Efficient Weighted Multiselection in Parallel Architectures*

Hong Shen
Graduate School of Information Science
Japan Advanced Institute of Science and Technology
1-1 Asahidai, Tatsunokuchi, Ishikawa 923-1292, Japan
Email: shen@jaist.ac.jp

Abstract

We study parallel solutions to the problem of weighted multiselection to select r elements on given weighted-ranks from a set S of n weighted elements, where an element is on weighted rank k if it is the smallest element such that the aggregated weight of all elements not greater than it in S is not smaller than k. We propose efficient algorithms on two of the most popular parallel architectures, hypercube and mesh. For a hypercube with $p < n$ processors, we present a parallel algorithm running in $O(n^\epsilon \min\{r, \log p\})$ time for $p = n^{1-\epsilon}$, $0 < \epsilon < 1$, which is cost optimal when $r \geq p$. Our algorithm on $\sqrt{p} \times \sqrt{p}$ mesh runs in $O(\sqrt{p} + \frac{n}{p} \log^3 p)$ time which is the same as multiselection on mesh when $r \geq \log p$, and thus has the same optimality as multiselection in this case.

Key words: Hypercube, mesh, multiselection, parallel algorithm, weighted selection.

1 Introduction

Given an unordered set $S = \{x_1, x_2, \ldots, x_n\}$ and a weighted-rank array of r integers $K = \{k_1, k_2, \ldots, k_r\}$, where element x_i is associated with a weight $w(x_i)$, the problem of *weighted multiselection* requires to select r elements from S, $x_1^*, \ldots, x_r^*$, such that x_i^* is the k_i-th weighted element in S for $i = 1, \ldots, r$. Here, x is said to be the k-th weighted element in S if it is the smallest element such that the aggregated weight of all elements not greater than it in S is not smaller than k: $x = \min\{x_i \mid \sum_{x_j \leq x_i} w(x_j) \geq k\}$. Weighted multiselection generalizes the problems of weighted selection (when $r = 1$) and multiselection (when $w(x_i) = 1$ for all $1 \leq i \leq n$), both having been studied extensively in sequential [2, 10, 11, 16, 18, 26] and parallel [1, 4, 6, 7, 15, 24, 25, 19] environments due to their important applications in various fields such as graph theory, order statistics and set theory [9, 12, 5, 17].

Recently it has been shown that weighted multiselection can be done optimally both sequentially and in parallel on PRAM, in the same order as for multiselection. This is consistent with the complexity relationship between the conventional selection and weighted selection. For parallel weighted multiselection, a cost (product of time and number processors) optimal algorithm following the general paradigm to achieve cost optimum has been proposed in [23], which runs in $O(n^\epsilon \log r)$ time on the EREW (Exclusive-Read and Exclusive-Write) PRAM for any $0 < \epsilon < 1$.

When the computation model is an interconnection network, several results are known for multiselection. For a hypercube containing $p < n$ processors, it has been shown [20] that multiselection can be done in $O(n^\epsilon \min\{r, \log p\})$ time with $p = n^{1-\epsilon}$ synchronous processors, for any $0 < \epsilon < 1$, which is cost optimal when $r \geq p$. For a $\sqrt{p} \times \sqrt{p}$ mesh-connected computer, where all processors operate synchronously, multiselection can be solved optimally in time $(p^{1/2} + \min\{r \log r, \log p\} \frac{n}{p} \log^2 p)$ [21].

In this paper we present two efficient parallel algorithms for weighted multiselection in hypercube and mesh respectively. They have the same time complexity as their corresponding multiselection algorithms when $r \geq \log p$. The rest of the paper is organized as follows: In the next section, we present our hypercube algorithm for weighted multiselection. Section 3 describes the weighted multiselection algorithm on mesh. Section 4 concludes the paper.

