since a binary circuit of bounded fanin cannot compare two long numbers in constant time.

Let n denote the number of elements that have to be sorted and let us start with a **word comparator circuit C_n**. A **comparator** has indegree and outdegree 2, and takes two elements of the sorting domain and outputs the minimum at the top and the maximum at the bottom output node. Let $\texttt{depth}(C_n)$ denote the depth of the circuit where each comparator is assumed to have depth 1 (see Fig. 1).
If the elements to be compared are binary numbers of some length m we call this the **(n, m)-sorting problem**. Now let us consider the physical realization of comparators. A comparator $\texttt{CM}_m$ that compares two m-bit numbers has to be built on the bit level. Such a subcircuit we will call a **comparator module**. There are two obvious alternatives how to design a comparator module and combine it with the topology of a word comparator circuit.

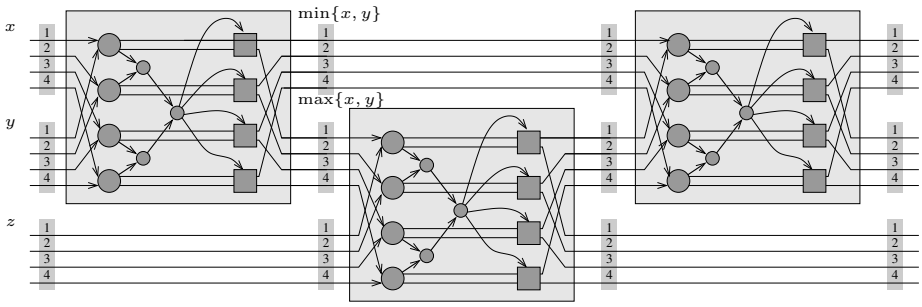

Fig. 2. *The sorting circuit $C_{3,4}$ with comparators of a tree architecture.*

On the one hand, one can compare two numbers x, y bitwise. Every bit comparison generates a result $<, =$ oder $>$ and these results can be combined in binary tree-like fashion to determine the result ρ determining which number is the smaller or whether the numbers are equal. Each pair of bits of x, y is then routed to the appropriate output position by a switch that is driven by ρ. Assume that the combination of two bit comparison results can be performed by a subcircuit of depth δ. Then such a comparator $\texttt{CM}_m$ can be implemented by a binary circuit of depth $\delta \log\, m + O(1)$. This assumes no bound on the fanout of a Boolean gate (if one insists on fanout at most 2 the depth becomes $(\delta + 1) \log\, m + O(1)$). Thus, in total we get a bit level sorting circuit $C_{n,m}$ of depth $(\delta \cdot \log\, m + O(1)) \cdot \texttt{depth}(C_n)$ (see Fig. 2).
Alternatively, one could compare the bits of two numbers in a linear fashion starting with the leading bits. This requires depth linear in the number of bits, but has the advantage that after δ steps the leading bits of the two results of that comparator are already known. This pipelined construction increases the depth of the sorting circuit C_n only by an additive term $\delta \cdot m$ resulting in a bit level sorting circuit $C'_{n,m}$ of depth at most $\delta \cdot (m + \texttt{depth}(C_n) - 1)$ (see Fig. 3).
A detailed discussion of sorting in the bit model can be found in Section 1.1.2 of [L92]. Several papers have considered worst-case delay of Boolean sorting networks explicitly. In [AB93] Al-Hajery and Batcher constructed *bit serial bitonic sorting networks* (BBSN) of size $O(n \log\, n)$ that sort n numbers each of length $m = O(\log\, n)$ in

$O(\log^2 n)$ steps. BBSN is a *periodic* network of depth $O(\log\, n)$ and size $O(n \log\, n)$ based on pipelining. The model of [AB93] differs, however, from the word comparator circuit in that BBSN is a network of bit processors which has the same topology as bitonic sorting network and which processes m-bit input strings in a bit serial fashion. In [LP90] Leighton using methods due to Thompson [T79] proved an $\Omega(n+m)$ lower time bound for (n, m)-sorting on a $(m \times n)$–array of bit processors. In addition, several papers have discussed VLSI architectures for sorting (e.g. [T83,LO99,HL00,LDH03]).

