Parallel Merge Sort

Richard Cole
New York University

Abstract. We give a parallel implementation of merge sort on a CREW PRAM that uses n processors and $O(\log n)$ time; the constant in the running time is small. We also give a more complex version of the algorithm for the EREW PRAM; it also uses n processors and $O(\log n)$ time. The constant in the running time is still moderate, though not as small.

1. Introduction

There are a variety of models in which parallel algorithms can be designed. For sorting, two models are usually considered: circuits and the PRAM; the circuit model is the more restrictive. An early result in this area was the sorting circuit due to Batcher [B, 1968]; it uses time $1/2 \log^2 n$. More recently, [AKS, 1983] gave a sorting circuit that ran in $O(\log n)$ time; however, the constant in the running time was very large (we will refer to this as the AKS network). The huge size of the constant is due, in part, to the use of expander graphs in the circuit. The recent result of Lubotsky et al [LPS, 86] concerning expander graphs may well reduce this constant considerably; however, it appears that the constant is still large [CO, 1986].

The PRAM provides an alternative, and less restrictive, computation model. There are three variants of this model that are frequently used: the CRCW PRAM, the CREW PRAM, and the EREW PRAM; the first model allows concurrent access to a memory location both for reading and writing, the second model allows concurrent access only for reading, while the third model does not allow concurrent access to a memory location. A sorting circuit can be implemented in any of these models (without loss of efficiency).

Preparata [P, 1978] gave a sorting algorithm for the CREW PRAM that ran in $O(\log n)$ time on $(n \log n)$ processors; the constant in the running time was small. (In fact, there were some implementation details left incomplete in this algorithm; this was rectified by Borodin and Hopcroft in [BH, 1982].) Preparata's algorithm was based on a merging procedure given by Valiant [V, 1975]; this procedure merges two sorted arrays, each of length at most n, in time $O(\log\log n)$ using a linear number of processors. When used in the obvious way, Valiant's procedure leads to an implementation of merge sort on n processors using $O(\log n \log\log n)$ time. More recently, Kruskal [K, 1983] improved this sorting algorithm to obtain a sorting algorithm that ran in time $O(\log n \log\log n / \log\log\log n)$ on n processors. (The basic algorithm was Preparata's; however, a different choice of parameters was made.) In part, Kruskal's algorithm depended on using the most efficient versions of Valiant's merging algorithm; these are also described in Kruskal's paper.

More recently, Bilardi and Nicolau [BN, 1986] gave an implementation of bitonic sort on the EREW PRAM that used $n/\log n$ processors and $O(\log^2 n)$ time. The constant in the running time was small.

In the next Section, we describe a simple CREW PRAM sorting algorithm that uses n processors and runs in time $O(\log n)$. In Section 3 we modify the algorithm to run on the EREW PRAM. The algorithm still runs in time $O(\log n)$ on n processors; however, the constant in the running time is somewhat less small than for the CREW algorithm. We note that apart from the AKS sorting network, the known deterministic EREW sorting algorithms that use about n processors all run in time $O(\log^2 n)$ (these algorithms are implementations of the various sorting networks such as Batcher's sort). Our algorithms will not make use of expander graphs or any related constructs; this avoids the huge constants in the running time associated with the AKS construction.

The contribution of this work is twofold: first, it provides a second $O(\log n)$ time, n processor parallel sorting algorithm (the first such algorithm is implied by the AKS sorting circuit); second, it considerably reduces the constant in the running time (by comparison with the AKS result). Of course, AKS is a sorting circuit; this work does not provide a sorting circuit.

This work was supported in part by an IBM Faculty Development Award and by NSF grant DCR-84-01633.

2. The CREW algorithm

We give an algorithm for sorting n numbers. For simplicity, suppose that n is a power of 2, and all the items are distinct. Our algorithm will be described in terms of an n-leaf complete binary tree. The inputs are placed at the leaves of the tree. Let v be an internal node of the tree and let T_v be the subtree rooted at v. The task, at node v, is to compute the list, L_v, comprising the items at the leaves of T_v in sorted order. At intermediate steps in the computation, at node v, we will have computed UP_v, a sorted subset of the items in L_v. The items in UP_v will be roughly evenly distributed over L_v.

