Efficient Parallel Algorithms for Selection and Multiselection on Mesh-Connected Computers* (Short Version)

Hong Shen†
School of Computing and Information Technology
Griffith University
Nathan, QLD 4111, Australia

1 Introduction

Let S be a set of n unordered elements, and K be an array of r integers, namely *ranks*, where $r \leq n$. The problem of *multiselection* requires to select the k_ith smallest (largest) element from S for $i = 1, \ldots, r$.

For sequential multiselection, Fredman et al. [4] established a tight lower bound of $\Omega(n \log r)$ time. This is consistent with the inherent complexity of multiselection that falls in between conventional selection $O(n)$ and sorting $O(n \log n)$. For parallel multiselection, efficient algorithms have been developed recently by Shen [10, 11] on PRAM and hypercube respectively, where the PRAM algorithm and MIMD hypercube algorithm are cost (product of time and number processors) optimal.

In this paper we present a set of efficient parallel algorithms for selection and multiselection on a $\sqrt{p} \times \sqrt{p}$ mesh-connected computer. Our main contributions include:

- a new algorithm for single-element selection for the general case when $p \leq n$ based on the generalized bitonic selection approach, which is more efficient than the previously known result for this problem when $p \geq n^{\frac{1}{2}+\epsilon}$ for any $\epsilon > 0$;
- an efficient algorithm for multiselection on SIMD mesh that runs in $O(p^{1/2} + \min\{r \log r, \log p\}\frac{n}{p} \log^2 p)$ time for selecting r elements from n given elements;
- an efficient algorithm for multiselection on an MIMD mesh that runs in $O(\log r(p^{1/2} + \frac{n}{p} \log^2 p))$ time and is time-optimal w.r.t. the current best result on single-element selection on mesh.

2 Preliminaries

A sequence consisting of two monotonic (sorted) sequences arranged in opposite orders, one ascending and the other descending, is called *bitonic sequence*. The following theorem was due to Batcher [1]:

*This work was partially supported by a research grant from the Australian Research Council.

†Part of the work was done when the author was visiting Dept. of Computer Science at Wayne State University.

Theorem 2.1 *If $a_0, a_1, \ldots, a_{m-1}$ is a bitonic sequence, sequences MIN and MAX are both bitonic and no element in MIN is greater than any element in MAX, where*

$$MIN : \min(a_0, a_{\frac{m}{2}}), \ldots, \min(a_{\frac{m}{2}-1}, a_{m-1});$$
$$MAX : \max(a_0, a_{\frac{m}{2}}), \ldots, \max(a_{\frac{m}{2}-1}, a_{m-1}).$$

We call the above operation to generate sequences MIN and MAX *Batcher's Compare-Exchange operation*, denoted by *BCE operation*.

Like merging and sorting [1, 12], parallel selection can also be done by performing a sequence of parallel comparisons [3], each of which carries out data comparisons over all pairs of processors whose addresses differ at precisely one bit position called *pivot*. Thus the sequence of parallel comparisons can be abstracted by a sequence of pivots, namely *pivot sequence*.

Theorem 2.2 *[3] For the problem of selecting k smallest elements from n elements in n processors, the pivot sequence for parallel comparisons is*

$$I_0, I_1, \ldots, I_{t-1}, t, I_{t-1}, t+1, I_{t-1}, \ldots, h-2, I_{t-1}, h-1, \quad (1)$$

where $k = 2^t$, $n = 2^h$, $I_i = i, i-1, \cdots, 0$ is the pivot sequence for merging a bitonic sequence of length 2^{i+1} $(0 \leq i \leq t-1)$.

The order setting for a comparison depends on the address of the processor that performs the comparison and the corresponding pivot's position in the pivot sequence [3].

3 Single-element selection

3.1 Selection when $p = n$

For selection on MCC when $p = n$, there have been several algorithms [2, 5, 7, 8] proposed. All of them require time $O(\sqrt{p})$, which is defined by the $(\sqrt{p}-1)$-step lower bound on communication steps for message passing between a pair of processors in the MCC. The algorithm developed in [2] based on the *bitonic selection network* [3] is the first selection algorithm on

MCC we know of. This network requires $O(\sqrt{p})$ communication steps and $O(\log(n/\sqrt{k})\log k)$ data comparisons. Its high regularity of data movements and simplicity in structure are desirable properties for development of efficient algorithms for selection in the more general case.

For simplicity and without loss of generality, assume that $p = 2^h$ and $k = 2^t$. For processor with index i, P_i, let $i_{h-1}i_{h-2}\cdots i_0$ be i's binary representation. We use $i^{(q)}$ to denote the index at pivot q to index i, that is, the resulting index after complementing bit i_q of i. Since $p = n$, selecting k smallest elements from n elements using p processors can be accomplished by a sequence of BCE operations over pairs of processors at pivots in the pivot sequence (1). This results in the following general paradigm for parallel selection, where default order setting is ascending.