2 Weighted multiselection on hypercube

We use a hypercube with $p \leq n$ processors. Processors in the hypercube operate synchronously (in SIMD mode). We denote by $cube(a, d)$ a d-dimensional hypercube with processor starting address a (and ending address $a + 2^d - 1$). A

*This work was supported by research grant #14380139 funded by Japan Society for the Promotion of Science (JSPS) under its Grant-in-Aid for Scientific Research Category **(B)**.

d-dimensional hypercube $cube(0, d)$ contains two $(d-1)$-dimensional subcubes $cube(0, d-1)$ and $cube(2^{d-1}, d-1)$ across which communication links connect pairs of processors whose addresses differ at precisely the d-th bit (the most significant bit).

For simplicity and without loss of generality, throughout this paper we assume that all elements in the rank array K are sorted in increasing order. If this is not the case, we can make it by simply calling a hypercube sorting algorithm with an additional cost of $O(r \log r \log\log r)$. For set $S = \{x_1, x_2, \ldots, x_n\}$, where x_i has weight $w(x_i)$ associated, we use $w(S)$ to denote the aggregated weight of all elements in S: $w(S) = \sum_{i=1}^{n} w(x_i)$. Given rank array K, to facilitate description we say selecting K from S to mean selecting those elements from S whose weighted-ranks are specified in K.

We first present the algorithm for (single-element) weighted selection on hypercube and then describe our algorithm for weighted multiselection.

2.1 Weighted selection

Let SelectCube(S, k, $cube(a, d)$) be an algorithm for selecting the k-th smallest element from S in $cube(a, d)$. Here we use the following algorithm given in [22] that combines Chandran and Rosenfelt's hypercube selection algorithm [3] and Cypher and Plexiton's $O(\log p \log\log p)$-time sorting algorithm [8].

Algorithm SelectCube(S, k, $cube(a, d)$)
{* Select the kth element from n elements in S in $cube(a, d)$ with p processors. *}

1. **for** $i = 0$ **to** $p-1$ **do in parallel**
   Processor i selects the median m_i of its local n/p data using an optimal sequential selection algorithm;
2. Select the median m of M using the parallel sorting algorithm of [8];
   {*M contains all local medians m_i, $0 \le i \le p-1$.*}
3. Split S into $S_L = \{x \in S \mid x < m\}$, $S_E = \{x \in S \mid x = m\}$ and $S_G = \{x \in S \mid x > m\}$;
4. **if** $|S_L| < k \le |S_L| + |S_E|$ **then** Output any element in S_E; EXIT;
5. **if** $k \le |S_L|$ **then** $S' = S_L$; $k' = k$ **else** $S' = S_G$; $k' = k - |S_L| - |S_E|$;
6. Distribute S' evenly over all processors in $cube(a, d)$;
7. SelectCube(S', k', $cube(a, d)$)

endSelectCube.

The time complexity of SelectCube, denoted by $T_S(n, p)$, is

$$T_S(n, p) = O(\log p \log\log p \log(n/p) + n/p). \quad (1)$$

Derivation of the above equation can be found in [22].

Our algorithm for weighted selection in hypercube can be constructed by applying SelectCube and is based on the following idea: Select the median of S and use it to split S into two halves; identify the half that contains the k-th weighted element by comparing the aggregated weight of each set with k and discard the other half of data; redistribute data and repeat this process. The algorithm is described as follows:

Algorithm WSelectCube(S, W, k, $cube(a, d)$)
{* Select the k-th weighted element from n weighted elements in S in $cube(a, d)$ with p processors. *}

1. **if** $d \le 1$ **then** employ an optimal sequential weighted selection algorithm [22];
2. **if** $p = n$ **then**
   Sort S using the algorithm of [8];
   Compute prefix sums of the sorted sequence of S; Search for k among the prefix sums;
   {*If $s_{i-1} < k \le s_i$, where s_i is the prefix sum up to x'_i and $x'_1 \le x'_2 \le \ldots \le x'_n$, the k-th weighted element is x'_i.*}
3. SelectCube(S, $n/2$, $cube(a, d)$);
4. Compute $w(S_L) = \sum_{x \in S_L} w(x)$;
5. **if** $k < w(S_L)$ **then** $S' = S_L$; $W' = W_L$
   else $S' = S_G$; $W' = W_G$; $k = k - w(S_L)$;
6. **if** $n/p > 1$ **then** $d' = d$ **else** $d' = d - 1$;
7. Redistribute S' in $cube(a, d')$;
8. WSelectCube(S', W', k, $cube(a, d')$)

endWSelectCube.