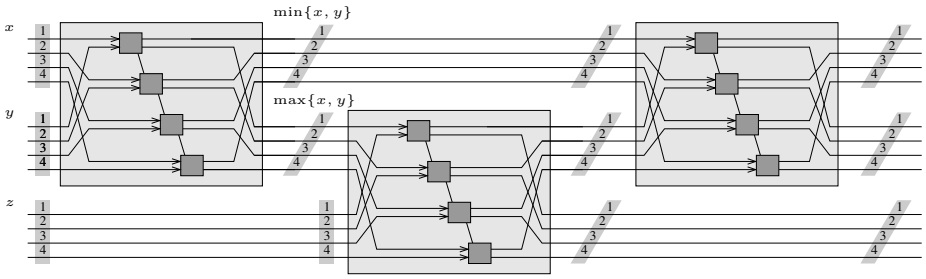

Fig. 3. *The sorting circuit $C'_{3,4}$ based on linear arrays. The gray boxes on the links that connect the comparator modules illustrate that output bits of higher order are available before the output bits of lower order.*

Using the topology of the asymptotically optimal AKS-network [AKS83] and noticing that $\delta = O(1)$, the pipelined construction by linear arrays described above sorts n numbers of length $m \leq O(\log\, n)$ in $O(\log\, n)$ depth. For $m \geq \log\, n\ \text{loglog}\, n$, tree-like comparator modules seem to be better suited and give depth $O(\log\, n\ \log\, m)$. Because of huge constants, AKS-networks are only advantageous for very large n.
In [LP90] Leighton and Plaxton constructed *butterfly*-based sorting networks that sort correctly with probability close to 1. These networks have depth $7.45 \log\, n$. An implementation of this topology with binary gates yields a randomized sorting circuit of $O(m + \log\, n)$, resp. $O(\log\, n\ \log\, m)$ depth with small constant factors and low error probability.
In this paper we investigate the (n, m)-sorting problem for m significantly larger than $\log\, n$. New comparator modules that are hybrid versions of the two basic topologies will be constructed that speed up sorting networks on average assuming uniformly distributed inputs.

Theorem 1. *For every m, there are comparator modules $\texttt{CM}_m$ with the following property: If a Boolean circuit $C_{n,m}$ for the (n, m)-sorting problem is derived from a word comparator circuit C_n by implementing its comparators as $\texttt{CM}_m$ modules then assuming uniformly distributed inputs $C_{n,m}$ faces a delay of at most $O(\texttt{depth}(C_n))$ with probability at least $1 - 1/n$. Even in the worst case, the delay does not exceed $O(\texttt{depth}(C_n) \cdot \log\,(n + m))$. Thus, the average delay is at most a constant factor larger than the depth of the word comparator circuit independent of the length of the binary numbers.*

The proof will be given in Section 4. This construction requires gates of unbounded fanout to spread information about comparison results fast. If one requires a constant fanout restriction the delay bound becomes $O(\mathtt{depth}(C_n) + \log\, m)$ with probability $1 - 1/n$ and the average delay stays independent of m as long as $m \leq 2^n$. Thus even in case of strictly bounded fanout, for large numbers m we achieve the best combination of the simple architectures described above concerning the average delay: only a logarithmic increase $\log\, m$ instead of m, and this only by an additive term rather than a multiplicative factor.
Small average delay can also be achieved for merging lists of m-bit numbers. In particular, based on the odd-even merge topology we show that there exists a bit comparator circuit that merges two lists of $n/2$ numbers each in $O(\log\, n + \log\, m)$ steps on average. As a consequence we obtain

Theorem 2. *Let $\mathcal{M}_n$ be the odd-even merge sort word comparator circuit for n elements. Then for every m, there are comparator modules $\mathtt{CM}_m$ such that the Boolean circuit derived from $\mathcal{M}_n$ by replacing its comparators by $\mathtt{CM}_m$ modules solves the (n, m)-sorting problem and with probability at least $1 - 1/n$ its delay is bounded by $O(\log^2 n)$.*

Furthermore, the average delay of these circuits is bounded by $O(\log^2 n)$ and their worst-case behaviour is as good as that of worst-case optimal ones. This small average delay bound can also be obtained for families of nonuniform distributions of low complexity (Theorem 4 below). For this result our circuit design does not use any knowledge about the actual distribution μ, it works uniformly for all such μ.

The rest of this paper is organized as follows. Section 2 defines asynchronous Boolean circuits and their timing, in particular the complexity measure delay. The design of efficient comparator modules is described in Section 3. In Section 4 we construct specific bit comparator circuits for sorting and merging and analyse their average delay for the uniform distribution. This is extended to nonuniform distributions in Section 5.

