Optimal Parallel Selection[1]

Yijie Han

School of Computing and Engineering
University of Missouri at Kansas City
5100 Rockhill Road
Kansas City, MO 64110
hanyij@umkc.edu

Abstract

We present an optimal parallel selection algorithm on the EREW PRAM. This algorithm runs in $O(\log n)$ time with $n/\log n$ processors. This complexity matches the known lower bound for parallel selection on the EREW PRAM model. We therefore close this problem which has been open for more than a decade.

keywords: parallel algorithms, selection, EREW PRAM.

1. Introduction

We consider the parallel computation of selection on the PRAM (Parallel Random Access Machine) model. A PRAM is a parallel machine with p processors and a global shared memory. Each processor can perform the usual computation of a sequential machine. Depending on the way the processors access the global memory, PRAM can be classified as EREW (Exclusive Read Exclusive Write) PRAM, CREW (Concurrent Read Exclusive Write) PRAM and CRCW (Concurrent Read Concurrent Write) PRAM. On the EREW PRAM each processor can access any global memory cell. But simultaneous access to the same memory cell by several processors is prohibited. On the CREW PRAM simultaneous read to the same memory cell by several processors are allowed, but simultaneous write is prohibited. On the CRCW PRAM simultaneous read and simultaneous write to the same memory cell are allowed. In the case on simultaneous write an arbitration mechanism is needed to decide which processor actually writes into the shared cell. It is known that CREW PRAM is more powerful than EREW PRAM[18] and CRCW PRAM is more powerful than CREW PRAM[8].

Selection refers to the finding of an element of a given rank from a set of ordered (but not sorted) elements. Selection is a fundamental computation problem and has been studied by many researchers. Blum *et al.* [4] showed that selection from a set of n elements

[1]This research is supported in part by National Science Foundation Grant 0310245. A preliminary version of this paper appeared in [12].

can be done in linear time ($O(n)$ time). In-depth research has been done on parallel selection. Valiant introduced the parallel comparison tree model and gave the lower bound of $\Omega(n/p+\log(\log n/\log(1+p/n)))$ time for computing the maximum among n elements using p processors [19]. Since computing the maximum is a special case of selection, this lower bound applies to parallel selection. Based on the work of Ajtai *et al.* [2] Azar and Pippenger gave a matching upper bound on the parallel comparison tree model [3]. Reischuk [16] showed that on the randomized parallel comparison tree model selection can be done in constant expected time with a linear number of processors. On the PRAM model Valiant's algorithm [19] can essentially be realized on the CRCW PRAM. Valiant raised the quesiton concerning the parallel complexity of selection especially that of finding the median. Significant progress has been made by Richard Cole[7] who showed a CRCW PRAM selection algorithm running in $O(\log n \log^* n/\log\log n)$ time with linear ($O(n)$, therefore optimal) operations (time processor product) and an EREW PRAM algorithm running in $O(\log n \log^* n)$ time with linear ($O(n)$, therefore optimal) operations. Cole's algorithms, when runs with optimal operations, are not time optimal. His algorithms can run in optimal time ($O(\log n/\log\log n)$ on the CRCW PRAM and $O(\log n)$ on the EREW PRAM) but then they require nonoptimal operations. Chaudhuri *et al.* [5] came up with the first simultaneous time and operation optimal selection algorithm on the CRCW PRAM. Their algorithm runs in $O(\log n/\log\log n)$ time with $O(n)$ operations. Dietz and Raman[9] showed that small ranks can be computed faster on the CRCW PRAM. Their show $\Theta(\log\log n + \log k/\log\log n)$ time for selecting the k-th smallest element.

Although much progress has been made on the CRCW model the situation on the EREW PRAM seemed doomed. Since Cole published his result in 1988 [7] it is only a factor of $\log^* n$ away to the optimal time and operation bound on the EREW PRAM. For more than a decade no progress has been significant enough to bridge the gap between Cole's algorithm and the simultaneous time and operation optimal bounds ($O(\log n)$ time and $n/\log n$ processors). A recent paper by Ramnath and Raman[15] used similar techniques as used by Cole[7] and only extended Cole's work. Cole's main techniques [7] are sorting and accelerated sampling. Cole's techniques, although bringing the current best selection algorithm on the EREW PRAM, are probably not powerful enough for achieving optimal time and operation bound.

