アルゴリズム 51－5
（1996. 5. 29）

ソート集合のある分割に対する並列アルゴリズムについて

Danny Z. Chen* 陳 慰† 和田 幸一† 川口 喜三男†

＊ノートルダム大学 計算機科学科
電子メール chen@cse.nd.edu
† 名古屋工業大学 電気情報工学科
電子メール [chen|wada|kawaguchi]@elcom.nitech.ac.jp

要旨：本稿では次のような分割問題を考える。要素数 n の集合 S が k 個の列からなりそれぞれソートされた n/k 個の集合として与えられているとき、パラメタ $h(1/k \leq h \leq n/k)$ に対して S を $|D_i| = \Theta(hk)$ であり、$1 \leq i < j \leq g$ なる任意の添字 i と j に対して D_i のどの要素も D_j のどの要素より大きくないように $g = O(n/(hk))$ 個の部分集合 $D_1, D_2, \ldots, D_g$ に分割する。この分割問題は k と h をうまく設定することにより、マージ、ソートや近似の中間値などを求める問題を含んでおり、また計算幾何学の問題へも応用できる。本稿では、この分割問題を解く並列アルゴリズムを示す。この並列アルゴリズムは EREW PRAM モデルにおいて $O(\frac{\min\{(n/h)*\max\{\log h,1\},n*\max\{\log(1/h),1\}\}}{\log n})$ 個のプロセッサを用いて $O(\log n)$ 時間で分割問題を解く。

Parallel Algorithms for Partitioning Sorted Sets and Related Problems

Danny Z. Chen* **Wei Chen†** **Koichi Wada†** **Kimio Kawaguchi†**
*Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556, USA.
e-mail:chen@cse.nd.edu
†Nagoya Institute of Technology
Gokiso-cho, Syowa-ku, Nagoya 466, JAPAN
e-mail:[chen|wada|kawaguchi]@elcom.nitech.ac.jp

Abstract: We consider the following partition problem: Given a set S of n elements that is organized as k sorted subsets of size n/k each and given a parameter h with $1/k \leq h \leq n/k$, partition S into $g = O(n/(hk))$ subsets $D_1, D_2, \ldots, D_g$ of size $\Theta(hk)$ each, such that for any two indices i and j with $1 \leq i < j \leq g$, no element in D_i is bigger than any element in D_j. In this paper, we present efficient parallel algorithms for solving the partition problem and its applications. Our parallel partition algorithm runs in $O(\log n)$ time using $O(\frac{\min\{(n/h)*\max\{\log h,1\},n*\max\{\log(1/h),1\}\}}{\log n})$ processors in the EREW PRAM model.

1 Introduction

We consider the following partition problem: Given a set S of n elements that is organized as k sorted subsets (called *columns*) $C_1, C_2, \ldots, C_k$ of size n/k each and given a parameter h with $1/k \leq h \leq n/k$, partition S into $g = O(n/(hk))$ subsets $D_1, D_2, \ldots, D_g$ of size $\Theta(hk)$ each, such that for any two indices i and j with $1 \leq i < j \leq g$, no element in D_i is bigger than any element in D_j. We call such a set sequence of $D_1, D_2, \ldots, D_g$ an *ordered set sequence*, and call the sets D_i *ordered sets*.

Note that the desired partition does not require that the sizes of the resulted subsets D_i be equal to each other, and such a partition is often sufficient to solving many problems. With various combinations of the values of parameters h and k, several fundamental problems can be formulated as or be reduced to this partition problem. For example, by letting $k = 2$ and $h = 1$, the problem of merging two sorted sequences can be reduced to the partition problem. By letting $k = n$ and $h = 1/k$, the partition problem becomes that of sorting an arbitrary set. By letting $k = n$ and $h = 1/2$, the partition problem becomes that of finding an approximate median in an unsorted set. The partition problem also finds applications in solving problems of parallel computing and computational geometry. In this paper, we present

efficient parallel algorithms for solving the partition problem and its applications.

The parallel computational model we use is the EREW PRAM [13, 15]. We denote the time $\times$ processor product of a parallel algorithm as its *work* (i.e., the total number of operations performed by the parallel algorithm). When analyzing the complexity bounds of our parallel algorithms, we often give only their time and work bounds; the desired processor bounds of all our algorithms in this paper follow from Brent's theorem [5].

Except for the special cases such as merging, sorting, and finding an approximate median, we are not aware of any previous sequential and parallel algorithms for the general version of the partition problem. A notable related work is due to Frederickson and Johnson [11] on sequential selection and ranking among sorted columns. Our results on the partition problem are as follows.

- Our parallel partition algorithm takes $O(\log n)$ time and $O(\min\{(n/h)*\max\{\log h, 1\}, n*\max\{\log(1/h), 1\}\})$ work on the EREW PRAM. Note that the complexity bounds of this parallel partition algorithm on the respective special cases match those of the optimal EREW PRAM algorithms for merging [3, 6, 12], sorting [1, 9], and finding an approximate median [8]. Our parallel solution also implies that the partition problem can be solved sequentially in $O(\min\{(n/h)*\max\{\log h, 1\}, n*\max\{\log(1/h), 1\}\})$ time.

A word on the representations of the resulted ordered subsets D_j of the partition is in order. When $h \leq 1$, the subsets D_j are actually constructed from the set S. But when $h > 1$, the subsets D_j are represented *implicitly*: Each D_j is described by using the sorted columns C_i, because for every such column C_i, it is sufficient to describe the elements of $D_j \cap C_i$ with at most two elements of C_i (hence each D_j can be described implicitly by using $O(k)$ elements of S). This implicit representation is necessary to obtaining the desired complexity bounds for the case with $h > 1$.

