MAX-PLANCK-INSTITUT FÜR INFORMATIK

Lower Bounds for Merging on the Hypercube

Ch. Rüb

MPI–I–93–148 October 1993

Im Stadtwald
66123 Saarbrücken
Germany

Lower Bounds for Merging on the Hypercube

Ch. Rüb

MPI-I-93-148 October 1993

Lower Bounds for Merging on the Hypercube
or
Why is Optimal Deterministic Sorting on the Hypercube Difficult?

Christine Rüb
Max-Planck Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany

Abstract We show lower bounds for the problems of merging two sorted lists of equal length and sorting by repeatedly merging pairs of sorted sequences on the hypercube. These lower bounds hold on the average for any ordering of the processors of the hypercube.

Key Words Hypercube, Merging, Sorting, Lower bounds.

1. Motivation and Introduction

The development of a deterministic algorithm for sorting on the hypercube that is work-optimal[1] and runs in polylogarithmic time is a long-standing open problem and has attracted considerable interest (see, e.g. [AH88], [CyP90], [CyS92], [LP90], [P89], [P92]). There are, however, several work-optimal deterministic algorithms for sorting on the PRAM-model that run in polylogarithmic time (see, e.g. [BN89], [C88], [HR89], [K83]). All of these algorithms have in common that they sort by repeatedly merging pairs of sorted sequences. Thus a question to ask is: is the same possible for the hypercube, i.e. is it possible to merge two sorted lists of altogether n elements using $O(n)$ work and polylog(n) time? In this paper we show that this is not the case.

In particular we show the following: Let $\mathcal{H}$ be an n-node hypercube with processors $P_0,\ldots,P_{n-1}$. Let A and B be two sorted sequences of length $n/2$ each such that the elements of A (B, resp.,) are stored in a sorted order at the first (last, resp.,) $n/2$ processors of $\mathcal{H}$. Assume that we want to merge A and B, i.e. we want to rearrange the elements in $A \cup B$ such that the i-th largest element in this sequence is stored at the i-th processor of $\mathcal{H}$. Then there are instances of A and B such that the number of traversed edges, summed over all elements in $A \cup B$, is at least $(n \log n)/2$. Since in each step a maximum of n edges can be used (one per processor), this means that $(\log n)/2$ steps are necessary to merge the two sequences. Thus sorting n elements by repeatedly using ("standard") two-way merging requires work $\Omega(n(\log p)^2)$ on a p-node hypercube, and cannot be optimal unless $p = O(2^{\sqrt{\log n}})$ and $t = \Omega((n \log n)/2^{\sqrt{\log n}})$. This also rules out a pipelined merge-sort as in the PRAM algorithm of [C88].

However, does this really show that sorting by (two-way) merging cannot be optimal? Above we required that the i-th element of A resides initially at the i-th processor of $\mathcal{H}$ and the

[1] A parallel algorithm is work-optimal if the product of the number of processors used and the running time is of the same order as the running time of the best sequential algorithm for the same problem.

This work was supported by the DFG, SFB 124, TP B2, VLSI Entwurfsmethoden und Parallelität.

i-th element of B at the $(n/2+i)$-th processor of $\mathcal{H}$, and that at the end the i-th element of $A \cup B$ resides at the i-th processor of $\mathcal{H}$, i.e. we assumed that the processors are ordered by their indices. So, what happens if we use a different ordering of the processors? Can we find an ordering of them such that two-way merging of n elements on an n-node hypercube can be done in $o(n \log n)$ work? (In this paper work means the total number of executed steps; idle time of processors is not counted.) If this were the case, we could, perhaps, use this ordering to sort the given elements in $O(n \log n) + o(n(\log n)^2)$ work and then rearrange the elements according to the standard ordering of processors in additional $O(n \log n)$ work, see, e.g. [L92], pp. 451 ff. Alas, it is of no use to change the ordering of the processors: we will show that two-way merging of n elements on an n-node hypercube needs at least work $(n \log n)/4$ in the worst case and $\Omega(n \log n)$ on the average (over all possible outcomes of the merging) regardless of the ordering of the processors, and that sorting n elements by two-way merging needs work $\Omega(n(\log p)^2)$ on a p-node hypercube if $p \geq n^{0.5+\epsilon}$ for every $\epsilon > 0$. The latter holds even if the ordering of the processors changes in each step of the recursion, as long as these orderings are independent of the input.
Note that the almost optimal algorithm in [CyP90] that sorts n elements on an n-node hypercube in time $O(\log n \log\log n)$ does this by merging $\sqrt{n}$ sorted lists of $\sqrt{n}$ elements each.