In this paper we basically give up the attempt of using the sampling technique to achieve

optimality and invent a totally new technique. We build a partial order to accomplish selection with simultaneous optimal time and operations, i.e. our EREW PRAM selection algorithm runs in $O(\log n)$ time using $n/\log n$ processors. This matches the time lower bound given in [8] and the operation lower bound of $\Omega(n)$ due to linear time bound of sequential algorithms for selection. Our result therefore closes the parallel selection problem on the PRAM model.

2. Preliminary

Although sampling is the key technique for selection in the sequential case[4] and used in previous parallel algorithm designs[5, 7], our success is not built on the sampling technique. When sampling is used a significant portion of the elements are left out which create the uncertain factor in determining the rank of the to-be selected element. In selecting the median the sampling technique only returns an element guaranteed to be ranked within $[n/2 - n/c, n/2 + n/c]$, where c is a constant, among the n elements in $O(\log n)$ time and linear operations. Thus the number of elements left to be continued to work on is $n - 2n/c$. Such a remaining set is OK in the sequential case, but it is too large in the parallel case and prevents the achievement of an optimal algorithm.

We cannot build n/t ordered chains each having t elements for a nonconstant t with optimal operations. Thus the natural way to circumvent the problem is to use partial order. Our goals of building a partially ordered set are first to ensure that with this partial order we are selecting among a set with size smaller than n. A set with size $O(n/\log\log n)$ meets our needs because we can then apply Cole's selection algorithm[7] on this set. This goal is relatively easy to achieve. For example we can, as in the sequential case, put $\log\log n$ elements in a row and form $n/\log\log n$ rows. First select the median of each row. The total number of medians is $n/\log\log n$. This set of medians E then meets our needs because we can use Cole's algorithm[7] to work on this set. Our second goal is to ensure that after we select from the reduced set we can eliminate a large amount of elements. For example, if we are to select the median and we do this by selecting the median in set E. Then we can eliminate only $n/4$ elements. In order to eliminate more elements we add partial order relations among the elements. This partial order is established with the help of expander graphs.

Expander graphs has been extensively studied and widely applied to various computation problems. See [1, 13, 14, 21] for the construction of expanders. We need the following definition. A bipartite graph $G_n = (U, V, E)$ is an (n, k, δ) expander if $|U| = |V| = n$, $|E| \leq kn$ and for any $X \subseteq U$, $|\Gamma_{G_n}(X)| \geq (1 + \delta(1 - \frac{|X|}{n}))|X|$, where $\Gamma_{G_n}(X)$ is the set of vertices in V connected to vertices in X with edges in E. It is known that there exist an (n, k, δ) expander with certain constants k and δ and it can be explicitly constructed. For example, Jimbo and Maruoka[13] give an $(n, 5, 1 - \frac{5}{8}\sqrt{2})$ expander. It is also known that the maximum valency of such an expander is bounded by a constant.

We use the notation $\log^{(1)} n = \log n$ and $\log^{(i)} n = \log\log^{(i-1)} n$, $\log^* n = \min\{i | \log^{(i)} n < 1\}$. Also if every element in set T is no less than (greater than) every element in set S then we use $S \leq T (S < T)$ to denote this fact. We also use $\max(S)$ to denote $\max\{a | a \in S\}$.

We make use of the following known results:

Theorem 1 (Cole[7]): There is an EREW PRAM selection algorithm which runs in $O(\log n \log^* n)$ time and linear ($O(n)$) operations.

We use Theorem 1 in the cases where there are many sets of very small size. For example if we are selecting medians from $n/\log\log n$ sets each of size $\log\log n$ then we can get $O(\log^{(3)} n \log^* n)$ time and $O(n)$ operations.

Theorem 2 (Cole[7]): There is an EREW PRAM selection algorithm which runs in $O(\log n)$ time with $O(n \log^{(i)} n)$ operations, where i is an arbitrarily large constant.

We use Theorem 2 in the cases when we want to select from a set of small size. For example if we are to select from a set of size $n/\log\log n$ we can get $O(\log n)$ time and $O(n \log^{(i)} n/\log\log n) < O(n)$ operations.

Theorem 3 (Chong et al.[6], Ramnath and Raman[15]): There is an EREW PRAM algorithm which selects element of rank $k \leq n/\log^{(i)} n$, where i is an arbitrarily large constant, in $O(\log n)$ time and $O(n)$ operations.