Algorithm ParSelect (n, k)
{*Select the kth element from n element using $p = n$ processors.*}

1. **for** $i = 0$ **to** $p-1$ **do in parallel**
   for $j = 1$ **to** $t-1$ **do**
   if $i_{h-1} \oplus i_{h-2} \oplus \cdots \oplus i_{j+1} = 0$ **then**
   for $q = j$ **downto** 0 **do**
   BCE operation between P_i and $P_i^{(q)}$;

2. **for** $j = t$ **to** $h-2$ **do**
   Set order "ascending";
   for $i = 0$ **to** $p-1$ **do in parallel**
   BCE operation between P_i and $P_{i^{(j)}}$;
   if $i_{h-1} \oplus i_{h-2} \oplus \cdots \oplus i_{t+1} = 0$
   then set order "ascending"
   else set order "descending";
   for $q = j$ **downto** 0 **do**
   BCE operation between P_i and $P_{i^{(q)}}$;

3. **for** $i = 0$ **to** $p-1$ **do in parallel**
   BCE operation between P_i and $P_{i^{(h-1)}}$;

4. Compare the two elements at $P_{k/2-1}$ and $P_{k/2}$ and move the larger one to P_0.

When implementing the above paradigm on a real interconnection network, we have to consider the cost of routing data between P_i and $P_{i^{(q)}}$. The goal is to minimize this cost.

Lemma 3.1 *In shuffled row-major indexing the number of routing steps required for carrying out data comparisons at pivot j is $2^{\lfloor j/2 \rfloor}$.*

Proof
This is because for shuffled row-major indexing $i = i_{2q}i_q i_{2q-1}i_{q-1}\cdots i_{q+1}i_0$, processors $i^{(r)}$ and $i^{(r+q)}$ are equally-distanced from processor i in row and column directions respectively, where $0 \le j \le 2q$ and $0 \le r \le q$. □

By Lemma 3.1 we can easily obtain $O(\sqrt{p})$ routing steps, $O(\log(n/\sqrt{k})\log k)$ data comparisons, and thus the following time complexity of algorithm ParSelect implemented on an MCC:

$$T(n,k) = O(R+C) = O(\sqrt{p}+\log(n/\sqrt{k})\log m). \quad (2)$$

3.2 Selection when $p \le n$

For selection on MCC in the general case when $p \le n$, the only known result is [5] that requires $O(\min\{\{(\frac{n}{p}+p)\log(n/p), n/p^{2/3}+\sqrt{p}\})$ time. We will show in this section that, based on the result of [2], we can derive a more efficient algorithm for this general case than that of [5].

When $p \le n$, each processor in the MCC holds a block of n/p input data. In this case, for bitonic selection, each pair of processors at any pivot in the pivot sequence (1), instead of comparing two elements, need to compare two blocks of data and partition them into two halves, one containing all greater elements and the other containing all smaller ones. Furthermore, each processor involved in a routing step needs to send a packet of n/p data instead of a single element.

Let processor i hold a block B_i of n/p data, $\min(B_i, B_j)$ and $\max(B_i, B_j)$ stand for the smaller half and larger half of the elements in (B_i, B_j), $i < j$, respectively. Clearly they can be computed in $2^{\lfloor w/2 \rfloor+1}$ routing steps and $O(n/p)$ computation steps in an MCC with shuffled row-major indexing for $j = i^{(w)}$.

A sequence $\{B_0, B_1, \ldots, B_{m-1}\}$ is said *block-sorted* if no element in B_i is greater than any element in block B_{i+1} for $0 \le i < m-1$ [9]. A *block-bitonic* sequence consists of two block-monotonic sequences arranged in opposite orders.

Theorem 3.1 ([9]) *Let $B_0, B_1, \ldots, B_{m-1}$ be a block-bitonic sequence and*

$$\overline{MIN} = \{\min(B_0, B_{\frac{m}{2}}), \ldots, \min(B_{\frac{m}{2}-1}, B_{m-1})\};$$
$$\overline{MAX} = \{\max(B_0, B_{\frac{m}{2}}), \ldots, \max(B_{\frac{m}{2}-1}, B_{m-1})\}.$$

Sequences $\overline{MIN}$ and $\overline{MAX}$ are both block-bitonic and no element in $\overline{MIN}$ is greater than any element in $\overline{MAX}$.

We call the above operation to generate $\overline{MIN}$ and $\overline{MAX}$ *block-BCE operation*. Our parallel selection algorithm for the general case when $p \le n$ is presented as follows.