A few definitions will be helpful. Let e, f, g be three items, with $e \le f$. g is *between* e and f if $e < g$ and $g \le f$. Sorted list L is a c-covering sampler for sorted list J if between each two adjacent items in $\{-\infty, L, \infty\}$ there are at most c items from J ($\{-\infty, L, \infty\}$ denotes the list comprising the item $-\infty$, followed by the items in L, followed by the item ∞).

At each node v we keep one list: UP_v; it is stored in an array. It comprises a sorted subset of the items at the leaves of T_v. Let x, y, w, and u denote, respectively, v's children, v's sibling, and v's parent.

We distinguish *external* and *inside* nodes. Node v is external if UP_v contains all the items originally at the leaves of T_v, and if u, v's parent, is not external. Let $UP_v(s)$ denote the list UP_v at the start of stage s. Initially, UP_v is empty at every node except the leaves. Let $UP'_v(s+1)$ be the following list: if v is an inside node it comprises every fourth item in $UP_v(s)$; on the first stage at which v is an external node, it comprises every fourth item in $UP_v(s)$, on the second such stage it comprises every second item in $UP_v(s)$, and on the third such stage it comprises every item in $UP_v(s)$. The algorithm has $3\log n$ stages; stage s consists of the following step, performed at every inside node v.

> Form the lists $UP'_x(s+1)$ and $UP'_y(s+1)$. Compute the new list $UP_v(s+1) := UP'_x(s+1) \cup UP'_y(s+1)$, where $\cup$ denotes a merge.

We have to show that the merge can be performed in $O(1)$ time. Among other things, this requires showing that $UP_u(s)$ is a covering sampler for $UP_v(s)$ and $UP_w(s)$. This follows from Lemma 2, below.

It is easy to see that 3 stages after v becomes an external node, its parent, u, becomes external. Thus,

Lemma 1: The algorithm has $3\log n$ stages.

Lemma 2: Between $k+1$ adjacent items in the list $\{-\infty, UP'_v(s), \infty\}$, there are at most $ka + a'$ items from $UP_v(s)$, for all $k \ge 1$ and $s \ge 1$, with $a = 8$ and $a' = 4$.

Proof: We prove the result by induction on s. The claim is true initially, for when UP'_v first becomes non-empty, at stage s, it contains one item and $UP_v(s)$ contains 8 items, and if $UP'_v(s)$ is empty then $UP_v(s)$ contains at most 4 items.

Inductive step. We first suppose that v is not external at stage s. Then, between $h+1$ items, adjacent in $\{-\infty, UP'_x(s), \infty\}$, there are at most $ha + a'$ items in $UP_x(s)$, and hence at most $\lceil (ha+a')/4 \rceil$ items in $UP'_x(s+1)$. Likewise, between $j+1$ items, adjacent in $\{-\infty, UP'_y(s), \infty\}$, there are at most $\lceil (ja+a')/4 \rceil$ items in $UP'_y(s+1)$. We call the range between two adjacent items from $\{-\infty, UP'_x(s), \infty\}$ (resp. $\{-\infty, UP'_y(s), \infty\}$) an *interval* with respect to x (resp. y). Recall $UP_v(s) = UP'_x(s) \cup UP'_y(s)$. The range between $4k+1$ items in $\{-\infty, UP_v(s), \infty\}$ intersects some h intervals with respect to x (between $h+1$ items from $\{-\infty, UP'_x(s), \infty\}$) and some j intervals with respect to y (between $j+1$ items from $\{-\infty, UP'_y(s), \infty\}$) with $h + j = 4k+1$. Thus between $4k+1$ items in $\{-\infty, UP_v(s), \infty\}$ (and hence between $k+1$ items in $\{-\infty, UP'_v(s+1), \infty\}$) there are at most $\lceil (ah+a')/4 \rceil + \lceil (a(4k-h+1)+a')/4 \rceil$ items in $UP'_x(s+1) \cup UP'_y(s+1)$ $(= UP_v(s+1))$. Thus we require:

$$ka + a' \ge \lceil (ah+a')/4 \rceil + \lceil (a(4k-h+1)+a')/4 \rceil$$

which is easily verified. The case with v an external node at stage s is simpler and is left to the reader (in fact, if v is an external node at the start of stage $s-1$ (resp. $s-2$), then rather than the bound of $8k+4$ we have a bound of $4k$ (resp. $2k$)). □

Corollary 1: Between $k+1$ adjacent items in $\{-\infty, UP'_v(s), \infty\}$, there are at most $2k+1$ items from $UP'_v(s+1)$, for all $k \ge 1$, if $UP'_v(s)$ is non-empty.