Theorem 3 is obtained by Chong *et al.* in [6] and Ramnath and Raman[15]. Ramnath and Raman formalized it. We use Theorem 3 in cases where the selection of a small rank from an $O(n)$ sized set is needed.

In the selection problem it is known[9, 15] that finding the median is the one which has or tends to have the highest complexity. We therefore first show how to select the median. We then generalize our algorithm to select other ranks.

3. Selecting the Median

We begin by describing the method for finding the median, the rank $n/2$ item; we then show how to extend the method to find the rank r item for any given r.

The algorithm proceeds in two steps. The first step, which is the new contribution in this paper, finds an item m with rank close to $n/2$. The set is then partitioned about m, creating a subset S containing the original median. Without loss of generality suppose that $\text{rank}(m) \leq n/2$. Theorem 3, which finds items of low (or high) rank in $O(\log n)$ time using $O(n)$ operatioins, is now applied to S to find the original median, which has low rank in S.

Now we describe how to find an item with rank close to the median. Without loss of generality assume that $n = 2^k$ for some integer k. The algorithm uses two parameters $a = \theta(\log \log n)$ and $b = \theta(\log^{(6)} n)$; it is convenient to set them to be integer powers of 2.

We use a bipartite graph $G = (U, V, E)$ with $|U| = |V| = n/a$. We place $a/2$ input items at each node of G. U is called the top level and V the bottom level. After a rearranging phase to be given in the next section, we establish the following partial order:

Partial Order: If S_u is the set at node $u \in U$, S_v the set at node $v \in V$, and $(u, v) \in E$, then $S_u < S_v$: that is, for each item $e \in S_u$ and $f \in S_v$, $e < f$.

The edge set E of G is chosen such that the following expansion property holds.

Expansion Property: For each set $W \subset U$ or $W \subset V$, $|W| \geq n/(ab)$, the set $N(W)$ of neighbors of W satisfies $|N(W)| \geq n/a - n/(ab)$.

Definition 4: A near-median is an item with rank in the range $[n/2 - n/b, n/2 + n/b]$.

Lemma 5: Given a collection of $2n/a$ size $a/2$ sets associated with the bipartite graph G satisfying the expansion property and the partial order, a near-median can be found in $O(\log n)$ time using $O(n)$ operations for $a = O(\log^{(i)} n)$ for any constant $i \geq 1$.

Proof: The algorithm simply finds the median of the following set S of $2n/a$ items: the largest item in each of the sets at a top node of G and the smallest item in each of the sets at the bottom node of G. This is computed using Theorem 2 which takes $O(\log n)$ time and $O(n)$ operations if $a = \log^{(i)} n$, for constant $i \geq 1$. It remains to show that the rank of m, the item found, lies in the required range.

First, we show that m has rank at least $n/2 - n/b$. So suppose that among the smallest n/a items in S, s came from bottom sets (and $n/a - s$ from top sets).

Case 1: $s < n/(ab)$.

Then there are at least $(a/2)(n/a - s)$ items no larger than m in the original set of n items; but this is more than $n/2 - n/b$ items, and so the result holds in this case.
Case 2: $n/(ab) \leq s$.

We say that two sets are neighbors if they are located at neighboring nodes in G. Consider the s sets at bottom nodes from which the s items of Case 2 are drawn. Next consider the t sets of neighbors of these s sets, which are at top nodes. All the items in these t sets are less than or equal to m (for any set S_u among these t sets satisfies $S_u < S_v$ where S_v is one of the bottom s sets and S_u and S_v are neighbors). Note that the smallest item in S_v is less than or equal to m. But, by the expansion property $t \geq n/a - n/(ab)$. Thus there are at least $s + ta/2 \geq n/ab + n/2 - n/(2b) > n/2 - n/b$ items in the original set less than or equal to m. In either case m has rank at least $n/2 - n/b$.

A symmetric argument shows that m has rank at most $n/2 + n/b$. □

Lemma 6: Let m be a near-median which has already been found. Then an exact median can be found in $O(\log n)$ time and $O(n)$ operations, for $b = O(\log^{(i)} n)$ for any constant $i \geq 1$.

Proof: First, partition the input items by m. If m is found to have rank $n/2$, then it is the median. Otherwise apply Theorem 3 to the partition containing the original median. □

Lemma 7: There is a bipartite graph of degree $c_1^{2^{O(b)}}$, with $|U| = |V| = n/a$ and a constant c_1, satisfying the expansion property.