To prove the above lower bounds we make use of the fact that we want to compute the elements in $A \cup B$ in a certain order, i.e. that we want to rearrange the input elements. If we are content with computing the rank of each element in $A \cup B$, this is no longer the case. Nevertheless, we will show similar lower bounds for computing only the ranks of the elements.

We are not aware of any other work on this subject. For lower bounds on the size of comparator networks for merging we refer to [MPT92].

This paper is organized as follows. Section 2 explains the general idea, Section 3 shows that there are inputs that cause high running times, Section 4 shows that the average running time for merging is high, and Section 5 considers the problem of computing the ranks.

2. The General Idea

Consider the problem of merging two lists of $n/2$ elements each on an n-node hypercube such that each processor holds one element. The ordering of the processors is arbitrary. Then there are clearly two input sequences such that for at least one element in them the hamming distance between the processors that hold it before and after the merging, say processors P_i and P_j, is $\Omega(\log n)$. Thus this element has to travel across $\Omega(\log n)$ edges and it will take $\Omega(\log n)$ time till the last element has reached its destination. Next consider the case where each processor holds $m > 1$ elements and the elements at each processor form a consecutive subsequence. We can construct two input sequences such that all elements stored at P_i have to travel to P_j. But that does not necessarily mean that merging these two sequences needs time $\Omega(m \log n)$; rather, by pipelining the movement of the elements at P_i,

it could be possible to achieve a running time of $O(m+\log n)$. Thus the above argument is too weak.

Instead we will use arguments of the following kind: Let $\mathcal{H}$ be an n-node hypercube. Consider the problem of merging two sorted sequences of $n/2$ elements each, one per processor, using any ordering of the processors. Then for a constant fraction of the input elements the average distance (over all possible outcomes of the merging) this element has to travel is $\Omega(\log n)$. Thus the average number of edges crossed by all elements together is $\Omega(n\log n)$ and for a constant fraction of all possible inputs the total number of crossed edges is $\Omega(n\log n)$. Since in each step at most n edges can be used, the running time for these inputs is $\Omega(\log n)$. A similar argument shows that merging mn, $m \geq 1$, elements on an n-node hypercube needs time $\Omega(m\log n)$ and that pipelining cannot improve this. Since the lower bound on the average number of crossed edges still holds if n', $n' \leq n$, elements reside at n' nodes of an n-node hypercube, $n = O((n')^c)$ for a constant c, this means that sorting n elements on a p-node hypercube by repeated two-way merging needs time $\Omega((n/p)(\log p)^2)$ and work $\Omega(n(\log p)^2)$ if $p \geq n^{0.5+\epsilon}$ for any constant $\epsilon > 0$.

3. Expensive Inputs

In this section we consider the problem of merging n elements on an n-node hypercube and show that there are inputs that cause a high running time. Namely we show that for every ordering of the processors there is an input with a running time of at least $(\log n+\sqrt{\log n/2})/4$, and for the standard ordering we give an input with a running time of at least $(\log n)/2$.

First we consider arbitrary orderings. Let $A[0..n/2-1]$ and $B[0..n/2-1]$ be the two sequences to be merged, and let $C[0..n-1]$ be the output of the merging. At the beginning (the end, resp.) each of the n processors holds exactly one element of $A \cup B$ (of C, resp.). We consider the following $n/2$ inputs $I_0, ..., I_{n/2-1}$: In I_i the element $A[j]$ is moved to $C[j+i]$, $0 \leq j \leq n/2-1$. We will show that for every possible ordering of the processors one of these inputs needs at least $(\log n + \sqrt{\log n/2})/4$ time.