We expect that our parallel partition algorithm would be useful in solving many other problems. In particular, we show how to use our parallel partition algorithm to improve the complexity bounds of the previously best known parallel algorithms for the following important problems.

- For an n-element set U and a value a (a need not be in U), we say that the *rank* of a in U, denoted as $rank_U(a)$, is j, $j \in \{0, 1, \ldots, n\}$, if a is no smaller than the j-th smallest element of U and is less than the $(j+1)$-th smallest element of U (with the 0-th smallest element of U being $-\infty$). Given the set S with k sorted columns C_i and a sequence A of m values $a_1, a_2, \ldots, a_m$ such that $a_1 < a_2 < \cdots < a_m$, $1 \leq m \leq n$, the *multi-ranking problem* is to find $rank_S(a_l)$ for each a_l in A. A related *multi-search problem* is to find, for each a_l in A, the unique element b of S such that $rank_S(b) = rank_S(a_l)$. Let $h = n/(km)$. Then $1/k \leq h \leq n/k$. A sequential $O(\min\{(n/h) * \max\{\log h, 1\}, n * \max\{\log(1/h), 1\}\})$ time algorithm for the multi-ranking and multi-search problems is possible. We are not aware of any parallel algorithm designed specifically for these two problems, but the following two simple parallel solutions are not difficult to obtain: Using the parallel merging algorithms [3, 6, 12], the two problems can be solved in $O(\log n)$ time and $O(n + km) = O(n + n/h)$ work on the EREW PRAM; using the CREW PRAM algorithm for sorting k sorted columns [26], the two problems can be solved in $O(\log n)$ time and $O(n \log k)$ work on the CREW PRAM. Using our parallel partition algorithm, we solve these two problems in $O(\log n)$ time and $O(\min\{(n/h) * \max\{\log h, 1\}, n * \max\{\log(1/h), 1\}\})$ work on the EREW PRAM. In the same complexity bounds, our parallel multi-search and multi-ranking algorithms can be used to partition S into $m + 1$ ordered subsets by using the m values in A. Since $1/h \leq k$, when $1/k < h < 1$ or $1 < h < n/k$, our algorithms can perform less work than the merge-based and sorting-based approaches, while using the weakest PRAM model.

- Given an unsorted set U of n elements and a sequence Q of integers $q_1, q_2, \ldots, q_m$ such that $1 \leq q_1 < q_2 < \cdots < q_m \leq n$, the *multi-selection problem* is to find all the q_1-th, q_2-th, $\ldots$, q_m-th smallest elements in U. This problem is clearly a generalization of the well-known selection problem [4], and finds applications in several areas (e.g., databases [20]). A sequential $O(n \log m)$ time algorithm for the multi-selection problem is easy and is in fact optimal. Olariu and Wen [22] solved this problem in $O(\log n \log^* n \log m)$ time and $O(n \log m)$ work on the EREW PRAM. Olariu and Wen's algorithm [22] is based on a divide-and-conquer strategy and on Cole's parallel selection algorithm [8]. Using our parallel partition algorithm, we solve the multi-selection problem in $O(\log n + \log(n/m) \log^*(n/m))$ time and $O(n \log m)$ work on the EREW PRAM. In the same complexity bounds, our parallel multi-selection algorithm can be used to partition U into $m + 1$ ordered subsets based on the ranks in Q. Hence we improve the time bound of the previously best known parallel multi-selection algorithm [22] by a factor of at least $\log m$. We also consider a problem related to multi-selection, which we call *approximate multi-selection*: Given the same input as the multi-selection problem, partition U into $O(m + 1)$ ordered subsets U_i of size $\Theta(n/(m+1))$ each, and find, for every rank $q_j \in Q$, the subset U_i that contains the q_j-th smallest element of U. The approximate multi-selection problem can be viewed as a generalization of the approximate median problem for which Cole gave an $O(\log n)$ time, $O(n)$ work EREW PRAM algorithm [8]. (In the approximate median problem, $m = 1$ and an element in the subset containing the $(n/2)$-th smallest element can be used as an approximate median.) We solve the approximate multi-selection problem in $O(\log n)$ time and

$O(n \log m)$ work on the EREW PRAM.

• The problem of sorting the set S from its k sorted columns is of theoretical interest and finds applications in several areas (e.g., information retrieval systems [25]). Sequentially, it is known how to sort such a set S optimally in $O(n \log k)$ time [16]. Optimal EREW PRAM algorithms are known for the special cases of this sorting problem with $k = n$ [1, 9] and $k = O(1)$ [3, 6, 12]. However, for the general version of this sorting problem (with $O(1) < k < n$), the best known EREW PRAM algorithms either take $O(\log n)$ and $O(n \log n)$ work (by simply using [1, 9]) or take $O(\log n \log k)$ time and $O(n \log k)$ work (by repeatedly using the $O(\log n)$ time, $O(n)$ work EREW PRAM merging algorithms [3, 6, 12]), and are clearly not optimal in either their time or work bound[1]. Recently, Wen [26] gave an $O(\log n)$ time, $O(n \log k)$ work algorithm for this sorting problem on the stronger CREW PRAM. Wen's CREW PRAM sorting algorithm is based on Cole's cascading divide-and-conquer technique [9]. Although Wen's algorithm is work optimal, it is not necessarily time optimal on the CREW PRAM. When $k = o(2^{\log n / \log \log n})$, for example, it is easy to obtain a $o(\log n)$ time, $O(n \log k)$ work algorithm on the CREW PRAM for this sorting problem, by making use of the optimal $O(\log \log n)$ time, $O(n)$ work CREW PRAM merging algorithms [13, 17]. Furthermore, although the cascading divide-and-conquer technique is elegant at designing numerous PRAM algorithms [2], it is still a challenging issue to implement the cascading divide-and-conquer based algorithms on existing parallel computers. In addition, simulating Wen's CREW PRAM algorithms on the EREW PRAM is likely to increase the time bound by a factor of $\log n$. Our algorithm sorts the k sorted columns in $O(\log n)$ time and $O(n \log k)$ work on the EREW PRAM, improving the time or work bound of the previously best known EREW PRAM algorithms and using the weaker EREW PRAM model than the best known CREW PRAM algorithm [26]. Furthermore, our solution need not rely on the cascading divide-and-conquer (although our partition algorithm does do sorting, the sorting operation is used in our solution merely as a black box). Hence, our approach is possibly of more practical value than the CREW PRAM algorithm [26].