Proof: We first build graph G_1 as shown in Fig. 1. We arrange nodes in $2^{O(b)}$ rows and $M = n/a$ columns with M nodes in each row. We say the nodes at i-th row from bottom are at level i. We connect nodes at level i with nodes at level $i+1$, $1 \leq i < 2^{O(b)}$, with an expander which expands the sets of nodes at level i to sets of nodes at level $i+1$ and expands the sets of nodes at level $i+1$ to level i. Note that the degree of each node is bounded by a constant.

We now collapse the levels in graph G_1. We build the expander we needed, i.e. graph G with $2M$ nodes arranged in 2 rows and M columns:

$x_1, x_2, ..., x_M$

$y_1, y_2, ..., y_M$

Where x_i represents the node at the top row and column i in Fig. 1, y_i represents the

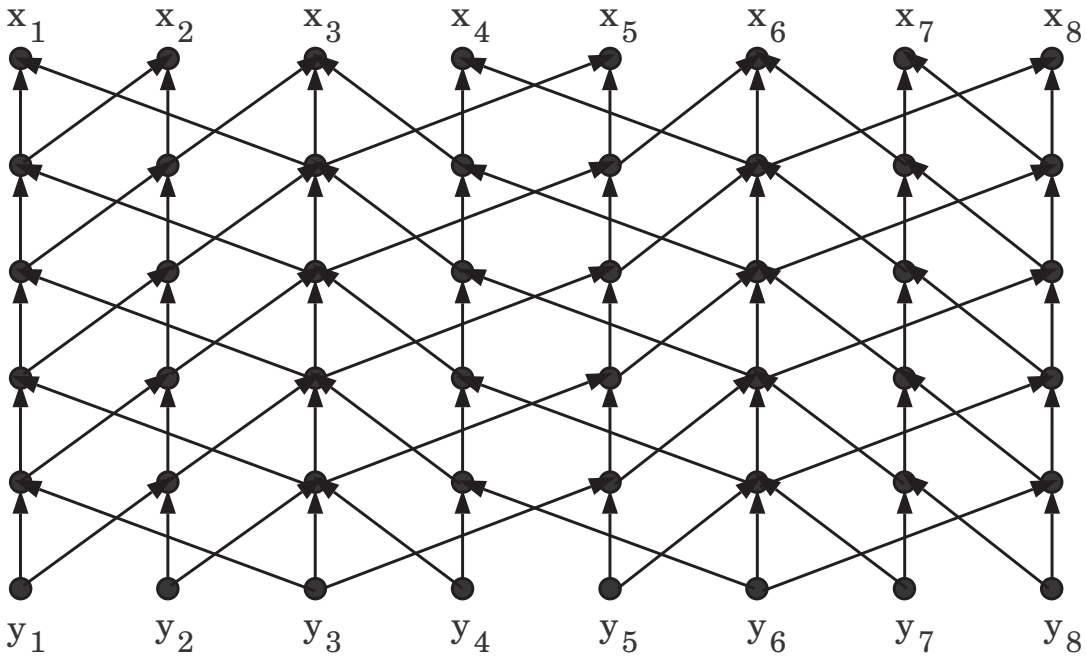

Fig. 1. Graph G_1.

node at the bottom row and column i in Fig. 1. There is an arc $< y_i, x_j >$ in G if in G_1 there is a directed path from the node at the bottom level and column i to the node at the top level and column j.

A set X of nodes expands to a set of size at least $(1+\delta(1-|X|/M))|X|$ in the expander graph which is one level in G_1 of our construction. For a set $|X| \leq M(1-1/b)$ the expanding factor $(1+\delta(1-|X|/M)) \geq (1+\delta/b)$. After expanding for $t = 2^{O(b)}$ times (note that there are t levels in G_1) any set X with $|X| = M/b$ expands to a set of size at least $\min\{M(1-1/b), (M/b)(1+\delta/b)^{2^{O(b)}}\} = M(1-1/b)$.

Because the expanders in G_1 have constant degree for each vertex and the array A has $2^{O(b)}$ rows, the maximum valency in G is $d_1 = {c_1}^{2^{O(b)}}$ for a constant c_1.