Let $d = \log n$. Consider a fixed input element $A[j]$ and let $A[j]$ reside at P^j at the beginning. In $I_0, ..., I_{n/2_1}$ there are $n/2$ different positions for $A[j]$ in C, i.e. $n/2$ different processors $A[j]$ has to be moved to. Assume that these processors are chosen such that the costs for $A[j]$ are minimized, i.e they are the processors with the $n/2$ smallest hamming distances to P^j. Thus the total costs for $A[j]$ in the $n/2$ inputs are

$$\geq \sum_{i=0}^{\lfloor d/2 \rfloor} i\binom{d}{i} \quad \text{if } d \text{ is odd, and}$$

$$\geq \sum_{i=0}^{d/2-1} i\binom{d}{i} + \frac{1}{4}d\binom{d}{d/2} \quad \text{if } d \text{ is even.}$$

Further, if d is odd,

$$\sum_{i=0}^{\lfloor d/2 \rfloor} i\binom{d}{i} = \sum_{i=1}^{(d-1)/2} d\binom{d-1}{i-1} = \frac{1}{4}d2^d + \frac{1}{2}d\binom{d-1}{(d-1)/2} \geq \frac{1}{4}(d+\sqrt{d/2})2^d,$$

and if d is even,

$$\sum_{i=0}^{d/2-1} i\binom{d}{i} + \frac{1}{4}d\binom{d}{d/2} = \sum_{i=1}^{\lfloor (d-1)/2 \rfloor} d\binom{d-1}{i-1} + \frac{1}{4}d\binom{d}{d/2} =$$

$$\frac{1}{4}d2^d + \frac{1}{4}d\binom{d}{d/2} \geq \frac{1}{4}(d+\sqrt{d/2})2^d.$$

(For the inequalities we used Sterling's approximation for $n!$.)
Thus, the total costs for all $n/2$ inputs, summed over all $A[j]$, $0 \leq j \leq n/2-1$, are at least $(n/2)(d+\sqrt{d/2})n/4$, and for at least one of the $n/2$ inputs the total costs (or work) are at least $(d+\sqrt{d/2})n/4$. Thus the running time for this input is at least $(d+\sqrt{d/2})/4 = (\log n + \sqrt{\log n/2})/4$. Note that this lower bound also holds if the orderings of the processors used at the beginning and at the end may differ.

If we use the standard ordering of the processors we can improve upon the constants in the above lower bound. Let A, B, and C be defined as above. At the beginning $A[i]$ resides at processor P_i and $B[i]$ at processor $P_{n/2+i}$, $0 \leq i \leq n/2-1$, and at the end $C[i]$ resides at processor P_i, $0 \leq i \leq n-1$. If the input elements move as follows, at least $(n\log n)/2$ edges have to be crossed altogether:

$$A[i] \rightarrow C[2i], 0 \leq i \leq n/2-1, \text{ and}$$
$$B[i] \rightarrow C[2i+1], 0 \leq i \leq n/2-1.$$

By induction on $\log n$ we can show that the performed work is at least $(n\log n)/2$. Applying this lower bound on each level of a two-way merge sort shows that such a sort algorithm performs a work of at least $n\log n(\log n+1)/4$.

4. Merging is expensive on the average

In this section we will show that merging two sorted sequences (of equal length), using an arbitrary ordering of the processors, is expensive on the average. (We average over all possible outcomes of the merging). This is the case even if the orderings of the processors used before and after the merging may differ.
First we consider the case where there is at most one element per processor; later we will extend this to the case where the number of processors is smaller than the number of elements. Thus, let $\mathcal{H}$ be a p-node hypercube, and let A and B be two sorted sequences of size $n/2$ each, $p = n^c$ for a constant $c \geq 1$, and n a power of 2. Assume that A and B are stored at the nodes of a subset S_A (S_B, resp.,) of the processors of $\mathcal{H}$, $|S_A| = |S_B| = n/2$ (one element per processor), and that we want to merge A and B such that the elements in $A \cup B$ are afterwards stored at a subset S of the processors, $|S| = n$, again one element per processor. The elements are stored in S_A, S_B, and S according to some fixed ordering of the processors in these sets. Note that S_A, S_B, and S need not be disjoint. (This corresponds to a merge step in a 2-way merge sort.)