Let $T_{WS}(n, p)$ denote the time complexity of the algorithm. T_{WS} can be obtained by the following analysis: Step 1 takes $O(n)$ time; Step 2 requires $O(\log p \log\log p)$ time for sorting ($O(\log p)$ time for prefix summation and searching); Step 3 requires $O(\log p \log\log p \log(n/p) + n/p)$ time by Equation 1; Steps 4–7 need $O(\log p + n/p)$ time; Step 8 requires $T_{WS}(n/2, p)$ time. We have therefore the following equation:

$$T_{WS}(n,p) = \begin{cases} O(n), \text{ if } d \le 1, \\ O(\log p \log\log p), \text{ if } p = n, \\ O(\log p \log\log p \log(n/p) + n/p) \\ +T_{WS}(n/2,p) \text{ if } n/p > 1. \end{cases} \quad (2)$$

Solution to the above equation is

$$T_{WS}(n,p) = O(\sum_{i=0}^{\log(n/p)-1} (\frac{n}{p2^i} + \log p \log\log p \log \frac{n}{p2^i})) = O(n/p + \log^2(n/p) \log p \log\log p) \quad (3)$$

We therefore have the following lemma:

Lemma 1 *Weighted selection in n elements on a p-processor hypercube computer can be completed in $O(n/p + \log^2(n/p) \log p \log\log p)$ time.*

2.2 Weighted multiselection

The structure of hypercube lends itself well for parallel execution of balanced divide-and-conquer algorithms. This leads to the following idea for our weighted multiselection: first we select the median m from S and use it as the splitter to partition S into two halves S_L and S_G containing all elements not greater than and not smaller than m respectively; we compute $w(S_L)$ and $w(S_G)$, and use them to partition K into two parts $K_L = \{k \mid k < w(S_L)\}$ and $K_G = \{k \mid k \ge w(S_L)\}$ accordingly; then we recursively select K_L from S_L and K_G from S_G respectively in parallel. Following this idea, our algorithm for weighted multiselection is a call to the following procedure with WMSelectCube(S, W, K, *cube*$(0, \log p)$).

Algorithm WMSelectCube(S, W, K, *cube*(a,d))
{*Select all elements in S whose weighted-ranks are given in K in a d-dimensional hypercube with processor starting address a, where $|S| = n$ and $|K| = r$.*}

1. **if** $d = 0$ **then** use an optimal sequential weight multiselection algorithm [23] and EXIT;
2. **if** $r = 1$ **then** WSelectCube(S, W, k_1, *cube*(a,d)) and EXIT;
3. SelectCube(S, $n/2$, *cube*(a,d)); {*Select the median of S on *cube*(a,d).*}
4. **if** $p = n$ **then**
   Sort S using the algorithm of [8];
   Compute prefix sums of the sorted sequence of S; Search for k_i in parallel among the prefix sums;
5. Partition S into two halves: $S_L = \{x \mid x \le m\}$ and $S_G = \{x \mid x \ge m\}$;
6. Compute $w(S_L) = \sum_{x \in S_L} w(x)$;
7. Partition K into two parts: $K_L = \{k \mid k < w(S_L)\}$ and $K_G = \{k \mid k \ge w(S_L)\}$;
8. **if** $|K_L| = 0$ **then**
   Redistribute S_G in *cube*(a,d);
   WMSelectCube(S_G, K_G, *cube*(a,d))
   else if $|K_G| = 0$ **then**
   Redistribute S_L in *cube*(a,d);
   WMSelectCube(S_L, K_L, *cube*(a,d))
   else
   Redistribute S_L in *cube*$(a,d-1)$ and S_G in *cube*$(a+2^{d-1}, d-1)$ respectively;
   Do in parallel
   WMSelectCube(S_L, K_L, *cube*$(a,d-1)$);
   WMSelectCube(S_G, K_G, *cube*$(a+2^{d-1}, d-1)$)

endWMSelectCube.