□

Lemma 8: Graph G can be built $O(\log n)$ time and $O(n)$ operations provided that $b \leq \log^{(4)} n$.

Proof: The procedure works on the graph G, calculating, for each node in the bottom level,

its distance in the i-th level in G_1, In turn for $i = 2, 3, .$

The procedure constructs a series of intermediate bipartite graph H_i. One bipartition of H_i is identical to the bottom level in G_1, the other to the ith level of G_1. Edges in H_i correspond to length $i-1$ paths in G_1. Let d_i be an upper bound on the node degree in H_i; each node of H_i numbers its up to d_i incident edges using distinct intergers in the range $[1, d_i]$.

H_{i+1} is built using arrays of siza $c_1 d_i$, one for each node in H_{i+1}. There is a processor for each node in the "top" partition of H_i (ith level in G_1). The processor for node v writes the name of each of its c_1 neighbors in the $(i+1)$st level of G_1 to the arrays for (the nodes in H_{i+1} corresponding to) each of its neighbors in H_i. More specifically, for edge (u, v)in H_i which has index h in v's ordering, it writes to array locations $[c_1(h-1)+1 : c_1 h]$ for vertex v in H_{i+1}. Similarly, it writes the names of its up to d_i neighbors in H_i to the nodes in H_{i+1} corresponding to its c_1 neighbors for the $(i+1)$th level of G_1. Finally, each vertex in H_{i+1} sorts its neighbors to remove duplicates and numbers the remainder sequentially.

The claimed complexity bounds are readily obtained. □

4. Establishment of the Partial Order

We shall demonstrate how to build the partial order.

4.1. Heap Operation

Definition 9: A t-minheap (t-maxheap) is a binary tree except that the root has only one child. A set of t elements is associated with each node of the tree. If v is a proper descendant of u then $S_u \leq S_v$ ($S_u \geq S_v$), where S_u and S_v are the sets at u and v respectively. The set at the root is denoted by $\overline{S}$ and the set at the child of the root is denoted by $\overline{\overline{S}}$.

The build heap operation and the Extract_Min operation (which extract out the t smallest elements at the root) for t-minheap (t-maxheap) is similar to the situation in the ordinary case. We show the move down process in the t-minheap.

Let u be a node in the tree of t-minheap. Let v, w be u's children. Let S_u, S_v, S_w be the sets associated with u, v, w. Suppose we are to move S_u down. Assume $\max(S_v) \geq \max(S_w)$. Let $S = S_u \cup S_v \cup S_w$. We partition S into three sets S_1, S_2, S_3, where $|S_1| = |S_2| = |S_3| = t$ and $S_1 \leq S_2 \leq S_3$. We let $S_u = S_1$, $S_v = S_2$ and S_3 is the set placed at w which needs to be moved down.

If the t-minheap (t-maxheap) has height h then Extract_Min (Extract_Max) operation takes $O(ht)$ sequential time.

To obtain a t-minheap from a $(2t)$-minheap we can first partition the set at each node arbitrarily into two equal sized sets. This will give us two t-minheaps. Then we combine the $\overline{S}$ sets of these two t-minheaps into one set A of $2t$ items. Partition A into A_1 and A_2 with $|A_1| = |A_2| = t$ and $A_1 \leq A_2$. Put A_1 at the root (becoming the $\overline{S}$ set) and make A_2 the child of A_1 (A_2 is now the $\overline{\overline{S}}$ set). And making the two $\overline{\overline{S}}$ sets of the two t-minheaps the children of A_2. Thus we obtained a t-minheap from a $(2t)$-minheap. Obtaining a t-maxheap from a $(2t)$-maxheap is similar.

If the $(2t)$-minheap has height h then obtaining a t-minheap from the $(2t)$-minheap takes $O(2^h + t)$ time.

4.2. Build the Partial Order

The most challenging issue is to establish the partial order. Our original contribution toward this are presented in [12]. Richard Cole later provided us with a simplified proof (also given in [12]). The content presented here combines some of our original ideas with Cole's proof.

Without loss of generality it is assumed that all elements are distinct.