We start off from the following idea. Let $M(n)$ be the number of possible outcomes of merging two lists of size $n/2$ each. For each input element each of these outcomes has some costs assigned to it, namely the hamming distance between the two processors storing the element before and after the merging. Consider a fixed input element X, stored at some processor P. We want to determine the set S of processors and the ordering of these processors such that the sum of the costs over all possible outcomes of the merging is minimized for X. Let S_X denote such a choice of S and let $R_X : S_X \to \{1, ..., n\}$, R_X bijective, be the ordering of the processors in S_i. We will show that, for a constant fraction of the input elements, the sum of the costs is $\Omega(M(n)\log n)$ when S_X and R_X are used. Thus for these input elements the sum of the costs is always $\Omega(M(n)\log n)$, regardless of the choice of S and the ordering of the processors in S. Since this is true for a constant fraction of the input elements, the sum of the overall (for all input elements) costs is $\Omega(M(n)n\log n)$ and thus the average costs of merging are $\Omega(n\log n)$.

The following bounds on binomial coefficients and sums of binomial coefficients will be needed later on.

Lemma 1

Let $n \in \mathbb{N}$ and let $\mu n \in \mathbb{N}$, $0 < \mu < 1$. Then

$$\frac{1}{\sqrt{8n\mu(1-\mu)}} 2^{nH_2(\mu)} \leq \binom{n}{\mu n} \leq \frac{1}{\sqrt{2\pi n\mu(1-\mu)}} 2^{nH_2(\mu)}, \text{ and}$$

$$\frac{1}{\sqrt{8n\mu(1-\mu)}} 2^{nH_2(\mu)} \leq \sum_{k=0}^{\mu n} \binom{n}{k} \leq 2^{nH_2(\mu)}, \text{ if } 0 < \mu < 1/2,$$

where $H_2(x) = -x\log x - (1-x)\log(1-x)$.

Proof: See, e.g. [WS78], pp. 308 ff. ∎

Let us next evaluate the number $M(n)$ of possible outcomes of merging two lists of size $n/2$ each.

Lemma 2

$$M(n) = \binom{n}{n/2}, \text{ and}$$

$$\sqrt{\frac{1}{2n}} 2^n \leq \binom{n}{n/2} \leq \sqrt{\frac{2}{\pi n}} 2^n$$

Proof: This follows from Lemma 1. ∎

Let $A = A[0..n/2-1]$ and $B = B[0..n/2-1]$ and let $C[0..n-1]$ be the result of the merging. We want to derive a lower bound on the average costs caused by the elements in A and B. To do this, we concentrate on the elements in A, since the situation for A and B is symmetrical. For each element $A[i]$, $0 \leq i \leq n/2-1$, there are $n/2$ possibilities for its location in C: it can move between 0 (when no element in B is smaller than $A[i]$) and $n/2$ (when all elements in B are smaller than $A[i]$) positions to the right. Each of these moves has

some costs associated with it: $costs(i,b)$, $0 \le b \le n/2$, is the hamming distance between the processor that stores $A[i]$ and the processor that stores $C[i+b]$. Each of these costs arises in several outcomes of the merging: $costs(i,b)$ arises in all outcomes where $A[i]$ moves b positions to the right. Let the number of these outcomes be $Z(i,b)$. Then the overall costs, in all possible outcomes of the merging, caused by $A[i]$ are

$$\sum_{0 \le b \le n/2} costs(i,b) Z(i,b).$$

($costs(i,b)$ of course depends on the chosen orderings on the processors, whereas $Z(i,b)$ is independent of them.)
We want to derive a lower bound on

$$\min_{S, \Pi_S} \left\{ \sum_{0 \le b \le n/2} costs(i,b) Z(i,b) \right\} =: Min(i),$$

where Π_S denotes the ordering of the processors in S. Note that this value is independent of the index of the processor that holds $A[i]$. To be able to derive a lower bound, we next examine $Z(i,b)$.