Algorithm SelectMesh (n, k, p, x)
{*Select the kth element from n elements using $p \le n$ processors and store it in x, where $k = 2^t$ and $p = 2^h$.*}

1. **if** $k \le n/p$ **then**
   for $i = 0$ **to** $p-1$ **do in parallel**
   Selects k smallest numbers in local data block of size n/p and discard the rest;
   {*The kth element must be within the collection of k smallest numbers of each local data block.*}

2. **if** $k > n/p$ **then**
   for $j = 1$ **to** $t - \log(n/p) - 1$ **do**
   if $i_{h-1} \oplus i_{h-2} \oplus \cdots \oplus i_{j+1} = 0$ **then**
   for $q = j$ **downto** 0 **do**
   Block-BCE operation between P_i and $P_{i(q)}$;
   {*All processors do $t - \log(n/p) - 1$ phases of block-merging on data blocks of length n/p to generate a series of block-sorted k-sequences, where all successive pairs form a series of block-bitonic $2k$-sequences.*}

3. **for** $j = \max\{t - \log(n/p), 0\}$ **to** $h - 1$
   Set order "ascending";
   for $i = 0$ **to** $p - 1$ **do in parallel**
   Block-BCE operation between P_i and $P_i^{(j)}$;
   $t' = \max\{t - \log(n/p), 0\}$;
   if $i_{h-1} \oplus i_{h-2} \oplus \cdots \oplus i_{t'+1} = 0$
   then set order "ascending"
   else set order "descending"
   if $(t > \log(n/p))$AND$(j < h - 1)$ **then**
   for $q = t - \log(n/p) - 1$ **downto** 0 **do**
   Block-BCE operation between P_i and $P_{i(q)}$;
   {*All processors repeatedly perform block-BCE operations to generate $\overline{MIN}$ and, when $k > n/p$ (i.e. $t > \log(n/p)$), block-merging to make $\overline{MIN}$s into block-sorted, until there is only one $\overline{MIN}$ sequence containing the k smallest numbers left.*}

4. Compare the two data block in $P_{\frac{kp}{2n}-1}$ and $P_{\frac{kp}{2n}}$ and select the maximum (store it at P_0 and x).

The following time complexity of the above algorithm which results in Theorem 3.2 can be easily verified:

$$T(n,k,p) = O(\frac{n}{p}\log p \log(kp/n)). \quad (3)$$

Theorem 3.2 *Selecting the kth element from n unordered elements on a $\sqrt{p} \times \sqrt{p}$ MCC for any $p \le n$ can be completed in $O(\sqrt{p} + \frac{n}{p}\log p \log(kp/n))$.*

We now compare the time complexity of our algorithm with the existing result $O(\min\{(\frac{n}{p} + p)\log(n/p), n/p^{2/3} + \sqrt{p}\})$ of [5]. Since $n/p^{2/3} = \Omega(\frac{n}{p}\log p \log(kp/n))$, and $p\log(n/p) = \Omega(\frac{n}{p}\log p \log(kp/n)$ when $p \ge n^{\frac{1}{2}+\epsilon}$ for any fixed constant $\epsilon > 0$, our time complexity is clearly better than that of [5] when p is not too small (satisfying the above condition).

Note that [5] gave only the number of routing steps which is $O(\min\{p\log(n/p), n/p^{2/3} + \sqrt{p}\}$. When taking into consideration of the computation cost defined mainly by the number of data comparisons, an additive factor of $O(\frac{n}{p}\log(n/p))$ should be included in the first component, which is trivial from the $\Omega(n)$ lower time bound for selection.

Combining our algorithm with that of [5], we may apply our algorithm when $p \ge n^{\frac{1}{2}+\epsilon}$ and the algorithm of [5] when $p < n^{\frac{1}{2}+\epsilon}$. This leads to the following result immediately:

Corollary 3.1 *Selecting the kth element from n unordered elements on a $\sqrt{p} \times \sqrt{p}$ MCC, $p \le n$, can be completed in $O(\min\{\sqrt{p} + \frac{n}{p}\log p \log(kp/n), (\frac{n}{p} + p)\log(n/p)\})$.*

4 Multiselection

We now consider the problem of selecting r elements at ranks specified in the rank array K from a set S of n elements on a $\sqrt{p} \times \sqrt{p}$ MCC. We call this problem selecting K from S. For simplicity and without loss of generality, we assume that $K = \{k_1, k_2, \ldots, k_r\}$ is sorted in increasing order. If this is not the case, we will sort it on the MCC at an additional $O(\sqrt{p})$ time.