Let $T(n,r,p)$ denote the time complexity of Algorithm WMSelectCube. Step 1 takes $O(n)$ time. Step 2 requires $O(n/p + \log^2(n/p) \log p \log\log p)$ time by Algorithm WSelectCube. Step 3 requires $O(\log p \log\log p \log(n/p) + n/p)$ time by Equation 1; Step 4 requires $O(\log p \log\log p)$ time; Steps 5–7 need $O(\log p + n/p)$ time; In Step 8, moving data to a subcube is realized by standard techniques for data aggregation and reduction [13] in $O(\log p + n/p)$ time, the recursive calls take $T(n/2, \max\{|K_L|, |K_G|\}, p)$ (for the first two) and $T(n/2, \max\{|K_L|, |K_G|\}, p/2)$ (for the last pair of parallel calls). As the three recursive calls are selective, the maximum time for them is $T(n/2, \max\{|K_L|, |K_G|\}, p/2)$. We have therefore the following equation:

$$T(n,r,p) = \begin{cases} O(n/p + \log^2(n/p) \log p \log\log p), \\ \qquad \text{if } r = 1, \\ O(n \log r), \text{ if } p = 1, \\ O(\log p \log\log p), \text{ if } p = n, \\ O(\log p \log\log p \log(n/p) + n/p) + \\ T(n/2, \max\{|K_L|, |K_G|\}, p/2). \\ \qquad \text{if } r, n/p > 1. \end{cases} \quad (4)$$

Noticing that $\max\{T(n/2, |K_L|, p/2), T(n/2, |K_G|, p/2)\} = T(n/2, \max\{|K_L|, |K_G|\}, p/2) \le T(n/2, r-1, p/2)$, we have the following solution to the above recurrence:

$$T(n,r,p) = O((\log p \log\log p \log(n/p) + n/p) \min\{r, \log p\}). \quad (5)$$

Since *cube*$(a, d-1)$ and *cube*$(a+2^{d-1}, d-1)$ are topologically identical two halves ($(d-1)$-D

hypercubes) of $cube(a, d)$, two parallel recursive calls WMSelectCube(S_L, K_L, $cube(a, d-1)$) and WMSelectCube(S_G K_G, $cube(a+2^{d-1}, d-1)$) can run easily in the synchronous mode. All other steps in the algorithm are executed also in the synchronous mode.

Clearly, if we let $p = n^{1-\epsilon}$ for any $0 < \epsilon < 1$, there is always a constant n^* s.t. $n/p = n^\epsilon > \log p \log\log p$ holds for all $n > n^*$. This shows that

$$T(n, r, p) = O(n^\epsilon \min\{r, \log p\}), \text{ where } p = n^{1-\epsilon}. \quad (6)$$

Therefore we have the following theorem:

Theorem 1 *Selecting r elements on given weighted-ranks in an arbitrary set of n elements can be completed in a hypercube with $p = n^{1-\epsilon}$ processors in $O(n^\epsilon \min\{r, \log p\})$ time for any $0 < \epsilon < 1$.*

The above result shows that weighted multiselection on hypercube has the same time complexity and hence same optimality as multiselection.

3 Weighted multiselection on mesh

We now consider the problem of weighted multiselection on a $\sqrt{p} \times \sqrt{p}$ mesh-connected computer. As for hypercube algorithm, we first give an algorithm for single-element weighted selection, and then present our algorithm for weighted multiselection.