2 Timing of a Boolean Circuit

In the following let $\log\, n := \lceil \log_2 n \rceil$ denote the binary logarithm rounded up.
If we want to exploit possibilities to speedup the computation of a Boolean circuit it has to work in an asynchronous fashion. For this mode one has to extend the binary logic to indicate when a Boolean value is *ready* of *valid.* How this can be done efficiently has been discussed in [JRS94]. To concentrate on the topological aspects of sorting circuits here we simply assume that each gate knows from its predecessors when their values are ready.
Let C be a Boolean circuit, and $V_{\text{in}},\ V_{\text{out}}$ denote its input, resp., output gates. For a gate g and input x of C let $\text{res}_g(x)$ denote the value that is generated by g on x. If g is the i-th input gate then $\text{res}_g(x) = x_i$. Otherwise, $\text{res}_g(x)$ is determined by the values $\text{res}_{g_i}(x)$ of its immediate predecessors g_i and the type of g.
Circuits that work in an asynchronous mode may not get all their input bits at the same time. Similar to [JS01] we therefore make the following definitions.

Definition 1. A **starting-line** for C is a function $\mathcal{S} : V_{\text{in}} \to \mathbb{N}$. Given a starting-line $\mathcal{S}$ for C, we define a function $\mathtt{time}^C_{\mathcal{S}}$ for pairs (g, x) where g is a gate of C and x an input as follows:

$$\mathtt{time}^C_{\mathcal{S}}(g, x) \;:=\; \begin{cases} \mathcal{S}(g) & \text{if } g \text{ is an input gate,} \\ 0 & \text{if } g \text{ is a constant gate,} \\ 1 + t_g(x) & \text{else,} \end{cases}$$

where $t_g(x)$ denotes the smallest time t, such that the values $\text{res}_{g_i}(x)$ of those immediate predecessors g_i of g with $\mathtt{time}^C_{\mathcal{S}}(g_i, x) \le t$ uniquely determine $\text{res}_g(x)$.
Thus, $\mathtt{time}^C_{\mathcal{S}}(g, x)$ denotes the earliest moment when g knows its value assuming that the inputs are available according to the starting time $\mathcal{S}$. For the circuit C itself we define the timing by $\mathtt{time}^C_{\mathcal{S}}(x) \;:=\; \max_{g \in V_{\text{out}}} \mathtt{time}^C_{\mathcal{S}}(g, x)$.
Let $\mathtt{time}^C(x)$ denote the timing if the starting-line $\mathcal{S}$ is identically 0.
Given a probability distribution μ on the input space, we define the **average delay of** C by $\quad \mathbf{Etime}_{\mu}(C) \;:=\; \sum_x \mu(x) \cdot \mathtt{time}^C(x)$. □

Normally, all input bits are available at the beginning of a computation, that is at time 0. In the following we will also consider the case when some bits are delayed. For this purpose, for $k \in [1..m]$ let us define the function $\sigma_k : [1..m] \to \mathbb{N}$ by

$$\sigma_k(j) \;:=\; \begin{cases} j - 1 & \text{if } j \le k, \\ k + 1 & \text{else.} \end{cases}$$

3 Average-case Efficient Comparator Modules

Definition 2. Let $\mathbb{B} = \{0, 1\}$ denote the binary alphabet and $\Sigma_\rho := \{\mathtt{LE}, \mathtt{EQ}, \mathtt{GT}\}$ an alphabet to specify the result of a comparison of two elements, numbers or bits: *less, equal, or greater*. Σ_ρ will suitably be coded over $\mathbb{B}$ – for example by the 3 vectors $(1,0,0),\ (0,1,0),\ (0,0,1)$. In the following x, y, u, v will always denote variables that hold a binary value and $\rho, \rho_1, \rho', \ldots$ are variables that take values from Σ_ρ. The Boolean sorting circuits will be constructed from 2 basic types of gates, S-gates and R-gates (see Fig 4).

- ***S*-gate:** it takes 3 inputs ρ, x, y and generates the 3 outputs u, v, ρ'. The input-output relation is defined as follows:

for $\rho = \mathtt{EQ}$ and $x < y$:	$u = \min\{x,y\} = x,$	$v = \max\{x,y\} = y,$	$\rho' = \mathtt{LE},$
for $\rho = \mathtt{EQ}$ and $x > y$:	$u = \min\{x,y\} = y,$	$v = \max\{x,y\} = x,$	$\rho' = \mathtt{GT},$
for $\rho = \mathtt{EQ}$ and $x = y$:	$u = \min\{x,y\},$	$v = \max\{x,y\},$	$\rho' = \mathtt{EQ},$
for $\rho = \mathtt{LE}$:	$u = x,$	$v = y,$	$\rho' = \mathtt{LE},$
for $\rho = \mathtt{GT}$:	$u = y,$	$v = x,$	$\rho' = \mathtt{GT}.$