• Chen, Wada, and Kawaguchi [7] presented an interesting parallel technique for computing the convex hull of n planar discs. The partition problem is one of the key operations to this technique, and was performed in [7] in $O(\log n)$ time and $O((n \log n)/f(n))$ work on the CREW PRAM, where $f(n)$ is a chosen integer such that $1 \leq f(n) \leq \log \log n$. Using our parallel partition algorithm, this operation can be performed in $O(\log n)$ time and $O(n)$ work on the weaker EREW PRAM.

Without loss of generality (WLOG), we assume that the elements of S are distinct (otherwise, ties can be broken based on the column indices and the positions of the elements in the columns), and that each column C_i is represented as an array with its elements in increasing order. Due to the space limitation, we focus in this extended abstract our parallel partition algorithm and omit the algorithms for all the "application" problems. We also omit the proofs of the lemmas.

2 Useful Building Blocks and a Preliminary Algorithm

Our parallel partition algorithm makes use of several procedures. One such procedure is a parallel solution for the *weighted median problem* [14], which we review as follows: Given an unsorted set A of n distinct elements $a_1, a_2, \ldots, a_n$, with each a_i having a nonnegative weight w_i, find the element a_j in A such that A is partitioned into two ordered subsets A_1 and A_2 with each element of A_1 (resp., A_2) $<$ (resp., $\geq$) a_j, and such that $\sum_{a_i \in A_1} w_i \leq (\sum_{a_i \in A} w_i)/2 < w_j + \sum_{a_i \in A_1} w_i$. The special case of this problem in which every weight w_i is 1 is the well-known (unweighted) median selection problem [4]. The weighted median problem can be solved sequentially in $O(n)$ time by using the linear time unweighted median selection algorithm [4]. By making use of Cole's approximate unweighted median selection algorithm on the EREW PRAM [8], the weighted median problem can be solved in parallel, as follows:

1. Find an approximate (unweighted) median a_z of A by using Cole's algorithm [8], in $O(\log n)$ time and $O(n)$ work.
2. Use a_z to reduce the problem to a subset A' of A, with $|A'|$ being a constant fraction of $|A|$.
3. If $|A'| = O(n/\log n)$, then sort A' by using a sorting algorithm [1, 9], and find the weighted median a_j from the sorted set A' (by performing a parallel prefix [18, 19] on the weights of the elements in the sorted set A'); otherwise, recursively solve the problem on A'.

The above parallel weighted median algorithm takes $O(\log n \log \log n)$ time and $O(n)$ work on the EREW PRAM, because it makes $O(\log \log n)$ recursive calls, with each call taking $O(\log n)$ time and performing a constant fraction of the amount of work of the previous recursive call.

The following observation is a key to our parallel partition algorithm.

Lemma 1 *Suppose that a set S is organized as k sorted columns C_i of size n/k each. For each $i = 1, 2, \ldots, k$, let C_i' be the subset of C_i that consists of every s-th element of C_i (i.e., the s-th, $(2s)$-th, ..., $(\lfloor n/(ks) \rfloor s)$-th elements of C_i). Let*

[1] Very recently, Nakano and Olariu [21] obtained an $O(\log n)$ time, $O(n \log k)$ work EREW PRAM algorithm for the restricted case of $k = \Theta((\log n)^{\epsilon})$ for any constant $\epsilon > 0$, by using a scheme called sampling and bucketing [23].

$S' = \bigcup_{i=1}^{k} C'_i$. *If x (resp., y) is the α-th (resp., β-th) smallest element of S', with $x < y$, then there are at most $s(\beta - \alpha + k - 1)$ elements of S that are in between x and y (i.e., they are $\geq x$ but $\leq y$).*

Based on the above parallel weighted median algorithm, we first obtain a preliminary parallel algorithm for the special case of the partition problem in which $1 \leq h \leq n/k$. Note that in this case, the sizes of the resulted ordered subsets are $\Theta(hk) = \Omega(k)$. This preliminary parallel algorithm takes $O(\log(n/(hk)) * (\log k \log\log k + \log(n/k)))$ time and $O((n/h) * \log h)$ work on the EREW PRAM. Although the time bound of the preliminary algorithm is not very efficient, we illustrate it here because it is a useful procedure for our final solution to the general partition problem.

The preliminary parallel partition algorithm works on sequences of consecutive blocks of the sorted columns. For each $i = 1, 2, \ldots, k$, let B_i be a consecutive block of C_i and $n_i = |B_i|$. Initially, $B_i = C_i$ and $n_i = n/k$. The algorithm proceeds as follows.