3.1 Weighted selection

For single-element selection SelectMesh, we use the algorithm given in [21], which requires time $O(\sqrt{p} + \frac{n}{p} \log p \log(kp/n))$.

Applying SelectMesh, we can obtain a single-element weighted selection algorithm, WSelectMesh, as follows.

Algorithm WSelectMesh (S, k, p, W)
{*Select the element on weighted-rank k from S on a $\sqrt{p} \times \sqrt{p}$ mesh.*}

1. **if** $p \leq 3$ **then**
   Use a sequential weighted selection algorithm; EXIT;
2. SelectMesh (S, $in/4$, p, x_i) for $i = 1, 2, 3$;
   {*Select the $\frac{n}{4}$, $\frac{n}{2}$th and $\frac{3n}{4}$ elements x_1, x_2 and x_3 in S.*}
3. Partition S into four subsets:
   $S_{SW} = \{x \mid x \leq x_1\}$;
   $S_{SE} = \{x \mid x_1 \leq x \leq x_2\}$;
   $S_{NW} = \{x \mid x_2 \leq x \leq x_3\}$;
   $S_{NE} = \{x \mid x \geq x_3\}$;
4. Compute $w(S_{SW}) = \sum_{x \in S_{SW}} w(x)$, $w(S_{SE}) = \sum_{x \in S_{SE}} w(x)$ and $w(S_{NW}) = \sum_{x \in S_{NW}} w(x)$;
5. Set S' to be
   S_{SW} if $k \leq w(S_{SW})$,
   S_{SE} if $w(S_{SW}) < k \leq w(S_{SW}) + w(S_{SE})$,
   S_{NW} if $w(S_{SW}) + w(S_{SE}) < k \leq w(S_{SW}) + w(S_{SE}) + w(S_{NW})$, and
   S_{NE} if $w(S_{SW}) + w(S_{SE}) + w(S_{NW}) < k \leq w(S_{SW}) + w(S_{SE}) + w(S_{NW}) + w(S_{NE})$;
   Let W' be the corresponding weight matrix of all elements in S';
6. Distribute S' evenly among all processors in the mesh;
7. WSelectMesh (S', k, p, W')

endWSelectMesh.

Let the time complexity of algorithm WSelectMesh be $T_{WS}(n, p)$. Step 1 requires $O(n)$ time [23] and Step 2 $O(\sqrt{p} + \frac{n}{p} \log^2 p)$ time by algorithm SelectMesh [21]. Steps 3 and 4 require broadcast of x_i, scanning (splitting) local data set on each processor and reduction (summation), which takes $O(\sqrt{p} + n/p)$ time. Step 5 requires $O(1)$ time and Step 6 $O(\sqrt{p})$ time by using standard techniques. Step 7 requires $T_{WS}(\frac{n}{4}, \frac{p}{4})$ time. In summary, we have

$$\begin{aligned} T_{WS}(n, p) &= O(\sqrt{p} + \frac{n}{p} \log^2 p) + T_{WS}(\frac{n}{4}, \frac{p}{4}) \\ &= O(\sum_{i=0} \log_4 p - 1 \frac{\sqrt{p}}{2^i} + \frac{n}{p} \log^2 \frac{p}{4^i}) \\ &= O(\sqrt{p} + \frac{n}{p} \log^3 p). \end{aligned} \quad (7)$$

Therefore, the following lemma holds:

Lemma 2 *Weighted selection in n elements on a $\sqrt{p} \times \sqrt{p}$ mesh-connected computer can be completed in $O(\sqrt{p} + \frac{n}{p} \log^3 p)$ time.*

3.2 Weighted multiselection

The basic idea of our algorithm for weighted multiselection selecting K from S on mesh is to repeatedly break S (or K) into equal-sized subsets and K (or S) into some subsets accordingly, and then do weighted multiselection on each corresponding pair of subsets on a submesh. In order to carry out weighted multiselection on each submesh synchronously in parallel, we should ensure that all submeshes are topologically identical. For this purpose, each phase we partition the $\sqrt{p} \times \sqrt{p}$ mesh into 4 submeshes of size