Lemma 3

(1) $$Z(i) = \binom{i+b}{i} \binom{n-(i+b)-1}{n/2-i-1}$$

(2) $$Z(i,b) = Z(n/2-i-1, n/2-b), \ 0 \le i \le n/2, \ 0 \le b \le n/2$$

(3) For a fixed i, $Z(i,b)$ first increases monotonically and then decreases monotonically, with a maximum at $Z(i,i)$ if $i \le n/4-1$ and at $Z(i,i+1)$ else.

Proof: We omit the proof. ∎

Because of Lemma 3.2 we can concentrate on the i where $0 \le i \le n/4-1$. Next we will estimate how large a fraction of all possible outcomes the largest "weight", $Z(i,i)$, comprises.

Lemma 4

$$\frac{\pi}{2}\sqrt{i} \le \frac{M(n)}{Z(i,i)} \le \frac{16}{\sqrt{\pi}}\sqrt{i}, \text{ if } n \ge 4.$$

Proof: This follows from Lemmas 1 through 3. ∎

That means that the largest existing weight is proportional to $M(n)/\sqrt{i}$. To achieve $Min(i)$ for a fixed i we should choose S and the ordering of the processors in S such that the largest weights are assigned to the smallest hamming distances, e.g. $A[i]$ and $C[2i]$ should be stored at the same processor. Assume that we assign the $L = \pi\sqrt{i}/(2c)$ largest weights ($c > 1$ a constant) to the L cheapest movements. The size of each of these weights is at most $2M(n)/(\pi\sqrt{i})$, and the sum of their sizes is at most $M(n)/c$. Thus the sum of the sizes of the remaining weights, i.e. the number of possible movements that have not yet been

assigned costs, is at least $(1-1/c)M(n)$. These remaining weights are all assigned to costs of at least $m(i,c)$, where $m(i,c)$ are the costs of the $L+1$ cheapest movement. Thus the sum of all costs is at least $m(i,c)(1-1/c)M(n)$, and the average costs are at least $m(i,c)(1-1/c)$. Thus, if $m(i,c) = \Omega(\log n)$, the average costs are $\Omega(\log n)$. The following lemma shows that this is indeed the case if $i = \Omega(n)$.

Lemma 5

Let $i = \Omega(n)$ and let $p = n^\alpha$ for a constant $\alpha > 1$. Then for every constant $c > 1$ there is a constant μ such that the number of movements with costs less than $\mu \log p$ is at most $\pi\sqrt{i}/(2c)$, if n is sufficiently large.

Proof: We have to show that

$$\sum_{j=0}^{\mu \log p} \binom{\log p}{j} \leq \frac{\pi\sqrt{i}}{2c} := x.$$

Since

$$\sum_{j=0}^{\mu \log p} \binom{\log p}{j} \leq 2^{\log p \, H_2(\mu)},$$

it is sufficient that

$$p^{H_2(\mu)} \leq x \text{ or}$$

$$H_2(\mu) \leq \log x / \log p.$$

However,

$$\log x / \log p = \log \frac{\pi}{2c} / \log p + (1/2) \log i / \log p \geq c'$$

for a constant $c' > 0$ if n is sufficiently large. Thus the constant μ exists. ∎

From Lemma 5 it follows that, for each constant c, $m(i,c) = \Omega(\log n)$, if $i = \Omega(n)$.

Thus we have proven the following theorem.

Theorem 1

Let $\mathcal{H}$ be a p-node hypercube, let A and B be two sorted sequences of size $n/2$ each, $p = n^c$ for a constant $c > 1$, and n a power of 2. Let A and B be stored at the nodes of a subset S_A (S_B, resp.,) of the processors of $\mathcal{H}$, $|S_A| = |S_B| = n/2$ (one element per processor), and let the result of the merging be stored at a subset S of the processors, $|S| = n$, again one element per processor. The elements are stored according to some fixed ordering of the used processors.

Then the average costs for merging A and B are $\Omega(n \log n)$.

We are also ready to prove a lower bound for sorting n elements by two-way merging on an n-processor hypercube.