1. Find the middle element b_i of B_i and associate with b_i a weight n_i. Let WM be the set of the k such b_i's.
2. Obtain the weighted median q of WM, in $O(\log k \log\log k)$ time and $O(k)$ work.
3. Use the element q to partition each B_i into two consecutive blocks B'_i and B''_i, such that each element in B'_i (resp., B''_i) is $\leq$ (resp., $>$) q. (Note that B'_i or B''_i can be empty.) This is done by performing a binary search in each B_i for q, and it takes $O(1+\max\{\log|B_i| \mid i = 1,2,\ldots,k\}) = O(\log(n/k))$ time. (Actually, in some extreme cases, parallel merging [3, 6, 12] instead of parallel binary search will be used in this step; more on these extreme cases in Lemma 4.) Assuming that on each (possibly empty) block B_i, the binary search performs $O(1+\max\{\log|B_i|, 0\})$ work, this step takes altogether $O(k+\sum_{i=1}^{k}\max\{\log|B_i|, 0\})$ work.
4. If $\sum_{i=1}^{k}|B'_i|$ (resp., $\sum_{i=1}^{k}|B''_i|$) $> c * hk$ for some chosen constant $c \geq 1$, then recursively solve the problem on the sequence of blocks B'_i (resp., B''_i).

It is clear from the above algorithm that no element in $\bigcup_{i=1}^{k} B'_i$ is larger than any element in $\bigcup_{i=1}^{k} B''_i$. The first three steps of the algorithm altogether take $O(\log k \log\log k + \log(n/k))$ time and $O(k + \sum_{i=1}^{k}\max\{\log|B_i|, 0\})$ work, and, as to be shown next, the algorithm stops after $O(\log(n/(hk)))$ levels of recursion. Hence the $O(\log(n/(hk)) * (\log k \log\log k + \log(n/k)))$ time bound of the algorithm follows. The algorithm clearly uses an EREW PRAM because at each of its recursion levels, a sequence of blocks B_i does not share any element of S with any other sequence of blocks at the same level (if such a block sequence exists). What remains to be done for our analysis is to: (i) show that when the algorithm stops its recursion at a sequence of blocks B'_i (resp., B''_i), the total sum of sizes of the blocks is $\Theta(hk)$, (ii) show that the algorithm stops after $O(\log(n/(hk)))$ levels of recursion, (iii) analyze the work bound of the algorithm, and (iv) discuss how desired processor bounds can be obtained from these time and work bounds.

(i) and (ii) follow from the next lemma.

Lemma 2 *Let the blocks B_i, B'_i, and B''_i be defined as in the above preliminary parallel partition algorithm. Then the following holds: $\frac{1}{4}\sum_{i=1}^{k}|B_i| \leq \sum_{i=1}^{k}|B'_i| \leq \frac{3}{4}\sum_{i=1}^{k}|B_i|$, and $\frac{1}{4}\sum_{i=1}^{k}|B_i| \leq \sum_{i=1}^{k}|B''_i| \leq \frac{3}{4}\sum_{i=1}^{k}|B_i|$.*

It is more difficult to show (iii) and (iv). We need to compute the sum of $O(k + \sum_{i=1}^{k}\log|B_i|)$ work for each block sequence over all sequences of all recursion levels of the algorithm. It turns out that the sum of the part $O(\sum_{i=1}^{k}\log|B_i|)$ over all sequences of all recursion levels of the algorithm is the dominating factor of the total work of the algorithm.

It is clear from the description of the algorithm that at recursion level j, the algorithm processes simultaneously at most 2^j sequences of disjoint blocks, and that every column C_i contributes at most one non-empty block to each such block sequence. Instead of calculating directly the total sum of $O(\sum_{i=1}^{k}\log|B_i|)$ over *each* individual sequence of blocks B_i at level j, we calculate the work of *all* these 2^j sequences *performed on every column C_i* at level j. The total work of each recursion level is then the sum of work performed over all the k columns. Thus, at recursion level j, letting C_i be partitioned into $r_j \leq 2^j$ disjoint non-empty blocks $B_{i,1}, B_{i,2}, \ldots, B_{i,r_j}$ with sizes $n_{i,1}, n_{i,2}, \ldots, n_{i,r_j}$, respectively, we want to find a tight upper bound for $\sum_{s=1}^{r_j}\log|B_{i,s}| = \sum_{s=1}^{r_j}\log n_{i,s}$. The lemmas below together prove such an upper bound.

Lemma 3 *At level j, $0 \leq j \leq O(\log(n/(hk)))$, suppose the preliminary parallel partition algorithm partitions a column C_i of size $m = n/k$ into $r_j \leq 2^j$ disjoint non-empty blocks $B_{i,1}, B_{i,2}, \ldots, B_{i,r_j}$ with sizes $n_{i,1}, n_{i,2}, \ldots, n_{i,r_j}$, respectively. If $m/2^j \geq 4$, then $\sum_{s=1}^{r_j}\log n_{i,s} \leq 2^j \log(m/2^j)$.*

Lemma 4 *At level j, $0 \leq j \leq O(\log(n/(hk)))$, the preliminary parallel partition algorithm performs a total of $O(k2^j * \log(m/2^j))$ work, where $m = n/k$.*

Lemma 5 *The preliminary parallel partition algorithm performs a total of $O((n/h) * \log h)$ work.*

We still need to discuss how desired processor bounds can be obtained from the time and work bounds shown above. We make use of Brent's theorem [5]. There are actually two qualifications to Brent's theorem before one can apply it to a

PRAM: (1) One must find out the amount of work W_l done by each parallel step l, in time $O(W_l/P)$ and with P processors, and (2) one must know how to assign each processor to its task. The key to applying Brent's theorem [5] to our algorithm is to find out the amount of work W_l done by each parallel step l (i.e., qualification (1)) of the algorithm. We like to leave the details on computing the amount of work W_l to the full paper.