- ***R*-gate:** the inputs are ρ, ρ_1, ρ_2, the only output is ρ':

for $\rho \neq \mathtt{EQ}$:	$\rho' = \rho,$
for $\rho = \mathtt{EQ}$ and $\rho_1 \neq \mathtt{EQ}$:	$\rho' = \rho_1,$
else	$\rho' = \rho_2.$

□

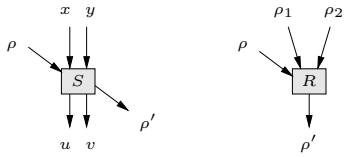

Fig. 4. S-gate and R-gate

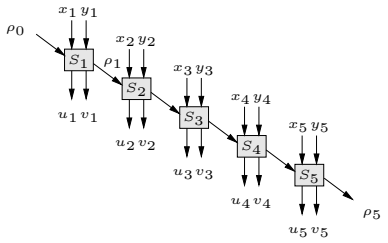

Fig. 5. *A line comparator module.*

According to the Boolean basis and the coding of Σ_ρ both types of gates can be realized by small subcircuits of some fixed depth at most δ. For simplicity, through the rest of the paper we will assume that $\delta = 1$, otherwise one has to add this as a constant factor to all the circuit bounds stated below. For an S-gate it is important to note that depending on its input values, its 3 outputs may be ready at different times. Thus, the timing information should rather be attached to the output wires of a gate than to the gate itself. Since this will not be important in the following we stick to the simpler model.

Our circuit designs will also make use of simplified versions of an S-gate. An L-gate is an S-gate where the ρ'-output is not needed. An U-gate in addition does not need the ρ-input and behaves as if this input were `EQ`. Furthermore, an U-gate does not have the outputs u and v. Also some R-gates will have the input ρ be missing.

A comparator module $\texttt{CM}_m$ is a subcircuit built from S- and R-gates that takes two binary numbers $x = x_1 \dots x_m$ and $y = y_1 \dots y_m$ and produces two output strings u and v such that $u = \min\{x, y\}$ and $v = \max\{x, y\}$. We assume that x_1, resp. y_1 are the leading bits of the binary numbers. In addition, $\texttt{CM}_m$ outputs the result $\rho \in \Sigma_\rho$ of the comparison, that is either `LE`, `EQ` or `GT`. In the following, ρ will be called the **compare info** of $\texttt{CM}_m$. Let us first describe more formally the line- and tree-comparator module introduced above.

Definition 3. A **line comparator module** $\texttt{LCM}_m$ (see Fig. 5) is a comparator module consisting of a linear array of S-gates $S_1, \dots, S_m$ where each S_i gets the i-th bit of x and y and the compare result ρ_{i-1} of S_{i-1}. For S_1 we define $\rho_0 := \texttt{EQ}$. S_i outputs the two bits u_i and v_i and ρ_i as the result of comparing the prefixes $x_1, \dots, x_i$ and $y_1, \dots, y_i$. The compare info of $\texttt{CM}_m$ is ρ_m, i.e. the compare result of the last S_m. □

Even though some of the pairs u_i, v_i may be computed faster (if $x_i = y_i$), since the ρ_i form a linear chain, gate S_i always has to wait for the output ρ_{i-1} of its left neighbour in order to determine its output ρ_i. Thus, we get the following timing for a `LCM`.

Lemma 1. If $\mathcal{S}$ is a starting-time for $\texttt{LCM}_m$ such that $\mathcal{S}(x_i), \mathcal{S}(y_i) \leq i - 1$ for all $i \in [1..m]$ then $\texttt{time}_{\mathcal{S}}^{\texttt{LCM}_m}(S_j, (x, y)) = j$ for all $j \in [1..m]$ and all input pairs (x, y).