Definition. Let L be a sorted set. The *rank* of item e in L is r if there are e items in L preceding e (including e itself if e is in L). This definition does not require e to be in L.

We need the following merging procedure. Let L, J, K be three sorted lists, stored in arrays, with L a c_1-covering sampler for J and a c_2-covering sampler for K. Suppose that for each item in L we have its rank in each of J and K, while for each item in J and K we have its rank in L (the *cross-ranks*). We show how to merge the items in J and K. We use two auxiliary arrays A and A_K, of lengths $c_1|L|$ and $c_2|L|$, respectively. To the positions $c_1 r$, ..., $c_1 r + c_1 - 1$ of A_J (resp. positions $c_2 r$, ..., $c_2 r + c_2 - 1$ of A_K) we write the (at most) c_1 items in J (resp. c_2 items in K) that are between the rth and $r+1$th items in L, for $r \ge 0$ (we assume that items $\pm\infty$ have been added to L in the $(|L|+1)$th and 0th positions, respectively). Consider item e in J. Suppose that e has rank r_1 in J and r_2 in L. Let e' be the item in L of rank r_2; suppose that e' has rank r_3 in K. Then the rank of e in the merge of J and K is at least r_1+r_3 and at most $r_1+r_3+c_2$. To determine the exact rank we merely have to merge corresponding portions of the arrays A_J and A_K. But this can be done in $O(1)$ time on an EREW PRAM, for c_1 and c_2 are constants. We call this procedure ***merging J*** and ***K with the help of L***. (Actually, there is no need for arrays A_J and A_K; they are here simply to clarify the exposition.)

We also need a second merging procedure. Let J and K be sorted lists. Suppose that K is the merge of sorted lists L and M, with L a c_3-covering sampler for J. Further suppose that for each item in M we have its rank in L and for each item in L we have its rank in J (the *cross-ranks*). We show how to merge J and K. Consider a pair, e and f, of adjacent items from L. There are at most c_3 items in J between e and f. Thus our task reduces to merging these at most c_3 items from J with those items from M that are between e and f. Because of the presence of the cross-ranks this can be done in $O(1)$ time on a CREW PRAM if we associate one processor with each item in K (see [V, 1975] and [BH, 1982]). We call this procedure ***merging J*** with ***covering K***.

For each item in the UP and UP' lists, at the start of stage s, we maintain the following ***cross-ranks***.

(i) For each item in $UP'_v(s)$: its rank in $UP'_w(s)$.

(ii) For each item in $UP'_v(s)$: its rank in $UP_v(s)$ (and, implicitly, its rank in $UP'_v(s+1)$).

We merge $UP'_v(s+1)$ and $UP'_w(s+1)$ as follows. We start by merging each of $UP'_v(s+1)$ and $UP'_w(s+1)$ with covering $UP_u(s) = UP'_v(s) \cup UP'_w(s)$ in $O(1)$ time. (For the first merge, set $J = UP'_v(s+1)$, $L = UP'_v(s)$, $M = UP'_w(s)$; by (i) and (ii) we have the necessary cross-ranks, and by Corollary 1, with $k=1$, L is a 3-covering sampler for J. The second merge is performed similarly.) Then we merge $UP'_v(s+1)$ and $UP'_w(s+1)$ with the help of $UP_u(s)$. (The merges of the previous sentence provide the cross-ranks; Corollary 1 shows that the constants c_1 and c_2 are both equal to 3.)