The implementation of the subroutine is described next. Initially, the items in each set at a top node (of the partial order) are in an t-maxheap, and those in each set at a bottom node are in an t-minheap. (It is convenient to let t be an integer power of 2.) These take time and work $O(t)$ to set up. At the start of the i-th iteration, the sets will be stored in $t/2^{i-1}$-max or minheaps, respectively. Further, for each edge (u, v), u a top node, v a bottom node, $S_u - \overline{S}_u < S_v - \overline{S}_v$, where S_u is the set of items at node u and $\overline{S}_u$ is the set of items at the root of its heap, and S_v and $\overline{S}_v$ are defined analogously.

In one iteration, the heaps are split and then the relation $S_u - \overline{S}_u < S_v - \overline{S}_v$ is restored.

We construct graph G'. Only vertices $x_1, x_2, ..., x_M$ in G are in G'. x_i and x_j are connected by an edge if there are arcs $< y_k, x_i >$ and $< y_k, x_j >$ in G for some k. The maximum valency in G' is $d_2 = d_1^2$, where $d_1 = c_1^{2^{O(b)}}$.

We use a simple scheme to vertex color graph G'. We first edge color the edges of graph G' with d_2^2 colors as in [10]. It is done by first letting each vertex x_i label d_2 edges incident

with it with $1, 2, ..., d_2$ and then assigning edge (x_i, x_j) with color $l_1 d_2 + l_2$ if (x_i, x_j) receiving lables l_1 and l_2 from x_i and x_j. This uses d_2^2 colors. Such assigned color may generate conflicts in that neighboring edges may obtain the same color. However, edges colored with the same color form linked lists or linked cycles. Using symmetry breaking technique in [11] we can recolor the edges such that neighboring edges receiving different colors. Thus the edge coloring can be done in $O(c_1^{2^{O(b)}})$ time and $O(n)$ operations[11] if $b \geq \log^{(6)} n$. For each edge we then arbitrarily assign label 1 to one of its end point and 2 to the other end point. Now for each vertex v in G' we assign color $(a_1, a_2, ..., a_{d_2^2})$, where $a_i = 1$ if v has an edge colored i and v receive label 1, $a_i = 2$ if v has an edge colored i and v receive label 2, $a_i = 0$ if v does not have an edge colored i. Obviously this is a valid vertex coloring. It uses $d_3 = 3^{d_2^2}$ colors. Obviously this coloring can be done in $O(\log n)$ time and $O(n)$ operations if $b = \theta(\log^{(6)} n)$.

We now color vertices $x_1, x_2, ..., x_M$ in G with the same color we colored vertices $x_1, x_2, ..., x_M$ in G'. In this way no vertex y_i is connected with two different x_j's colored with the same color.

To this end, the bipartite graph's top vertices are partitioned into subsets $U_1, U_2, ..., U_{d_3}$, such that if w and x are both in U_i they have no neighbors in common. Each set U_i is processed in turn. To process U_i means to restore the relations $S_u - \overline{S}_u < S_v - (\overline{S}_v \cup \overline{\overline{S}}_v)$ for each $u \in U_i$ and each neighbor v of u, without destroying it for any other pair (x, v). Let $\overline{\overline{S}}_u$ be the set at the child of the root u's t-heap (recall $\overline{S}_u$ is the set at the root). Let $v_1, v_2, ..., v_d$ be the neighbors of u. Let $\overline{\overline{S}}_{v_i}$ and $\overline{S}_{v_i}$ be defined analogously, $1 \leq i \leq d$. First, find the maximum element in each of the sets $\overline{S}_u, \overline{\overline{S}}_u, \overline{S}_{v_i}$ and $\overline{\overline{S}}_{v_i}$, $1 \leq i \leq d$. If $\overline{\overline{S}}_u$ has the smallest maximum, then $S_u - \overline{S}_u < S_{v_i} - \overline{S}_{v_i}$, $1 \leq i \leq d$, as $\overline{\overline{S}}_u < \max_x\{x \in \overline{S}_{v_i}\} < \overline{\overline{S}}_{v_i}$. Otherwise, without loss of generality, suppose $\overline{S}_{v_1}$ has the smallest maximum; the set $\overline{\overline{S}}_u$ replaces $\overline{S}_u$, $\overline{S}_u$ replaces $\overline{S}_{v_1}$, and $\overline{S}_{v_1}$ replaces $\overline{\overline{S}}_u$. Note that the new $\overline{\overline{S}}_u$ satisfies $\overline{\overline{S}}_u < \max_x\{x \in \overline{\overline{S}}_{v_i}\}$, for $i > 1$, and clearly $S_u - \overline{S}_u < S_{v_i} - (\overline{S}_{v_i} \cup \overline{\overline{S}}_{v_i})$, $1 \leq i \leq d$, now, and further all relationships $S_x - \overline{S}_x < S_v - (\overline{S}_v \cup \overline{\overline{S}}_v)$ that held previously continue to hold, as these sets $(S_x - \overline{S}_x, S_v - (\overline{S}_v \cup \overline{\overline{S}}_v))$ are unchanged. But the t-heaps at u and v_1 may need to be restored, which is done by reheapnization. At u only smaller items will be found in the root's proper descendants as a result of the reheapnization; at v_1, only larger elements.