Definition 4. A **tree comparator module** $\texttt{TCM}_m$ (see Fig. 6) makes all the comparisons of input pairs x_i, y_i in parallel by a sequence of U-gates $U_1, \dots, U_m$, and then

combines their results $\rho_1, \ldots, \rho_m$ by a binary tree of R-gates to obtain the compare info ρ. The root of this tree will be denoted by $\hat{R}$. The compare info ρ at $\hat{R}$ is then used to drive m L-gates $L_1, \ldots, L_m$. L_i either leads the two inputs x_i, y_i simply through if ρ equals LE or EQ, or exchanges their order otherwise. Value ρ can either be forwarded to the L_i directly if we allow unbounded fanout or we have to use another binary tree to duplicate this information if the fanout is bounded. In the following we will consider the case of unbounded fanout, otherwise in the timing bounds below one has to add another additive term $\log m$. □

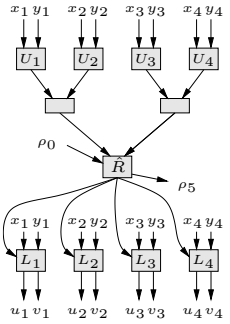

Fig. 6. *A tree comparator module.*

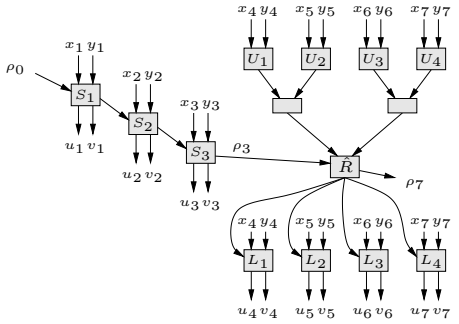

Fig. 7. *A line tree comparator module.*

Lemma 2. For arbitrary $\mathcal{S}$ and all inputs (x, y) it holds for all $j \in [1..m]$

$$\mathtt{time}_{\mathcal{S}}^{\mathtt{TCM}_m}(L_j, (x, y)) \;=\; 2 + \log\, m + \max\{\mathcal{S}(x_i),\, \mathcal{S}(y_i) \mid i \in [1..m]\}\;.$$

To be more efficient in the average case we now define hybrid versions of these two architectures. They will depend on an additional parameter $k \in \mathbb{N}$. When applying to the sorting problem of n elements k typically will be of order $\log n$.

Definition 5. A ***k*-line tree comparator module** (see Fig. 7), $\mathtt{LTCM}_{m,k}$ for short, consists of a line comparator module $\mathtt{LCM}_k$ for the prefixes $x_1, \ldots, x_k$ and $y_1, \ldots, y_k$, and a tree comparator module $\mathtt{TCM}_{m-k}$ for the suffixes $x_{k+1}, \ldots, x_m$ and $y_{k+1}, \ldots, y_m$ with the following modification. The root $\hat{R}$ of the tree comparator additionally gets the compare info ρ_k of $\mathtt{LCM}_k$ and if this result is EQ then it works as previously. Otherwise, $\hat{R}$ outputs this value as a result of the comparison between x and y and propagates this value to the L_i. □

A combination of the two timing bounds for LCM and TCM modules gives

Lemma 3. For all starting-lines $\mathcal{S}$ with $\mathcal{S}(x_i), \mathcal{S}(y_i) \le \sigma_k(i)$ and all input pairs (x, y) it holds $\mathtt{time}_{\mathcal{S}}^{\mathtt{LTCM}_{m,k}}(S_j, (x, y)) \;\le\; j$ for $j \le k$ and

$$\mathtt{time}_{\mathcal{S}}^{\mathtt{LTCM}_{m,k}}(L_j, (x, y)) \;\le\; \begin{cases} k + 2 + \log\,(m-k) & \text{for } \rho_k = \mathtt{EQ}, \\ k + 2 & \text{for } \rho_k \neq \mathtt{EQ}. \end{cases}$$

To achieve small average-case delay for nonuniform distributions we need another type of comparator module that can be found in the full paper.

4 Average-case Delay for the Uniform Distribution

The previous section has shown that the delay of a comparator module depends on the length of the prefix up to which its two inputs x and y are identical. Therefore, we make the following definition.

Definition 6. Let $X = X^1, \ldots, X^n$ be a sequence of strings with $X^i = x_1^i \ldots x_m^i \in \{0,1\}^m$, $c \in \mathbb{N}$, and $w \in \{0,1\}^c$. We call w a **conflict prefix** of X if X contains two string X^i, X^j $(i \neq j)$ with prefix w. Let $\mathtt{conf}_c(X)$ denote the number of different conflict prefixes in X of length c.
A $\boldsymbol{c}$**-congestion** of X is a subsequence of X such that all its members have identical prefixes of length c. Let $\mathtt{con}_c(X)$ denote the maximal size (number of elements of the subsequence) of a c-congestion of X.