Theorem 2

Sorting n elements on an n-node hypercube by repeated two-way merging needs time $\Omega((\log n)^2)$ on the average. This is true for all possible orderings of the processors in the levels of the recursion and even if the ordering may change from one level to the next as long as these orderings are independent of the input.

Proof: This follows directly from Theorem 1 by considering the upper, say, $1/2 \log n$ levels of the recursion. ∎

Next we want to examine the case where each processor may hold more than one element. First we consider the situation where we want to merge mn elements on a p-node hypercube, $p = n^\alpha$ for a constant $\alpha \geq 1$, and where, before and after the merging, each of n processors of the hypercube holds m elements. We will show that the average costs for doing this are $\Omega(mn \log p)$ if $m \leq n^\beta$ for a constant $\beta < 1$.
To do this, we reconsider the proof of Theorem 1. Lemmas 2, 3, and 4 hold if we replace n by $n' = mn$. Again we assign the cheapest movements to the largest weights. Instead of Lemma 5 we now have to show:

Lemma 6
Let $i = \Omega(mn)$, $m \leq n^\beta$ for a constant $\beta < 1$, and $p = n^\alpha$ for a constant $\alpha > 1$.
Then for every constant $c > 1$, there exists a constant μ such that the number of movements with costs less than $\mu \log p$ is smaller than $\pi\sqrt{i}/(2c)$, if n is sufficiently large.

Proof: Let $i \geq c'mn$. Since each movement is now assigned to m weights, we have to show that

$$\sum_{j=0}^{\mu \log p} m \binom{\log p}{j} \leq \frac{\pi\sqrt{i}}{2c} := x.$$

It is sufficient that

$$p^{H_2(\mu)} \leq x/m \text{ or}$$

$$H_2(\mu) \leq \log(x/m)/\log p.$$

However,

$$\log(x/m)/\log p \geq \log \frac{\pi}{2c}/\log p + 1/2 \log(c'n/m)/\log p \geq c''$$

for a constant $c'' > 0$ since $m \leq n^\beta$ and if n sufficiently large. Thus the constant μ exists. ∎

Again the lower bound for merging can be used to prove a corresponding lower bound for sorting.

Theorem 3
Sorting n elements on a p-node hypercube using repeated two-way merging needs work $\Omega(n(\log p)^2)$ if $p > n^{0.5+\epsilon}$ for any constant $\epsilon > 0$.

Proof: Consider once again the upper $(1/2)\log n$ levels of the recursion. To apply Lemma 6 on these levels the condition $n/p < p^\gamma$ for a constant $\gamma < 1$ must hold. This is equivalent to $p > n^{0.5+\epsilon}$ for a constant $\epsilon > 0$. ∎

5. Computing the rank

In Section 4 we showed that merging two sorted sequences A and B is, under certain conditions, expensive on the average. To prove these lower bounds we made use of the fact

that we wanted to rearrange the input elements according to their rank in A merged with B. Thus, if we only want to compute the rank of each input element in $A \cup B$, we cannot apply these lower bounds immediately. In this section we sketch how to change the claims and the proofs to obtain similar lower bounds if we only want to compute the ranks.

We make use of the following observation: Let $A = a[0..n/2-1]$ and $B = B[0..n/2-1]$ be the two input sequences and let $C = C[0..n-1]$ be the result of merging A with B. Assume that an $A[i]$ has rank j in C, i.e. $A[i]$ is equal to $C[j]$, and that $C[j-1] = B[j-i-1]$, $0 \le i \le n/2-1$, $0 \le j \le n-1$. Let $k = j-i-1$. Then $A[i]$ and $B[k]$ have to be compared to determine the rank of $A[i]$ in C.

We use the notation of Section 4. Let $A[i]$ be stored at processor P^i and $B[k]$ at processor P^k. To compare $A[i]$ and $B[k]$ these two input elements have to meet at some processor P^l. Further, $h(P^i, P^l) + h(P^k, p^l) \ge h(P^i, P^k)$, where the function h denotes the hamming distance between the indices of the two processors. Thus we can claim costs of $h(P^i, P^k)$ for each assignment of the input elements where $B[k]$ is directly before $A[i]$ in C. We will charge these costs on $A[i]$, i.e. we will consider only the elements in A.