3 The Parallel Partition Algorithm

We classify the partition problem into three cases based on the actual value of the parameter h: (1) $h = 1$ (we call this the *basic case*), (2) $1/k \leq h < 1$, and (3) $1 < h \leq n/k$. In all three cases, our parallel partition algorithm runs in $O(\log n)$ time and performs $O(\min\{(n/h) * \max\{\log h, 1\}, n * \max\{\log(1/h), 1\}\})$ work on the EREW PRAM. Since our solutions for cases (2) and (3) both depend on the solution for the basic case, we present first the parallel solution for the basic case, and then the parallel solutions for cases (2) and (3).

3.1 The Case with $h = 1$

When $h = 1$, the problem is to partition the k sorted columns C_i of size n/k each into $O(n/k)$ ordered subsets of size $\Theta(k)$ each. We solve this case in $O(\log n)$ time and $O(n)$ work. We further classify case (1) into three cases based on the values of n/k and k: (1.a) $n/k > \log n$ and $k > \log n$, (1.b) $n/k \leq \log n$ and $k > \log n$, and (1.c) $n/k > \log n$ and $k \leq \log n$ (in general, we cannot have $n/k \leq \log n$ and $k \leq \log n$). Case (1.a) is more typical than cases (1.b) and (1.c) because both (1.b) and (1.c) can be treated as special cases of (1.a). Hence we present here only the general algorithm (for case (1.a)). It is not hard to modify the general algorithm to solve cases (1.b) and (1.c).

Our general parallel algorithm for case (1.a) consists of four phases: (I) Partition the k columns C_i into $O(n/(k \log n))$ ordered subsets E_a of size $\Theta(k \log n)$ each, (II) partition every subset E_a into $O(k/\log n)$ arbitrary subsets F_b of size $\Theta(\log^2 n)$ each, with each F_b consisting of $\log n$ sorted columns of size $O(\log n)$ each, (III) partition every subset F_b into $\log n$ ordered subsets of size $\Theta(\log n)$ each, and (IV) let every subset E_a (of size $\Theta(k \log n)$) consist of $O(k/\log n)$ "roughly sorted" columns (each column corresponds to a subset F_b that has been partitioned into $\log n$ ordered subsets of size $\Theta(\log n)$ each), and partition E_a into $O(\log n)$ ordered subsets of size $\Theta(k)$ each. We discuss each of these four phases below.

Phase I

The input is a set S with k sorted columns of size n/k each. Phase I consists of the following steps.

1. Choose $n/(k \log n)$ elements from each sorted column C_i, such that these $n/(k \log n)$ chosen elements together partition C_i into $n/(k \log n)$ ordered consecutive blocks of size $\log n$ each. From all the k columns, we have chosen $n/\log n$ elements of S. Let these $n/\log n$ elements form a set Q. This step can be easily done in $O(1)$ time and $O(n/\log n)$ work.
2. Sort Q by a sorting algorithm [1, 9], in $O(\log n)$ time and $O(n)$ work.
3. Choose $n/(k \log n)$ elements from Q such that these $n/(k \log n)$ elements together partition Q into $n/(k \log n)$ ordered subsets of size k each, in $O(1)$ time and $O(n/(k \log n))$ work. Let these $n/(k \log n)$ elements form a sorted set Q'. Then by Lemma 1, for any two consecutive elements $x, y \in Q'$, there are at most $2k \log n$ elements of S that are in between x and y.
4. Use the elements of Q' to partition the set S into $d = n/(k \log n)$ ordered subsets E_1, E_2, ..., E_d of size $O(k \log n)$ each (some subsets E_a possibly contain less than $k \log n$ elements). This partitioning is done by first performing a parallel binary search on each column C_i for all elements of Q', and then for every pair of consecutive elements $x, y \in Q'$, adding up the number of elements of each C_i that are in between x and y, over all the k columns. By using Chen's parallel binary search algorithm [6] and parallel prefix [18, 19], each column C_i can be partitioned with the elements of Q' in $O(\log n)$ time and $O(n \log\log n/(k \log n))$ work. Altogether, this step takes $O(\log n)$ time and $O(n \log\log n/\log n)$ work.
5. Union the sets E_a whose sizes are $< k \log n$ so that the resulted ordered set sequence of the E_a's consists of only sets whose sizes are $\Theta(k \log n)$, as follows. Mark every set E_a in the sequence whose size $e_a = |E_a|$ is $< k \log n$, and group the marked sets of the sequence into blocks of consecutive marked sets. The following is then done on each such block $R_{i,j}$ of consecutive marked sets E_i, E_{i+1}, ..., E_j $(i \leq j)$: Compute the prefix sums $p_l = \sum_{a=i}^{l} e_a$, $l = i, i+1, \ldots, j$ (by parallel prefix [18, 19]). If $p_j < k \log n$, then let $E' = E_i \cup E_{i+1} \cup \cdots \cup E_j$, and union E' with either the preceding or succeeding unmarked set of the block $R_{i,j}$ in the sequence (e.g., let $E_{i-1} = E_{i-1} \cup E'$). If $p_j \geq k \log n$, then for every integer $c = 1, 2, \ldots, \lfloor p_j/(k \log n) \rfloor$, there is a set E_{i_c} in the block $R_{i,j}$ such that $ck \log n \leq p_{i_c} < (c+1)k \log n$ (because the size of every E_l in $R_{i,j}$ is $< k \log n$). Identify such a set E_{i_c} in $R_{i,j}$ for each $c = 1, 2, \ldots, \lfloor p_j/(k \log n) \rfloor$, and partition the block $R_{i,j}$ into sub-blocks of consecutive sets by using the sets $E_{i_{c'}}$, $c' = 2, 4, 6, \ldots, 2\lfloor p_j/(2k \log n) \rfloor$. Union the sets in each such sub-block into a single set

(the size of the resulted set is $< 3k \log n$). If the set resulted from unioning the sets in the last sub-block is of a size $< k \log n$, then union this set with either the preceding set or the succeeding unmarked set of the block $R_{i,j}$. Relabel the sets in the resulted sequence of $O(n/(k \log n))$ ordered sets and still denote these sets as E_a's. As a result, the size e_a of every set E_a in the ordered set sequence is such that $k \log n \leq e_a < 4k \log n$. This step takes $O(\log n)$ time and $O(n/\log n)$ work.