Obviously, the values $\mathtt{conf}_c(X)$ are monotonically decreasing with c. If $\mathtt{con}_c(X) = 1$ then the strings in X have pairwise different prefixes of length c.
In this section we assume that $\mathcal{X}^{n,m}$ is a uniformly distributed random variable generating an independent sequence $X^1, \ldots, X^n$ of binary numbers of length m each. We can upperbound conflicts and congestion as follows.

Lemma 4. [Conflict Prefix and Congestion Bound] For every $c,\ \beta,\ \gamma \in \mathbb{N}$ it holds:

$$\Pr[\mathtt{conf}_c(\mathcal{X}^{n,m}) \geq \beta] \ \leq\ 2^{-\beta(c-2\log\ n)}$$

and

$$\Pr[\mathtt{con}_c(\mathcal{X}^{n,m}) \geq \gamma] \ \leq\ 2^{-\gamma(c-\log\ n)+c}\ .$$

We will use k-line tree comparators with different parameters k. Circuits of such comparator modules work efficiently if the k-congestion of the input strings is small. From the lemma above follows that for $c \geq 3\log\ n$ the c-congestion does not exceed 1 with high probability, in particular $\Pr[\mathtt{con}_{3\log\ n}(\mathcal{X}^{n,m}) > 1] \ \leq\ 1/n$.
On the other hand for $c \leq (1-\epsilon)\log\ n$ with $\epsilon > 0$, the c-congestion may typically be quite large. Hence, our circuit designs will choose the parameter k in the interval $[2\log\ n + \varepsilon..3\log\ n]$. For the rest of this section we will choose $k\ :=\ 3\log\ n$ and assume that $m \geq k$ is large enough.
Let X^i, X^j be inputs of an $\mathtt{LTCM}_{m,k}$. If the prefixes of length k of X^i, X^j are different then the module can obtain the compare info in $O(k)$ steps. In this case, we say that it *gets the result fast,* otherwise it gets the result *slowly*.
Using the function σ_k introduced at the end of Section 2, we define a starting line $\mathcal{S}_k$ for a sorting circuit C as follows. For $i \in [1..n]$ and $j \in [1..m]$ let $x_{i,j}$ denote the gate that gets the j-th input bit of the i-th number X^i, and $y_{i,j}$ the corresponding output gate. Then $\mathcal{S}_k(x_{i,j}) \ =\ \sigma_k(j)$. Note that an input sequence $X = X^1, \ldots, X^n$ with $\mathtt{con}_k(X) = 1$ can be sorted by comparing the prefixes of length k and exchange the remaining part of the strings according to the compare info of these prefixes.

Lemma 5. Let C be a circuit for the (n,m)-sorting problem that is obtained from an arbitrary word comparator circuit C_n by implementing its comparators as $\texttt{LTCM}_{m,k}$. Then with probability at least $1-1/n$, for every output gate $y_{i,j}$ of C it holds $\texttt{time}^C_{\mathcal{S}_k}(y_{i,j},\mathcal{X}^{n,m}) \leq \sigma_k(j)+\texttt{depth}(C_n)$.

From this lemma we get that all output gates can compute their values by time step $k+\texttt{depth}(C_n)+1$. This proves Theorem 1.
For circuits with fanout at most 2 one obains a slightly worse estimation of the form

$$\texttt{time}^C_{\mathcal{S}_k}(y_{i,j},\mathcal{X}^{n,m}) \leq \begin{cases} \sigma_k(j)+\texttt{depth}(C_n) & \text{if } j \leq k, \\ \sigma_k(j)+\texttt{depth}(C_n)+\log(m-k) & \text{else.} \end{cases}$$

In the rest of this section we will concentrate on particular sorting and merging circuits, namely on odd-even merge and bitonic architectures. We start by considering the (n,m)-merging problem for binary numbers of length m.

Lemma 6. Let C_n be an odd-even-merge word comparator circuit merging two sorted m-bit sequences of $n/2$ elements each. For $k \leq m$ let $C_{n,m,k}$ be derived from C_n by replacing its comparators by $\texttt{LTCM}_{m,k}$. Then for every integer $\gamma \geq 1$, every input X with $\texttt{con}_{k+1}(X)=\gamma$ and $\texttt{conf}_{k+1}(X)=\beta$ and for every output gate $y_{i,j}$ of C it holds

$$\texttt{time}^{C_{n,m,k}}_{\mathcal{S}_k}(y_{i,j},X) \leq \begin{cases} \sigma_k(j)+\log n & \text{if } j \leq k, \\ \sigma_k(j)+\log n+\log(m-k)\cdot(\beta+\log\gamma) & \text{else.} \end{cases}$$

The proof of the lemma above is based on the following properties of odd-even-merge circuits:

- Let X be an input of length n for odd-even -merge and X' be one of the two sorted subsequences of length $n/2$. Then within the first ℓ steps of the recursive problem division X' is partitioned into 2^ℓ subsequences $X'_1,\ldots,X'_{2^\ell}$.
- Let $B_1,\ldots,B_r$ be a partition of X' into consecutive strings. After $\log\max_i|B_i|$ recursive steps every subsequence X'_i contains at most one element from each B_j.
- Every pair of input strings X^i and X^j of X is compared at most once.