The new cross-ranks are all yielded by these merges. (i) is clear; for (ii), we note that $UP'_v(s+1)$ is a subset of $UP_v(s)$, and the rank in $UP_v(s+1) = UP'_x(s+1) \cup UP'_y(s+1)$ of each item from $UP_v(s)$ (and hence of each item from $UP'_v(s+1)$) was implicitly computed in merging $UP_v(s)$ with each of $UP'_x(s+1)$ and $UP'_y(s+1)$.

We have shown

Theorem 1: There is an CREW PRAM sorting algorithm that that runs in $O(\log n)$ time on n processors.

In fact, if we count the number of comparisons performed by this algorithm we discover that it is just $4n\log n$ comparisons. However, to achieve this few comparisons requires using time $28\log n$ for comparisons (and thus $n/7$ processors), and a similar time for indexing and other computations. If we use $8/7n$ processors, a time of $12\log n$ for comparisons can be achieved (by performing 4 comparison steps per phase). By noting that c_i is 1, for $i=1,2,3$, for the merges in the last two phases for each external node, we deduce that the total number of comparisons is actually $5/2n\log n$.

Remark. The 'obvious' algorithm that follows the pattern described above would have $UP'_v(s+1)$ comprising every second item in $UP_v(s)$. But then it is impossible to obtain a result of the form of Lemma 2, and the algorithm would be unable to achieve $O(\log n)$ running time. Even so, our definition of UP'_v is not the only possible definition, although it does seem to be the simplest. In fact, a uniform definition is not necessary (for example, on alternate stages $UP'_v(s+1)$ could comprise every second and every fourth item in $UP_v(s)$, respectively).

3. The EREW algorithm

The algorithm here has the same structure as the algorithm in Section 2. The UP and UP' lists are as before. We aim to merge the UP' lists, as in Section 2. However, we observe that we cannot use the same algorithm as above, for it used a CREW merging procedure (the second merging procedure we gave) to merge lists $UP_u(s)$ and $UP'_v(s+1)$ in $O(1)$ time. Instead, we need to ensure that all the merges are done using the first merging procedure from Section 2. In order to achieve this we keep a second list, $DOWN_v$, at each node v. It has the following properties.

(i) Let w be the sibling of v; then $DOWN_v = DOWN_w$.

(ii) $DOWN_v$ is a covering sampler for the following lists: $DOWN_x$, $DOWN_y$, $DOWN_u$, and UP_v, where x and y are the children of v, and u is the parent of v.

Initially, $DOWN_v$ is empty for eveiy node. $DOWN'_v(s+1)$ denotes the list that comprises every fourth item in $DOWN_v(s)$. The algorithm has $3\log n$ stages; stage s comprises the following two steps, performed at every inside node v; the second step is also performed at external nodes.

1) $UP_v(s+1) := UP'_x(s+1) \cup UP'_y(s+1)$.

2) $DOWN_v(s+1) := DOWN'_u(s+1) \cup UP'_u(s+1)$.

We have to show that each of the merges can be performed in $O(1)$ time. This requires showing that the DOWN lists are covering samplers, as claimed above. More precisely, we will show that the following invariants are maintained.

(A) Between $k+1$ adjacent items in the list $\{-\infty, UP'_v(s), \infty\}$, there are at most $ka+a'$ items from $UP_v(s)$, for all $k \geq 1$ and $s \geq 1$, with $a=8$, $a'=4$.

(B) Between $k+1$ adjacent items in the list $\{-\infty, DOWN_v(s), \infty\}$ there are at most $kb+b'$ items from $UP_v(s) \cup UP_w(s)$, for all $k \geq 1$ and $s \geq 1$, with $b=64$ and $b'=40$.

(C) Between $k+1$ adjacent items in the list $\{-\infty, DOWN_x(s), \infty\}$ there are at most $kc+c'$ items from $DOWN_v(s)$, for all $k \geq 1$ and $s \geq 1$, with $c=8$ and $c'=4$.

(D) Between $k+1$ adjacent items in the list $\{-\infty, DOWN_v(s), \infty\}$ there are at most $kd+d'$ items from $DOWN_x(s)$, for all $k \geq 1$ and $s \geq 1$, with $d=8$ and $d'=12$.

(A) has already been shown in Lemma 2; the remaining invariants are shown in Lemmas 3-5, below.