Now the bipartite graph's bottom vertices are partitioned into subsets $V_1, V_2, ..., V_{d_3}$, such

that if w and x are both in V_i they have no neighbors in common. Each set V_i is processed in turn. To process V_i means to restore the relations $S_u - \overline{S}_u < S_v - \overline{S}_v$ for each $v \in V_i$ and each neighbor u of v, without destroying it for any other pair (u, y). Let $u_1, u_2, ..., u_d$ be the neighbors of v. Find the minimum element in each of the sets $\overline{S}_v, \overline{\overline{S}}_v, \overline{S}_{u_i}$ and $\overline{\overline{S}}_{u_i}$, $1 \leq i \leq d$. If $\overline{\overline{S}}_v$ has the largest minimum, then $S_v - \overline{S}_v > S_{u_i} - \overline{S}_{u_i}$, $1 \leq i \leq d$, as $\overline{\overline{S}}_v > \min_x\{x \in \overline{S}_{u_i}\} > \overline{\overline{S}}_{u_i}$. Otherwise, without loss of generality, suppose $\overline{S}_{u_1}$ has the largest minimum; the set $\overline{\overline{S}}_v$ replaces $\overline{S}_v$, $\overline{S}_v$ replaces $\overline{S}_{u_1}$, and $\overline{S}_{u_1}$ replaces $\overline{\overline{S}}_v$. The new $\overline{\overline{S}}_v$ satisfies $\overline{\overline{S}}_v > \min_x\{x \in \overline{S}_{u_i}\} > \overline{\overline{S}}_{u_i}$, for $i > 1$, and $S_v - \overline{S}_v > S_{u_i} - \overline{S}_{u_i}$, $1 \leq i \leq d$, and all relationships $S_y - \overline{S}_y > S_u - \overline{S}_u$ or $S_y - (\overline{S}_y \cup \overline{\overline{S}}_y) > S_u - \overline{S}_u$ that held previously continue to hold. The t-heaps at v and u_1 may need to be restored, which is done by reheapnization. After that the relation $S_u - \overline{S}_u < S_v - \overline{S}_v$ has been restored.

Theorem 10: The parital order can be established in $O(\log n)$ with with $O(n)$ operations if $a = \theta(\log \log n)$ and $b = \theta(\log^{(6)} n)$. □

5. Selecting a rank r satisfying $n/\log \log n \leq r \leq n/2$

Since by Theorem 3 we know a rank $\leq n/\log \log n$ can be selected optimally we now consider only the selection of rank r satisfying $n/\log \log n \leq r \leq n/2$.

In the partial order we now put sets of size a at the top level and sets of size $(n/r - 1)a$ at the bottom level.

The partial order can be established similarly as in Theorem 10. Here the min-heap at a bottom node is larger than the max-heap at a top node. Otherwise the procedure is basically the same.

Theorem 11: The partial orders can be built in $O(\log n)$ time and $O(n)$ operations. □

Theorem 12: The selection of an element of rank m satisfying $r - r/b \leq m \leq r + n/b - r/b$ can be computed on the EREW PRAM in $O(\log n)$ time and $O(n)$ operations.

Proof: We first build the partial order as stated above and then select the median m in the set S of $2r/a$ items: r/a items are the largest item in each of the sets at a top node of G and r/a items are the smallest item in each of the sets at a bottom node of G. We apply Theorem 2 and achieve $O(\log n)$ time and $O(n)$ operations for finding m.

Now we claim that $\text{rank}(m) \geq r - r/b$.

Suppose that among the smallest r/a items in S, s come form bottom sets.

Case 1: $s \leq r/(ab)$.

There are $a(r/a - r/ab)$ items no larger than m. This is at least $r - r/b$ items.