Theorem 3 (Odd-Even Merge). Let C_n be an odd-even-merge word comparator circuit merging two sorted m-bit sequences of $n/2$ elements. Let $C_{n,m,k}$ a Boolean circuit derived from C_n by implementing its comparators as $\texttt{LTCM}_{m,k}$ modules. Given a sequence $\mathcal{X}^{n,m}$, let $Z_1,\ldots,Z_{n/2}$ be a permutation of the subsequence $X^1,\ldots,X^{n/2}$ sorted in nondecreasing order, and similarly $Z^{n/2+1},\ldots,Z^n$ for $X^{n/2+1},\ldots,X^n$. Then with probability at least $1-1/n$: $\texttt{time}^C(Z^1,\ldots,Z^n) \leq 5\cdot\log n$.

The proof follows from the Congestion-Bound and the lemma above. This theorem implies also the result for the sorting problem as stated in Theorem 2 in Section 1. A similar bound can be obtained for bitonic circuits.

5 Average-case Delay for Nonuniform Distributions

This section will extend the previous results to nonuniform distributions. We have to bound the complexity of distributions somehow, because otherwise the average case would equal the worst case. This will be done within the circuit model itself.

Definition 7. A *distribution generating circuit* is a Boolean circuit D of fanin and fanout at most 2. If D has r input gates and n output gates it performs a transformation of a random variable $\mathcal{Z}$ uniformly distributed over $\{0,1\}^r$ into a random variable $\mathcal{X}$ over $\{0,1\}^n$. The input vector for D is chosen according to $\mathcal{Z}$, and the distribution of $\mathcal{X}$ is given by the distribution of the values obtained at the output gates. □

In the following we will identify a distribution over $\{0,1\}^{n\cdot m}$ with a corresponding random vector variable $\mathcal{X}$. Let $\mathcal{X} = (X^1, \ldots, X^n)$ with $X^i = X^i_1 \ldots X^i_m \in \{0,1\}^m$.

Definition 8. Let $\mathcal{D}_{n,m}$ denote the set of all probability distributions μ on $\{0,1\}^{n\cdot m}$. For $\mu \in \mathcal{D}_{n,m}$ let $\mathrm{Supp}(\mu)$ be the set of all vectors $X \in \{0,1\}^{n\cdot m}$ with nonzero probability $\mu(X)$. We call a distribution in $\mathcal{D}_{n,m}$ *strictly positive* if $\mathrm{Supp}(\mu) = \{0,1\}^{n\cdot m}$ and let $\mathcal{D}^+_{n,m}$ denote the set of such distributions. Finally define

$$\begin{aligned}\mathtt{Depth}_{n,m}(d) \;:=\; \{\mu \in \mathcal{D}^+_{n,m} \mid &\exists \text{ an } r\text{-input and } (n\cdot m)\text{-output Boolean circuit } D\\ &\text{of depth } d \text{ that transforms a uniformly distributed random}\\ &\text{variable } \mathcal{Z} \text{ over } \{0,1\}^r \text{ into a random variable } \mathcal{X} \text{ with}\\ &\text{distribution } \mu, \text{ where } r \text{ may be any integer}\}\ .\end{aligned}$$ □

By definition, $\mathtt{Depth}_{n,m}(d)$ contains strictly positive probability distributions only. In our setting where a single circuit should have good average-case behaviour for every distribution in this class this is obviously necessary to exclude trivial cases. Otherwise one could concentrate the probability mass on the worst-case inputs and average-case complexity would equal worst-case complexity. The same problem would arise if the distribution generating circuits may use gates of unbounded fanin or fanout.
To guarantee small average delay the congestion has to be low as seen above. Below we establish a bound on the congestion of a random variable generated by a circuit of small depth.

Lemma 7. Let $\mathcal{X} \in \mathtt{Depth}_{n,m}(d)$ and $c \geq 3 \cdot 2^{2^{d+1}+2d+1} \log n$. Then it holds $\Pr[\mathtt{con}_c(\mathcal{X}) \geq 2] \leq \frac{1}{n}$ and $\Pr[\mathtt{conf}_c(\mathcal{X}) \geq 1] \leq \frac{1}{n}$.