$\frac{\sqrt{p}}{2} \times \frac{\sqrt{p}}{2}$. We use two different strategies to partition S and K, depending on the size of K. When $r > \log p / \log\log p$, we select 3 elements at ranks $\frac{n}{4}$, $\frac{n}{2}$ and $\frac{3n}{4}$ in S respectively and partition S and K each into 4 subsets and move them to 4 submeshes of size $\frac{\sqrt{p}}{2} \times \frac{\sqrt{p}}{2}$ in the mesh respectively for next phase process. When $r \leq \log p / \log\log p$, we use $k_{r/4}$, $k_{r/2}$ and $k_{3/4}$ as ranks to select 3 elements in S and partition S and K each into 4 subsets. Below is the sketch of our algorithm.

Algorithm WMSelectMesh (S, K, p, M)
{*Select all elements in S whose ranks are specified in K on a $\sqrt{p} \times \sqrt{p}$ mesh M, where $|S| = n$ and $|K| = r$.*}

1. **if** $r \leq 3$ **then**
   WSelectMesh (S, k_i, p, x_i) for $i = 1, 2, 3$;
   exit;

2. **if** $p \leq 3$ **then**
   Select K using a sequential algorithm;
   exit;

3. **if** $r > \log p / \log\log p$ **then** $k'_i \leftarrow \frac{in}{4}$
   for $i = 1, 2, 3$
   else $k'_i \leftarrow k_{ir/4}$ for $i = 1, 2, 3$;
   SelectMesh (S, k'_i, p, x_i) for $i = 1, 2, 3$;
   {*Select the $\frac{n}{4}$, $\frac{n}{2}$th and $\frac{3n}{4}$ elements x_1, x_2 and x_3 in S.*}

4. Partition S into four subsets:
   $S_{SW} = \{x \mid x \leq x_1\}$;
   $S_{SE} = \{x \mid x_1 \leq x \leq x_2\}$;
   $S_{NW} = \{x \mid x_2 \leq x \leq x_3\}$;
   $S_{NE} = \{x \mid x \geq x_3\}$;

5. Compute $w(S_{SW}) = \sum_{x \in S_{SW}} w(x)$, $w(S_{SE}) = \sum_{x \in S_{SE}} w(x)$ and $w(S_{NW}) = \sum_{x \in S_{NW}} w(x)$;

6. Partition K into four subsets:
   $K_{SW} = \{k \mid k < w(S_{SW})\}$;
   $K_{SE} = \{k \mid w(S_{SW}) \leq k < w(S_{SW}) + w(S_{SE})\}$;
   $K_{NW} = \{k \mid w(S_{SW}) + w(S_{SE}) \leq k < w(S_{SW}) + w(S_{SE}) + w(S_{NW})\}$;
   $K_{NE} = \{k \mid k \geq w(S_{SW}) + w(S_{SE}) + w(S_{NW})\}$;

7. Move S_i to $\frac{\sqrt{p}}{2} \times \frac{\sqrt{p}}{2}$ submesh M_i of M for $i = SW, SE, NW, NE$;
   {*M_{SW}, M_{SE}, M_{NW} and M_{NE} are respectively the south-west, south-east, north-west and north-east quadruples of M.*}

8. Balance load within M_i for $i = SW, SE, NW, NE$ in parallel.
   {*Each processor holds n/p data after load balancing.*}

9. Do in parallel for $i = SW, SE, NW, NE$
   if $|K_i| \neq 0$ **then**
   WMSelectMesh (S_i, K_i, $p/4$, M_i).
   {*Select K_i in S_i on submesh M_i in parallel.*}

endWMSelectMesh.

In the above equation, the first item is routing cost and the second item is the cost for data comparisons.

We now proceed with time complexity analysis for algorithm WMSelectMesh.