As before, the algorithm has $3\log n$ stages.

For each item in the UP and $DOWN$ lists we maintain the following *cross-ranks*.

(i) For each item in $UP_v(s)$: its rank in $DOWN_v(s)$.

(ii) For each item in $DOWN_v(s)$: its rank in $UP_v(s)$, and its rank in $DOWN_u(s)$, $DOWN_x(s)$, $DOWN_y(s)$ $(= DOWN_x(s))$.

To compute $UP_v(s+1) := UP'_x(s+1) \cup UP'_y(s+1)$, we merge $UP'_x(s+1)$ and $UP'_y(s+1)$ with the help of $DOWN_x(s)$ $(= DOWN_y(s))$. The constants c_1 and c_2 are both equal to $\lceil (b+b')/4 \rceil$ $(=26)$ by invariant (B), above. To compute the new $DOWN_v(s+1) = DOWN'_u(s+1) \cup UP'_u(s+1)$ is immediate, given the cross-ranks.

We also need to explain how to maintain the cross-ranks. We first explain how to cross-rank $UP_v(s+1) = UP'_x(s+1) \cup UP'_y(s+1)$ with $DOWN_v(s+1) = DOWN'_u(s+1) \cup UP'_u(s+1)$. The procedure follows.

(1) Compute $DOWN_v(s) \cup DOWN_x(s)$ with the help of $DOWN_v(s)$; by invariant (D) the constants c_1 and c_2 are 1 and $d+d'$ $(=20)$, respectively.

(2) Compute $DOWN_u(s) \cup (DOWN_v(s) \cup DOWN_x(s))$ with the help of $DOWN_v(s)$; by invariants (C) and (D) the constants c_1 and c_2 are $c+c'$ $(=12)$ and $d+d'+1$ $(=21)$, respectively.

(3) Merge (or rather cross-rank) $DOWN'_u(s+1) \cup UP'_u(s+1)$ with $DOWN_u(s) \cup DOWN_v(s)$ (computed in Step 1 for node u) with the help of $DOWN_u(s)$; By invariants (B) and (D), the constants c_1 and c_2 are $\lceil (b+b')/4 \rceil + 1$ $(=27)$ and $d+d'+1$ $(=21)$, respectively.

(4) Cross-rank $DOWN'_u(s+1) \cup UP'_u(s+1)$ with $DOWN_u(s) \cup DOWN_v(s) \cup DOWN_x(s)$ with the help of $DOWN_u(s) \cup DOWN_v(s)$; again c_1 and c_2 are $\lceil (b+b')/4 \rceil + 1$ $(=27)$ and $d+d'+1$ $(=21)$, respectively.

(5) Cross-rank $UP'_x(s+1) \cup UP'_y(s+1)$ with $DOWN_v(s) \cup DOWN_x(s)$ with the help of $DOWN_x(s)$; by invariants (B) and (C) the constants c_1 and c_2 are $\lceil (b+b')/4 \rceil$ $(=26)$ and $c+c'+1$ $(=13)$, respectively.

(6) Cross-rank $UP'_x(s+1) \cup UP'_y(s+1)$ with $DOWN_u(s) \cup DOWN_v(s) \cup DOWN_x(s)$ with the help of $DOWN_v(s) \cup DOWN_x(s)$; again c_1 and c_2 are $\lceil (b+b')/4 \rceil$ $(=26)$ and $c+c'+1$ $(=13)$, respectively.

(7) Cross-rank $UP'_x(s+1) \cup UP'_y(s+1)$ with $DOWN'_u(s+1) \cup UP'_u(s+1)$ with the help of $DOWN_u(s) \cup DOWN_v(s) \cup DOWN_x(s)$; by invariant (B) the constants c_1 and c_2 are $\lceil (b+b')/4 \rceil$ $(=26)$ and $\lceil (b+b')/4 \rceil + 1$ $(=27)$, respectively.