For small d, i.e. $d = \mathrm{loglog} n$, the bound given in Lemma 7 implies $\Pr[\mathtt{con}_c(\mathcal{X}) \geq 2] \leq \frac{1}{n}$ for $c \in \Theta(\log^2 n \cdot \mathrm{loglog}\, n)$. One should note that even with such a small depth bound d one can construct highly biased bits x (for example such that $\Pr[x = 1] = 1/\log n$) and also a lot of dependencies among subsets of bits.

Theorem 4. Let $C_{n,m}$ be a Boolean circuit for the (n,m)-sorting problem derived from the word comparator odd-even merge sort circuit C_n by replacing its comparators by a specific family of comparator modules CM. Then for $\mathcal{X} \in \mathtt{Depth}_{n,m}(\mathrm{loglog}\log n)$, with probability greater than $1-1/n$ it holds $\mathtt{time}^{C_{n,m}}(\mathcal{X}) \leq 5\ \log^2 n\ \mathrm{loglog}\, n$.

That a tiny depth bound is indeed necessary can be seen as follows. For $d = \text{loglog}\, n$ one can construct $\mathcal{X} \in \texttt{Depth}_{n,m}(d)$ such that $\Pr[\texttt{con}_{m^{\varepsilon/2}}(\mathcal{X}) \geq n^{\varepsilon/2}] \geq \frac{1}{2}$ for some $\varepsilon > 0$. In this case a larger delay has to occur even in line tree comparator modules.

6 Conclusion

We have presented new topologies for bit level comparators. Using these modules to replace the comparators of a word level sorting circuit yields sorting circuits that are highly efficient on the average. For odd-even sorting circuits we could show that one can achieve an average-delay on the bit level that is asymptotically the same as on the word level.
The question arises whether simular results can be shown for other computational problems that can be realized on the word as well as on the bit level.

References

[AKS83] M. Ajtai, J. Komlos, and E. Szemeredi, *Sorting in $c \log n$ parallel steps*, Combinatorica 3, 1983, 1-19.

[AB93] M. Al-Hajery and K. Batcher, *On the bit-level complexity of bitonic sorting networks*, Proc. 22. Int. Conf. on Parallel Processing, 1993, III.209 – III.213.

[HL00] I. Hatirnaz and Y. Leblebici, *Scalable binary sorting architecture based on rank ordering withlinear area-time complexity*, Proc. 13. IEEE ASIC/SOC Conference, 2000, 369-373.

[J98] A. Jakoby, *Die Komplexität von Präfixfunktionen bezüglich ihres mittleren Zeitverhaltens,* Dissertation, Universität zu Lübeck, 1998.

[JRS94] A. Jakoby, R. Reischuk, and C. Schindelhauer, *Circuit complexity: from the worst case to the average case*, Proc. 26. ACM STOC, 1994, 58-67.

[JRS95] A. Jakoby, R. Reischuk, and C. Schindelhauer, *Malign distributions for average case circuit complexity*, Proc. 12. STACS, 1995, Springer LNCS 900, 628-639.

[JRSW94] A. Jakoby, R. Reischuk, C. Schindelhauer, and S. Weis, *The average case complexity of the parallel prefix problem*, Proc. 21. ICALP, 1994, Springer LNCS 820, 593-604.

[JS01] A. Jakoby, C. Schindelhauer, *Efficient Addition on Field Programmable Gate Arrays*, Proc. 21. FSTTCS, 2001, 219-231.

[LDH03] Y. Leblebici, T. Demirci, and I. Hatirnaz, *Full-Custom CMOS Realization of a High-Performance Binary Sorting Engine with Linear Area-Time Complexity,* Proc. IEEE Int. Symp. on Circuits and Systems 2003.

[L92] T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[LP90] T. Leighton and C. G. Plaxton, *A (fairly) simple circuit that (usually) sorts*, Proc. 31. IEEE FOCS, 1990, 264-274.

[LO99] R. Lin and S. Olariu, *Efficient VLSI architecture for Columnsort*, IEEE Trans. on VLSI 7, 1999, 135-139.

[T79] C.D. Thompson, *Area-Time Complexity for VLSI*, Proc. 11. ACM STOC 1979, 81-88.

[T83] C.D. Thompson, *The VLSI Complexity of Sorting*, IEEE Trans. Comp. 32, 1983, 1171-1184.