- Steps 1 requires $O(\sqrt{p} + \frac{n}{p}\log^3 p)$ time.
- Step 2 can be done in $O(n \log r)$ time with the optimal sequential algorithm [9].
- Step 3 requires $O(\sqrt{p} + \frac{n}{p}\log^2 p)$ by algorithm SelectMesh [21].
- In Step 4, to partition S each processor in M needs to scan through its block of n/p data of S. This can be done in parallel for all processors and hence requires $O(n/p)$ time.
- Step 5 can be completed in $O(\sqrt{p} + \frac{n}{p})$ time.
- In Step 6, partitioning K at a processor requires $O(\log r)$ time as K is sorted. So in total $O(n/p + \log r)$ time is required.
- Step 7 can be completed by 4 phases of permutation routing [21] as follows:
  - Processors in the bottom half ($M_{SW} \cup M_{SE}$) of M send their S_{NW} to the corresponding processors and append them to the same partition in the top half ($M_{NW} \cup M_{NE}$), and those in the top half send their S_{SE} and append to the corresponding ones in the bottom half;
  - Processors in the left half ($M_{SW} \cup M_{NW}$) send their S_{NE} to the corresponding processors and append them to the same partition in the right half ($M_{SE} \cup M_{NE}$), and those in the right half send their S_{SW} and append to the corresponding ones in the left half;
  - Processors in M_{SE} send their S_{NE} and append to the corresponding processors in M_{NE}, and those in M_{NW} send their S_{SW} and append to the corresponding ones in M_{SW};
  - Processors in M_{SW} send their S_{SE} and append to the corresponding processors in M_{SE}, and those in M_{NE} send their S_{NW} and append to the corresponding ones in M_{NW}.

As each permutation routing in M requires in $2\sqrt{p}-2$ steps [14], the above task can be completed in $O(\sqrt{p})$ time.

- Using similar approach as above ("folding" and "unfolding"), Step 8 load balancing within each M_i can be completed in $O(\sqrt{p})$ time.

Let $T(n,r,p)$ be the time complexity of the algorithm. It is easy to see that the recursive call in Step 9 has a time complexity of at most $T(n-3r/4, r/4, p/4)$ when $r > \log p/\log\log p$, and $T(n/4, r, p/4)$ when $r \leq \log p/\log\log p$. This results in the following equation

$$T(n,r,p) = \begin{cases} O(\sqrt{p} + \frac{n}{p}\log^3 p)), \text{ if } r \leq 3, \\ O(n\log r), \text{ if } p \leq 3, \\ O(\sqrt{p} + \frac{n}{p}\log^2 p) + \min\{T(n-3r/4, \\ r/4, p/4), T(n/4, r, p/4)\}, \text{ if } r,p > 3. \end{cases} \quad (8)$$

The solution to the above recurrence is

$$T(n,r,p) = O(\sqrt{p} + \frac{n}{p}\log^3 p). \quad (9)$$

We have therefore the following theorem regarding weighted multiselection on mesh:

Theorem 2 *Selecting r elements on specified weighted-ranks in an arbitrary set of n elements can be completed on mesh-connected computer with p processors operating synchronously in $O(\sqrt{p} + \frac{n}{p}\log^3 p)$ time, for any $p \leq n$.*

In comparison with the multiselection algorithm of [21], our weighted multiselection algorithm has the same time complexity as multiselection when $r \geq \log p$. In this case, the weighted multiselection algorithm is optimal.

4 Concluding remarks

We have proposed two efficient parallel algorithms for weighted multiselection in hypercube and mesh respectively. For selecting r elements on specified weighted-ranks in a given set of n elements in a hypercube with p synchronous processors, $p < n$, our algorithm requires time $O(n^\epsilon \min\{r, \log p\})$ when $p = n^{1-\epsilon}$, $0 < \epsilon < 1$, which is the same as multiselection on hypercube and is cost optimal when $r \geq p$.