Case 2: $s \geq r/(ab)$.

By the expansion property we know that there are at least $(r/a)(1 - 1/b)$ sets at the top level with elements no larger than m. Therefore there are $a(r/a)(1 - 1/b) = r - r/b$ items no larger than m.

A similar argument shows that m has rank at most $r + n/b - r/b$. The number of items that are larger than m is at least $(n/r - 1)a(r/a - r/(ab)) = n - r - n/b + r/b$. □

6. Conclusion

Although the selection problem on the CRCW PRAM has been solved, our constructs works at least for some cases of selection on the CRCW PRAM (for example, selecting the median). Our constructs may also be applied to approximate selection and multi-selection. These require further investigation.

Acknowledgment

We would like to thank Professor Richard Cole for bringing his proof to our attention. We also very much grateful to an anonymous referee who suggested many modifications to our original paper[12].

References

[1] M. Ajtai. Recursive construction for 3-regular expanders. *Combinatorica*, vol. 14, No. 4, 379-416(1994).

[2] M. Ajtai, J. Komlós, W. L. Steiger, E. Szemerédi. Optimal parallel selection has complexity $O(\log \log n)$. *Journal of Computer and System Sciences*, 38 (1989), pp. 125-133.

[3] Y. Azar, N. Pippenger. Parallel selection. *Discrete Applied Mathematics*, 27 (1990), pp. 45-58.

[4] M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, **7**, (1973), pp. 448-461.

[5] S. Chaudhuri, T. Hagerup, R. Raman. Approximate and exact deterministic parallel selection. *Proc. Mathematical Foundations of Computer Science*, Springer-Verlag, 1993, pp. 352-361.

[6] K. W. Chong, Y. Han, Y. Igarashi, T. W. Lam. Improve parallel computation with fast integer sorting. *Proceedings of the 5th International Conference on Computing and Combinatorics, Lecture Notes in Computer Sciecne* 1627, 452-461(1999).

[7] R. Cole. An optimally efficient selection algorithm. *Information Processing Letters*, **26** (1988), pp. 295-299.

[8] S. Cook, C. Dwork, R. Reischuk. Upper and lower time bounds for parallel random access machines without simultanous write. *SIAM J. Comput.*, Vol. 15, No. 1, Feb. 1986. pp. 87-97.

[9] P. Dietz, R. Raman. Small-rank selection in parallel, with applications to heap construction. *J. Algorithms* 30(1), 1999, 33-51.

[10] M. Fürer and B. Raghavachari. Parallel edge coloring approximation. *Parallel Processing Letters* **6** (1996), 321–329.

[11] Y. Han. Matching partition a linked list and its optimization. *Proc. 1st Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'89)*, 1989, 246–253.

[12] Y. Han. Optimal parallel selection. *Proc. 2003 ACM-SIAM Symposium on Discrete Algorithms (SODA'03)*, 1-9(2003).

[13] S. Jimbo, A. Maruoka. Expanders obtained from affine transformations. *Combinatorica*, Vol. 7, no. 4, 343-355(1987).

[14] G. A. Magulis. Explicit construction of concentrators. *Problemy Peredachi Informatsii*, vol. 9, no. 4, 71-80(1973). (English translation in *Problems of Inform. Transmission*, 325-332(1975)).

[15] S. Ramnath, V. Raman. Selecting small ranks in EREW PRAM. *Information Processing Letters* 71, 183-186(1999).

[16] R. Reischuk. Probabilistic parallel algorithms for sorting and selection. *SIAM J. on Computing*, 14 (1985), pp. 396-409.

[17] H. Shen. Optimal parallel multiselection on EREW PRAM. *Parallel Computing*, 23(1997), 1987-1992.

[18] M. Snir. On parallel searching. *SIAM J. Comput.*, 14, 3(Aug. 1985), pp. 688-708.

[19] L. G. Valiant. Parallelism in comparison problems. *SIAM Journal on Computing*, 4 (1975), pp. 348-355.

[20] R. A. Wagner, Y. Han. Parallel algorithms for bucket sorting and the data dependent prefix problem. *Proc. 1986 Int. Conf. on Parallel Processing*, 924-930(August 1986).

[21] A. Wigderson, D. Zukerman. Expanders that beats the eigenvalue bound: explicit constructions and applications. *Proc. 1993 Symposium on Theory of Computing*, 245-251(1993).