It is easy to see that Phase I takes $O(\log n)$ time and $O(n)$ work.

Phase II

As the result of Phase I, we have obtained a sequence of $d = n/(k \log n)$ ordered subsets $E_1, E_2, \ldots, E_d$ of size $\Theta(k \log n)$ each. Phase II simply partitions every such subset E_a into $O(k/\log n)$ unordered subsets F_b, $b = 1, 2, \ldots, O(k/\log n)$, of size $\Theta(\log^2 n)$ each, as follows.

1. Note that for each set E_a and each column C_i, $E_a \cap C_i$ is a (possibly empty) consecutive block of C_i. We assume WLOG that the size of each such block $E_a \cap C_i$ is $c \log n$ for some nonnegative integer c (if this is not the case, we can implicitly patch at most $\log n - 1$ dummy elements of value $+\infty$ to the end of $E_a \cap C_i$, so that the assumption holds). Note that c need not be a constant integer. With the dummy elements, we have $|E_a| < 5k \log n$ for each E_a. Partition each $E_a \cap C_i$ of size $c \log n$, for some integer $c \geq 1$, into c sorted columns of size $\log n$ each. Hence each set E_a consists of $O(k)$ such sorted columns. Group the $O(k)$ sorted columns of each E_a into $O(k/\log n)$ matrices F_b, $b = 1, 2, \ldots, O(k/\log n)$, with each such matrix F_b consisting of $\log n$ sorted columns of E_a. Note that the matrices F_b are not ordered subsets of E_a.

It is clear that Phase II takes $O(\log n)$ time and $O(n)$ work.

Phase III

As the result of Phase II, we have, for each set E_a, $O(k/\log n)$ (unordered) matrices F_b, such that every matrix F_b of E_a consists of $\log n$ sorted columns of size $\log n$ each. Phase III partitions every matrix F_b of E_a into $\log n$ ordered subsets of size $\log n$ each, as follows.

1. Use the preliminary parallel partition algorithm in Section 2 to partition each matrix F_b into a sequence of $O(\log n)$ ordered subsets G_q of size $\Theta(\log n)$ each. Since $|F_b| = \log^2 n$, this takes $O((\log \log n)^2 \log \log \log n)$ time and $O(\log^2 n)$ work on each F_b. Altogether, this step takes $O((\log \log n)^2 \log \log \log n)$ time and $O(n)$ work.
2. For every F_b, compute the prefix sums of the sizes of the subsets G_q along the ordered subset sequence of F_b. Use the information of these prefix sums to guide an appropriate partitioning of each subset G_q of F_b, so that F_b is partitioned into $\log n$ ordered subsets of size exactly $\log n$ each. Since this partition of F_b requires that each subset G_q of F_b (with $|G_q| = \Theta(\log n)$) be further partitioned into at most $O(1)$ ordered subsets, such a partitioning of G_q can be done by using the sequential linear time selection algorithm [4] on G_q. This step takes $O(\log n)$ time and $O(n)$ work.

Phase III altogether takes $O(\log n)$ time and $O(n)$ work.

Phase IV

As the result of Phase III, we have partitioned, for every set E_a, each of its $O(k/\log n)$ (unordered) matrices F_b into $\log n$ ordered subsets of size $\log n$ each. We can view every such matrix F_b as a "roughly sorted" column whose elements form $\log n$ ordered consecutive blocks (of size $\log n$ each) of the column (the elements in each such block of F_b are not sorted, but this does not matter to our algorithm). Phase IV partitions every E_a (of size $\Theta(k \log n)$) into $O(\log n)$ ordered subsets of size $\Theta(k)$ each.

1. Let every set E_a form a matrix of $O(k/\log n)$ roughly sorted columns, with each column being a subset F_b of E_a. Let a set F'_b contain the $\log n$ elements of the column F_b such that these $\log n$ elements together partition F_b into $\log n$ ordered blocks of size $\log n$ each. Let a set E'_a be the union of the sets F'_b for E_a. Then $|E'_a| = O(k)$.
2. Sort the set E'_a, in $O(\log k)$ time and $O(k \log k)$ work.
3. Choose $O(\log n)$ elements from E'_a such that these $O(\log n)$ elements together partition E'_a into $O(\log n)$ ordered subsets of size $k/\log n$ each, in $O(1)$ time and $O(\log n)$ work. Let these $O(\log n)$ elements form a sorted set E''_a. Then by Lemma 1, for any two consecutive elements x and y of E''_a, there are $O(k)$ elements of the matrix E_a that are in between x and y.
4. Use the elements of E''_a to partition E_a into $O(\log n)$ ordered subsets of size $O(k)$ each. This is done by partitioning every roughly sorted column F_b of E_a with the elements of E''_a, as follow. Merge E''_a with each F'_b by using a parallel merging algorithm [3, 6, 12], in $O(\log \log n)$ time and $O(\log n)$ work. Then the elements of E''_a fall into some of the $\log n$ ordered blocks of F_b (of size $\log n$ each) that are delimited by the elements of F'_b (note that each such block of F_b may be unsorted). For every such block Z of F_b into which some elements of E''_a fall, the following is done. Suppose $c > 0$ elements of E''_a fall into the block Z of F_b. Make $\log n$ copies of the sorted sequence $X(Z, E''_a)$ formed by the c elements of