For weighted multiselection on a $\sqrt{p} \times \sqrt{p}$ mesh, our algorithm runs in $O(\sqrt{p} + \frac{n}{p}\log^3 p)$ time for any $p \leq n$, where processors in the mesh operate synchronously. When $r \geq \log p$, our algorithm has the same time complexity and optimality as multiselection.

References

[1] S. G. Akl. An optimal algorithm for parallel selection. *Information Processing Letters*, 19, 1984.

[2] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*, 7:448–461, 1972.

[3] S. Chandran and A. Rosenfeld. Order statistics on a hypercube. *Information Processing Letters*, 27:129–132, 1988.

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

[5] J. Chen. Studies on partial order production. Phd thesis, Department of Computer Science, Lund University, 1993.

[6] R. Cole and C. K. Yap. A parallel median algorithm. *Information processing Letters*, 20:137–139, 1985.

[7] R. J. Cole. An optimally efficient selection algorithm. *Information Processing Letters*, 26:295–299, 1988.

[8] R. Cypher and G. Plaxton. Deterministic sorting in nearly logarithmic time on the hypercube and related computers. In *Proc. 22nd ACM Symp. Theory of Computing*. ACM Press, 1990.

[9] M. L. Fredman and T. H. Spencer. Refined complexity analysis for heap operations. *Journal of Computer and System Sciences*, pages 269–284, 1987.

[10] F. Fussenegger and D. B. Johnson. A counting approach to lower bounds for selection problems. *J. Asso. Comput. Mach.*, 26:540–543, 1979.

[11] L. Hyafil. Bounds for selection. *SIAM J. Comput.*, 5:114–119, 1976.

[12] D. G. Kirkpatrick. A unified lower bound for selection and set partitioning problems. *J. Asso. Comput. Mach.*, 28:150–165, 1981.

[13] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *In troduction to Parallel Computing*. Benjamin/Cummings, 1993.

[14] M. Kunde. Routing and sorting on mesh-connected architectures. In *Proc. Agean Workshop on Computing: VLSI algorithms and architectures*, pages 423–433. Lecture Notes on Computer Science, 1988.

[15] C. G. Plaxton. On the network complexity of selection. Technical Report STAN//CS-TR-89-1276, Stanford University, Department of Computer Science, August 1989.

[16] V. R. Pratt and F. F. Yao. On lower bounds for computing the ith largest element. In *Proc. Annual Symp. Switching and Atomata Theory.* Iowa City, 1973.

[17] Eric Ruppert. Parallel algorithms for the k shortest paths and related problems, 1996.

[18] A. Schonhage, M. Paterson, and N. Pippenger. Finding the median. *J. Comput. Syst. Sci.*, 13:184–199, 1976.

[19] H. Shen. Improved universal k-selection in hypercubes. *Parallel Computing*, 18:177–184, 1992.

[20] H. Shen. Optimal parallel multiselection on EREW PRAM. *Parallel Computing*, 188:287–298, 1997.

[21] H. Shen. Efficient parallel algorithms for selection and multiselection on mesh-connected computers. In *Proc. 13th IEEE Int. Parallel Processing Sypm. and 10th IEE Sypm. On Parallel and Distributed Processing*, pages 426–430. IEEE, 1999.

[22] H. Shen. Efficient parallel multiselection in hypercubes. *Parallel Algorithms and Applications*, 14(3):217–227, 2000.

[23] H. Shen. Optimal parallel weighted multiselection. Technical report, Japan Advanced Institue of Science and Technology, 2002.

[24] H. Shen and G. L. Chen. Parallel selection using recursive filtering. *Chinese Journal of Computers*, 11:523–532, 1988.

[25] H. Shen and G. L. Chen. A new upper bound of delay time in selection network. *Chinese Journal of Computers*, 13:88–100, 1990.

[26] C. K. Yap. New bounds for selection. *Comm. ACM*, 19:501–508, 1976.