Following this observation we define $\overline{Z}(i,b)$ as the number of all inputs where $A[i]$ has rank $i+b$ in C and $B[b-1]$ has rank $i+b-1$ in C, $0 \le i \le n/2-1$, $1 \le b \le n/2$. As in Section 4, we want to derive a lower bound on

$$\min_{S,\Pi_S} \left\{ \sum_{0 \le b \le n/2} costs(i,b)\overline{Z}(i,b) \right\} =: \overline{Min}(i),$$

where $costs(i,b)$ is the hamming distance between the indices of the processors that hold $A[i]$ ($B[b]$, resp.) in the beginning. Further,

$$\overline{Z}(i,b) = \binom{i+b-1}{i} \binom{n-(i+b)-1}{n/2-i-1} = \frac{b}{i+b} Z(i,b).$$

Thus,

$$Z(i,b)/3 \le \overline{Z}(i,b) \le Z(i) \text{ if } b \ge i/2, \text{ and}$$

$$M(n) \ge \sum_{b=0}^{n/2} \overline{Z}(i,b) \ge \sum_{b=i/2}^{n/2} Z(i,b)/3 \ge \frac{1}{6} \sum_{i=0}^{n/2} Z(i,b) = M(n)/6.$$

From this we can conclude that the average costs are only decreased by a constant factor if we merely want to compute the ranks of the input elements. We omit the details.

Summary

In this paper we have shown lower bounds for two-way merging and sorting by two-way merging on the hypercube. We have shown that such a sorting algorithm cannot be optimal unless the number of processors is significantly smaller than the number of elements to be sorted. This is true even if it is not the standard ordering of the processors that is used but any ordering that may even change from one merge to the next. The same ideas can be applied to the problem of merging two sorted lists of different lengths.

Acknowledgments

I would like to thank Torben Hagerup for helpful discussions.

References

[AH88] A. Aggarwal, M.-D. A. Huang, *Network complexity of sorting and graph problems and simulating CRCW PRAMs by interconnection networks*, AWOC, LNCS 319, pp. 339-350, 1988

[BN89] G. Bilardi, A. Nicolau, *Adaptive bitonic sorting: An optimal parallel algorithm for shared-memory machines*, SIAM J. Comput. 18, pp. 216–228, 1989.

[C88] R. Cole, *Parallel merge sort*, SIAM J. Comput. 17, 770–785, 1988.

[CyP90] R. Cypher, G. Plaxton, *Deterministic sorting in nearly logarithmic time on the hypercube and related computers*, 20th STOC, pp. 193–203, 1990

[CyS92] R. Cypher, J.L.C. Sanz, *Cubesort: A parallel algorithm for sorting N data items with S-sorters*, J. of Algorithms 13, pp. 211-234, 1992

[HR89] T. Hagerup, Ch. Rüb, *Optimal merging and sorting on the EREW PRAM*, Information Processing Letters 33 , pp. 181–185, 1989.

[K83] C.P. Kruskal, *Searching, merging and sorting in parallel computation*, IEEE Transactions on Computers, Vol. C-32, 942-946, 1983.

[L92] F.T. Leighton, *Introduction to parallel algorithms and architectures: Arrays, trees, hypercubes*, Morgan Kaufmann Publishers, San Mate, California, 1992

[LP90] T. Leighton, C.G. Plaxton, *A (fairly) simple circuit that (usually) sorts*, 31st FOCS, Vol. I, pp. 264-274, 1990

[WS78] F.J. MacWilliams, N.J. Sloane, *The Theory of Error Correcting Codes*, North-Holland Mathematical Library,Vol. 16, North-Holland, Amsterdam - New York - Oxford, 1978

[MPT92] P.B. Miltersen, M. Paterson, J. Tarui, *The asymptotic complexity of merging networks*, 33rd FOCS, pp. 236-246, 1992

[P89] C.G. Plaxton, *Load balancing, selection and sorting on the hypercube*, SPAA, pp. 64-73, 1989

[P92] C.G. Plaxton, *A hypercubic sorting network with nearly logarithmic depth*, 24th STOC, pp. 405-416, 1992