E''_a that fall into Z, in $O(\log\log n)$ time and $O(c\log n)$ work. For all elements of E''_a, altogether $|E''_a| \times \log n = O(|F_b|) = O(\log^2 n)$ copies of them are made for each column F_b. WLOG, assume the result of the above copy making process for the block Z is a matrix $M(Z, E''_a)$ of size $c \times \log n$, with each row of $M(Z, E''_a)$ being the sequence $X(Z, E''_a)$ of the c elements of E''_a fell into Z. Then for every element z_j in the j-th position of the unsorted block Z, $j = 1, 2, \ldots, \log n$, find the unique l-th element x_l in the sorted sequence $X(Z, E''_a)$ stored at the j-th row of the matrix $M(Z, E''_a)$, such that $x_l \leq z_j < x_{l+1}$ (with $l \in \{0, 1, \ldots, c\}$, $x_0 = -\infty$, and $x_{c+1} = +\infty$). Associate such an element z_j of Z with the l-th element of the j-th row of $M(Z, E''_a)$. Then a parallel prefix on each column of the matrix $M(Z, E''_a)$ for the associated elements from the block Z gives the partition of Z by the elements of $X(Z, E''_a)$. This step takes $O(\log\log n)$ time and $O(n)$ work.

5. Remove all dummy elements of E_a, and union the $O(\log n)$ ordered subsets (of size $O(k)$ each) of E_a as in Step 5 of Phase I, such that each such subset of E_a is of size $\Theta(k)$.

6. If the constant factor for the size of such a subset of E_a is still too big, then Cole's approximate unweighted median selection algorithm [8] can be applied to that subset $O(1)$ times, further partitioning the subset into $O(1)$ ordered subsets of size $\Theta(k)$ each.

Phase IV takes altogether $O(\log n)$ time and $O(n)$ work.

In summary, our parallel algorithm for case (1.a) runs in $O(\log n)$ time and performs $O(n)$ work on the EREW PRAM. The discussion of the correctness of the algorithm has been given in each step of the four phases.

3.2 The Case with $1/k \leq h < 1$

When $1/k \leq h < 1$, the problem is that of partitioning the k sorted columns C_i of size n/k each into $O(n/(hk))$ ordered subsets of size $\Theta(hk)$ ($\leq \Theta(k)$) each. We solve this case in $O(\log n)$ time and $O(n\log(1/h))$ ($\leq O(n\log k)$) work, as follows.

1. Use the algorithm for the basic case to partition the k sorted columns C_i of size n/k each into $O(n/k)$ ordered subsets U_a of size $\Theta(k)$ each, in $O(\log n)$ time and $O(n)$ work.

2. For each subset U_a (of size $\Theta(k)$), partition it arbitrarily into $O(hk)$ subsets V_b of size $1/h$ each, and sort each subset V_b. This step takes $O(\log(1/h))$ time and $O(n\log(1/h))$ work.

3. Let each set U_a form a matrix of $O(hk)$ sorted columns, with every column being one of its sorted subsets V_b (of size $1/h$ each). Use the algorithm for the basic case to partition U_a into $O(1/h)$ ordered subsets of size $\Theta(hk)$ each. This step takes $O(\log k)$ time and $O(n)$ work.

The correctness of this algorithm follows from that of the algorithm for the basic case. Clearly, the algorithm takes altogether $O(\log n)$ time and $O(n\log(1/h))$ work on the EREW PRAM.

3.3 The Case with $1 < h \leq n/k$

When $1 < h \leq n/k$, the problem is that of partitioning the k sorted columns C_i of size n/k each into $O(n/(hk))$ ordered subsets of size $\Theta(hk)$ ($\geq \Theta(k)$) each. We solve this case in $O(\log n)$ time and $O((n/h)\log h)$ ($\geq O(k\log h)$) work, as follows.

1. Choose $n/(hk)$ elements from each sorted column C_i, such that these $n/(hk)$ chosen elements together partition C_i into $n/(hk)$ ordered consecutive blocks of size h each. Let these $n/(hk)$ elements of C_i form a sorted column C'_i. From all the k columns C_i, we have chosen n/h elements, and these n/h elements form a matrix Q of k sorted columns C'_i of size $n/(hk)$ each. This step can be easily done in $O(1)$ time and $O(n/h)$ work.

2. Use the algorithm for the basic case to partition the k sorted columns C'_i (of size $n/(hk)$ each) of Q into $O(n/(hk))$ ordered subsets Q_a of size $\Theta(k)$ each, in $O(\log n)$ time and $O(n/h)$ work. Let Q' be the sorted set of the $O(n/(hk))$ elements of Q that delimit the ordered subsets Q_a of Q. Then for any two consecutive elements x and y of Q', there are $\Theta(k)$ elements of Q that are in between x and y. Furthermore, by Lemma 1, for any two consecutive elements x and y of Q', there are $O(hk)$ elements of S that are in between x and y.

3. Use the elements of Q' to partition S into $O(n/(hk))$ ordered subsets of size $O(hk)$ each. This is done by partitioning each sorted column C_i of S with the elements of Q', as follow. Perform a parallel binary search on each column C_i for all the elements of Q' by using Chen's EREW PRAM algorithm for parallel binary search on sorted arrays [6]. The parallel binary search on each column C_i takes $O(\log n)$ time and $O((n/(hk))\log h)$ work. Hence this step altogether takes $O(\log n)$ time and $O((n/h)\log h)$ work.

4. Union the resulted $O(n/(hk))$ ordered subsets (of size $O(hk)$ each) of S as in Step 5 of Phase I of the algorithm for the basic case, such that each such subset of S is of size $\Theta(hk)$. This step takes $O(\log n)$ time and $O(n/h)$ work.