It remains to explain how to compute the cross-ranks for $DOWN_v(s+1) = DOWN'_u(s+1) \cup UP'_u(s+1)$ and $DOWN_x(s+1) = DOWN'_v(s+1) \cup UP'_v(s+1)$. We already have the cross-ranks of $DOWN'_u(s+1) \cup UP'_u(s+1)$ with $DOWN_u(s) \cup DOWN_v(s)$ (Step 3, above) and (implicitly) of $DOWN'_v(s+1) \cup UP'_v(s+1)$ with $DOWN_v(s)$, so the following two steps suffice:

(1) Cross-rank $DOWN'_v(s+1) \cup UP'_v(s+1)$ with $DOWN_u(s) \cup DOWN_v(s)$ with the help of $DOWN_v(s)$; by invariant (B) and (C) the constants c_1 and c_2 are $\lceil (b+b')/4 \rceil + 1$ $(=27)$ and $c+c'+1$ $(=13)$, respectively.

(2) Cross-rank $DOWN'_u(s+1) \cup UP'_u(s+1)$ with $DOWN'_v(s+1) \cup UP'_v(s+1)$ with the help of $DOWN_u(s) \cup DOWN_v(s)$; by invariant (B) the constants c_1 and c_2 are both $\lceil (b+b')/4 \rceil + 1$ $(=27)$.

The merging steps are simplified as appropriate, when they involve external nodes.

We now prove that the invariants are maintained. All 3 lemmas that follow are proved by induction on the stage number.

Lemma 3: Between $k+1$ adjacent items in the list $\{-\infty, DOWN_v(s), \infty\}$ there are at most $kb+b'$ items from $UP_v(s) \cup UP_w(s)$, for all $k \geq 1$ and $s \geq 1$, with $b=64$ and $b'=40$.

Proof: The claim is true initially, for when $DOWN_v$ becomes non-empty, at stage s, it contains one item, and $UP_v(s) \cup UP_w(s)$ contains 64 items, and if $DOWN_v(s)$ is empty then $UP_v(s) \cup UP_w(s)$ contains at most 32 items.

Inductive step. $UP_u(s)$ contains at least 8 items. By Lemma 2, between $4k+1$ items in $\{-\infty, UP_u(s), \infty\}$, there are at most $(4k+1)a+2a'$ items in $UP_v(s) \cup UP_w(s)$, and thus at most $(((4k+1)a+2a'+2)a+4a')$ items in the $UP(s)$ lists at the grandchildren of u. Hence, between $k+1$ items in $\{-\infty, UP'_u(s+1), \infty\}$ (i.e. between at least $k+1$ items in $\{-\infty, DOWN_v(s+1), \infty\}$) there are at most $ka^2 - a(a+2a'+2)/4 + 4\lceil a'/4 \rceil$ items in $UP_v(s+1) \cup UP_w(s+1)$. Thus we require

$$kb+b' \geq ka^2 + a(a+2a'+2)/4 + 4\lceil a'/4 \rceil$$

This is easily verified. □

Lemma 4: Between $k+1$ adjacent items in $\{-\infty, DOWN_x(s), \infty\}$ there are at most $kc+c'$ items from $DOWN_v(s)$, for all $k \geq 1$ and $s \geq 1$, with $c=8$ and $c'=4$.

Proof: We actually prove the following result. Between $k+1$ adjacent items in $\{-\infty, DOWN_x(s), \infty\}$ there are at most $hc+c'$ items in $DOWN_v(s)$ and $ja+a'$ items in $UP_v(s)$, for some h,j, with $h+j=k+1$ and $h,j \geq 1$.

The claim is initially true, for when $DOWN_x$ becomes non-empty, at stage s, there are 0 items in $DOWN_v(s)$ and 8 items in $UP_v(s)$, and if $DOWN_x(s)$ is empty then there are 0 items in $DOWN_v(s)$ and at most 4 items in $UP_v(s)$. Further, while $|DOWN_v(s)| \leq 4$, the relationship between $DOWN_x(s)$ and $UP_v(s)$ is the same as that between $UP'_v(s)$ and $UP_v(s)$, as given by Lemma 2. Thus the claim remains true while $|DOWN_v| \leq 4$. For later stages, the result will follow by induction.