The correctness of the above algorithm follows from that of the algorithm for the basic case and from Lemma 1. The algorithm obviously takes altogether $O(\log n)$ time and $O((n/h)\log h)$ work on the EREW PRAM.

Finally, we should point out that it is an easy matter to reduce the maximum size of the resulted ordered subsets of the partition by a chosen constant factor. For example, by letting $h' = h/c$ for some constant $c > 1$ and using h' instead of h in our partition algorithm, the maximum size of the resulted ordered subsets is reduced by a constant fraction depending on c, in the same asymptotic complexity bounds.

We now summarize the results of this section in the following theorem.

Theorem 1 *Given an n-element set S that is organized as k sorted columns of size n/k each and given a parameter h with $1/k \leq h \leq n/k$, suppose the following partitioning on S is to be done: Partition S into $g = O(n/(hk))$ subsets $D_1, D_2, \ldots, D_g$ of size $\Theta(hk)$ each, such that for any two indices i and j with $1 \leq i < j \leq g$, no element in D_i is bigger than any element in D_j. Such a partition of S can be obtained in $O(\log n)$ time and $O(\min\{(n/h) * \max\{\log h, 1\}, n * \max\{\log(1/h), 1\}\})$ work on the EREW PRAM.*

References

[1] M. Ajtai, J. Komlos, and E. Szemeredi. "Sorting in $c \log n$ parallel steps," *Combinatorica*, 3 (1983), pp. 1–19.

[2] M.J. Atallah, R. Cole, and M.T. Goodrich. "Cascading divide-and-conquer: A technique for designing parallel algorithms," *SIAM J. Computing*, 18 (3) (1989), pp. 499–532.

[3] G. Bilardi and A. Nicolau. "Adaptive bitonic sorting: An optimal parallel algorithm for shared-memory machines," *SIAM J. Computing*, 18 (1989), pp. 216–228.

[4] M. Blum, R.W. Floyd, V.R. Pratt, R.L. Rivest, and R.E. Tarjan. "Time bounds for selection," *J. of Computer and System Sciences*, 7 (4) (1972), pp. 448–461.

[5] R.P. Brent. "The parallel evaluation of general arithmetic expressions," *J. of the ACM*, 21 (1974), pp. 201–206.

[6] D.Z. Chen. "Efficient parallel binary search on sorted arrays, with applications," *IEEE Trans. on Parallel and Distributed Systems*, 6 (4) (1995), pp. 440–445.

[7] W. Chen, K. Wada, and K. Kawaguchi. "A parallel method for finding the convex hull of discs," *IEEE 1st International Conf. on Algorithms and Architectures for Parallel Processing*, 1995, pp. 274–281.

[8] R. Cole. "An optimally efficient selection algorithm," *Information Processing Letters*, 26 (1987/1988), pp. 295–299.

[9] R. Cole. "Parallel merge sort," *SIAM J. Computing*, 17 (1988), pp. 770–785.

[10] R. Cole and U. Vishkin. "Deterministic coin tossing and accelerating cascades: Micro and macro techniques for designing parallel algorithms," *Proc. 18th Annual ACM Symp. Theory of Computing*, 1986, pp. 206–219.

[11] G.N. Frederickson and D.B. Johnson. "The complexity of selection and ranking in $X + Y$ and matrices with sorted columns," *J. of Computer and System Sciences*, 24 (2) (1982), pp. 197–208.

[12] T. Hagerup and C. Rub. "Optimal merging and sorting on the EREW PRAM," *Information Processing Letters*, 33 (1989), pp. 181–185.

[13] J. JáJá. *An Introduction to Parallel Algorithms*, Addison-Wesley, Reading, Massachusetts, 1992.

[14] D.B. Johnson and T. Mizoguchi. "Selecting the Kth element in $X+Y$ and $X_1+X_2+\cdots+X_m$, *SIAM J. Computing*, 7 (1978), pp. 147–153.

[15] R.M. Karp and V. Ramachandran. "Parallel algorithms for shared-memory machines," *Handbook of Theoretical Computer Science*, J. van Leeuwen (eds.), Vol. 1, Elsevier Science Publishers, 1990.

[16] D.E. Knuth. *The Art of Computer Programming, Vol. 1, Fundamental Algorithms*, Second Edition, Addison-Wesley, Reading, Massachusetts, 1973.

[17] C.P. Kruskal. "Searching, merging and sorting in parallel computation," *IEEE Trans. on Computers*, C-32 (10) (1983), pp. 942–946.

[18] C.P. Kruskal, L. Rudolph, and M. Snir. "The power of parallel prefix," *IEEE Trans. on Computers*, C-34 (10) (1985), pp. 965–968.

[19] R.E. Ladner and M.J. Fischer. "Parallel prefix computation," *J. of the ACM*, 27 (1980), pp. 831–838.

[20] T.H. Merrett. *Relational Information Systems*, AFIPS Press, Reston, Virginia, 1984.

[21] K. Nakano and S. Olariu. Private communication (1996).

[22] S. Olariu and Z. Wen. "An efficient parallel algorithm for multiselection," *Parallel Computing*, 17 (1991), pp. 689–693.

[23] S. Olariu and J.L. Schwing. "A faster sorting algorithm in the broadcast communication model," *Proc. 9th International Parallel Processing Symp.*, 1995, pp. 319–323.

[24] J.H. Reif. "An optimal parallel algorithm of integer sorting," *Proc. 26th IEEE Annual Symp. Foundations of Computer Science*, 1985, pp. 496–504.

[25] G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*, Addison-Wesley, Reading, Massachusetts, 1988.

[26] Z. Wen. "Multi-way merging in parallel," to appear in *IEEE Trans. on Parallel and Distributed Systems.*