Inductive step. $DOWN_v(s)$ contains at least 4 items. Between $4h+1$ items in $\{-\infty, DOWN_v(s), \infty\}$ (and hence between $h+1$ items in $\{-\infty, DOWN'_v(s+1), \infty\}$) there are at most $h'c+c'$ items in $DOWN_u(s)$ (and hence at most $\lceil (h'c+c')/4 \rceil$ items in $DOWN'_u(s+1)$) and at most $(4h+1-h')a+a'$ items in $UP_u(s)$ (and hence at most $\lceil ((4h+1-h')a+a')/4 \rceil$ items in $UP'_u(s+1)$), with $1 \leq h' \leq 4h$. Between $4j+1$ items in $\{-\infty, UP_v(s), \infty\}$ (and hence between $j+1$ items in $\{-\infty, UP'_v(s+1), \infty\}$) there are at most $4ja+a+2a'$ items in $UP_x(s) \cup UP_y(s)$ (and hence at most $ja+a/4+2\lceil a'/4 \rceil$ items in $UP'_x(s+1) \cup UP'_y(s+1) = UP_v(s+1)$). Let the range between each two adjacent items in $\{-\infty, DOWN'_v(s+1), \infty\}$ define an interval (*with respect to* v_D) and the range between each two adjacent items in $\{-\infty, UP'_v(s+1), \infty\}$ define an interval (*with respect to* v_U). Then the range between $k+1$ items in $\{-\infty, DOWN_x(s+1), \infty\}$ $(=\{-\infty, DOWN'_v(s+1) \cup UP'_v(s+1), \infty\})$ overlaps some h intervals with respect to v_D (between $h+1$ items in $\{-\infty, DOWN'_v(s+1), \infty\}$) and j intervals with respect to v_U (between $j+1$ items in $\{-\infty, UP'_v(s+1), \infty\}$), with $h+j=k+1$ and $h,j \geq 1$. So between $k+1$ items in $\{-\infty, DOWN_x(s+1), \infty\}$ there are at most $\lceil (h'c+c')/4 \rceil + \lceil ((4h+1-h')a+a')/4 \rceil$ items in $DOWN_v(s+1)$ $(=DOWN'_u(s+1) \cup UP'_u(s+1))$ and $ja+a/4+2\lceil a'/4 \rceil = ja+a'$ items in $UP_v(s+1)$, with $h+j=k+1$ and $h,j \geq 1$. Thus, we require

$$hc+c' \geq \lceil (h'c+c')/4 \rceil + \lceil ((4h+1-h')a+a')/4 \rceil,$$

where $1 \leq h' \leq 4h$. This is easily verified. □

Lemma 5: Between $k+1$ adjacent items in $\{-\infty, DOWN_v(s), \infty\}$ there are at most $kd'+d$ items from $DOWN_x(s)$, for all $k \geq 1$ and $s \geq 1$, with $d=8$ and $d'=12$.

Proof: The claim is initially true, for when $DOWN_v$ becomes non-empty, at stage s, there are 4 items in $DOWN_x(s)$, and if $DOWN_v(s)$ is empty there are at most 2 items in $DOWN_x(s)$. Further, while $|DOWN_v| \leq 4$, the relationship between $DOWN_v(s)$ and $DOWN_x(s)$ is the same as that between $UP'_u(s)$ and $UP'_v(s)$ (a subset of $UP_u(s)$); it is (implicitly) given by Lemma 2; thus the claim holds in this case. Also, by inspection, the claim will hold when $|DOWN_v| \leq 16$ (for this leads to at most 2 more items in $DOWN_x(s)$ in addition to those from $UP'_v(s)$). For later stages, the result follows by induction.

Inductive step. $DOWN_v(s)$ contains at least 16 items and so $DOWN_u(s)$ contains at least 4 items. Between $4h+1$ items in $\{-\infty, DOWN_u(s), \infty\}$ (and hence between $h+1$ items in $\{-\infty, DOWN'_u(s+1), \infty\}$) there are at most $4hd+d'$ items in $DOWN_v(s)$ (and hence at most $hd+\lceil d'/4 \rceil$ items in $DOWN'_v(s+1)$). Between $4j+1$ items in $\{-\infty, UP_u(s), \infty\}$ (and hence between $j+1$ items in $\{-\infty, UP'_u(s+1), \infty\}$) there are at most $4ja+a'$ items in $UP_v(s)$ (and hence at most $ja+\lceil a'/4 \rceil$ items in $UP'_v(s+1)$). Let the range between each two adjacent items in $\{-\infty, UP'_u(s+1), \infty\}$ define an interval (*with respect to* u_U) and the range between each two adjacent items in $\{-\infty, DOWN'_u(s+1), \infty\}$ an interval (*with respect to* u_D). Then $k+1$ adjacent items in

$\{-\infty, DOWN_v(s+1), \infty\}$ $(=\{-\infty, DOWN'_u(s+1) \cup UP'_u(s+1), \infty\})$ overlap some h intervals with respect to u_D (enclosed by $h+1$ adjacent items in $\{-\infty, DOWN'_u(s+1), \infty\}$) and j intervals with respect to u_U (enclosed by $j+1$ adjacent items in $\{-\infty, UP'_u(s+1), \infty\}$), with $h+j=k+1$. Thus between $k+1$ items in $\{-\infty, DOWN_v(s+1), \infty\}$ there are at most $hd + \lceil d'/4 \rceil + ja + \lceil a'/4 \rceil$ items in $DOWN_x(s+1)$ $(=DOWN'_v(s+1) \cup UP'_v(s+1))$. So we require

$$kd + d' \geq hd + (k-h+1)a + \lceil d'/4 \rceil + \lceil a'/4 \rceil .$$

This is easily verified. □

We have shown

Theorem 2: There is an EREW PRAM sorting algorithm that takes time $O(\log n)$ to sort n numbers using n processors.

Remark. The work in this section is intended to demonstrate the existence of an EREW algorithm of the type described. The author suspects that it should be possible to improve the constants by changing the definition of the *DOWN* lists.

Acknowledgements. Thanks to Richard Anderson and Allen Van Gelder for commenting on an earlier version of this result. In particular, Allen Van Gelder suggested the term *cross-rank*. Many thanks to Jeanette Schmidt for questions and comments on various versions of the result.

References

[AKS, 1983] M. Ajtai, J. Komlos, E. Szemeredi, "An $O(n\log n)$ sorting network", *Combinatorica*, 3(1983), 1-19.

[B, 1968] K. E. Batcher, "Sorting Networks and their applications", *Proc. AFIPS Spring Joint Summer Computer Conf.*, Vol.32, 307-314.

[BH, 1982] A. Borodin and J. Hopcroft, "Routing, merging and sorting on parallel models of computation", *Proc. Fourteenth Annual ACM Symp. on Theory of Computing, 338-344.*

[BN,1986] G. Bilardi and A. Nicolau, "Bitonic sorting with $O(n\log n)$ comparisons", *Proc. Twentieth Annual Conf. on Information Sciences and Systems.*

[CO, 1986] R. Cole and C. O'Dunlaing, "Notes on the AKS sorting network, as improved by Paterson", in preparation.

[IS, 1983] J. Incerpi and R. Sedgewick, "Improved upper bounds on shellsort", *Twenty Fourth Annual Symposium on Foundations of Computer Science*, 48-55.

[L, 1984] T. Leighton, "Tight bounds on the complexity of parallel sorting",*Proc. Sixteenth Annual ACM Symp. on Theory of Computing*, 71-80.

[LPS, 1986] A. Lubotzky, R. Phillips, and P. Sarnak, "Ramanujan conjecture and explicit constructions of expanders and superconcentrators", *Eighteenth Annual Symposium on Theory of Computing.*

[K, 1983] C. Kruskal, "Searching, merging, and sorting in parallel computation", *IEEE Trans. Comp.*, Vol.C-32, No.10, 942-946.

[M, 1983] N. Megiddo, "Applying parallel computation algorithms in the design of serial algorithms", *JACM, 30, 4(1983), 852-865.*

[P, 1978] F. P. Preparata, "New Parallel-Sorting Schemes", *IEEE Transactions on Computers*, Vol. c-27, No. 7, 669-673.

[V, 1975] L. Valiant, "Parallelism in comparison problems", *SIAM J. Comput*, Vol. 4, 348-355.