4.1 SIMD algorithm

The basic idea of our algorithm for selecting K from S is to repeatedly break S (or K) into equal-sized subsets and K (or S) into some subsets accordingly, and then do multiselection on each corresponding pair of subsets on a submesh. In order to carry out 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 $r \le \log p/\log\log p$, 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 MSelectMesh1 (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** $r \le 3$ **then**
   SelectMesh (S, k_i, p, x_i) for $i = 1, 2, 3$; **exit**;

2. **if** $p \le 3$ **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 and K each into four subsets:
   $S_{SW} = \{x \mid x \le x_1\}$; $K_{SW} = \{k \mid k < n/4\}$;
   $S_{SE} = \{x \mid x_1 \le x \le x_2\}$;
   $K_{SE} = \{k \mid n/4 \le k < n/2\}$;
   $S_{NW} = \{x \mid x_2 \le x \le x_3\}$;
   $K_{NW} = \{k \mid n/2 \le k < 3n/4\}$;
   $S_{NE} = \{x \mid x \ge x_3\}$; $K_{NE} = \{k \mid k \ge 3n/4\}$;

5. 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.*}

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

7. Do in parallel for $i = SW, SE, NW, NE$
   if $|K_i| \neq \emptyset$ **then**
   MSelectMesh1 (S_i, K_i, $p/4$, M_i).
   {*Select K_i in S_i on submesh M_i in parallel.*}

The correctness of the algorithm can be seen easily from the comments interspersed with the the command lines in the algorithm. We now proceed with time complexity analysis.

- Steps 1 and 3 require $O(\sqrt{p} + \frac{n}{p} \log p \log(kp/n))$ by algorithm SelectMesh.
- Step 2 can be done in $O(n \log r)$ time with the optimal sequential algorithm [4].
- 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. Partitioning K at processor 0 requires $O(\log r)$ time as K is sorted. So in total $O(n/p + \log r)$ time is required.
- Step 5 can be completed by 4 phases of permutation routing among the 4 submeshes, each requiring in $2\sqrt{p} - 2$ steps [6].
- Using similar approach as above ("folding" and "unfolding"), Step 6 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 7 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 $r \leq \log p / \log\log p$. For single-element selection, taking into consideration of the worst-case of selecting a median ($k = n/2$) yields a time complexity of at most $O(\sqrt{p} + \frac{n}{p} \log^2 p)$ for SelectMesh. This results in the following equation

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

The solution to the above recurrence is

$$T(n,r,p) = O(\sqrt{p} + \min\{r \log r, \log p\} \frac{n}{p} \log^2 p). \quad (5)$$

We have therefore our first result on multiselection on MCC:

Theorem 4.1 *Selecting r elements at specified ranks in an arbitrary set of n elements can be completed on an SIMD mesh-connected computer with p processors in $O(\sqrt{p} + \min\{r \log r, \log p\} \frac{n}{p} \log^2 p)$ time, for any $p \leq n$.*

Clearly, for large n/p our algorithm is significantly faster than directly applying the single-element selection algorithm r times, which would require $O(r\sqrt{p} + \frac{rn}{p} \log^2 p)$ time, and the sorting algorithm on MCC that requires $O(n/\sqrt{p})$ time as each processor holds n/p elements.

If we use the algorithm of [5] to replace SelectMesh when $p \geq n^{\frac{1}{2}+\epsilon}$, the following corollary holds immediately:

Corollary 4.1 *Selecting r elements at specified ranks in an arbitrary set of n elements can be completed on an SIMD mesh-connected computer with $p \leq n$ processors in $O(\sqrt{p} + \min\{r \log r, \log p\} \min\{\frac{n}{p} \log^2 p, (\frac{n}{p} + p) \log(n/p)\})$ time, for any $p \leq n$.*

4.2 MIMD algorithm

Let $T_s(n, p)$ be the time complexity required by the known fastest algorithm for selecting a single element from n elements on an MCC of $p \leq n$ processors. We say a parallel multiselection algorithm on the MCC is optimal if it runs in $O(T_s(n, p) \log r)$ time [10].

We now empower the processors in the MCC from SIMD execution to MIMD execution, that is, they may execute different instructions at the same time. This lifts our previous requirement of topological identity on all submeshes in each phase of recursive partitioning, and allows us to partition the mesh more flexibly into any-shape submeshes. We show if we do so, we will be able to obtain an optimal algorithm.

Our strategy for multiselection on MIMD MCC M is as follows: we use the mid-element in K to partition S and K each into two subsets S_i and K_i, $i = 1, 2$, move them to a submesh M_i of size $|S_i|$ in M and then in parallel select K_i from S_i on M_i. Clearly the shape of M_i is arbitrary, depending on $|S_i|$. We consider the worst scenario that M_i is a row-major "strip" in M with side-size $|S_i|/c_i \times c_i$, where $1 \leq c_i \leq \sqrt{p} - 1$, since routing in this case is most expensive (in the same order as in M). Our algorithm is a call MSelectMesh2 (S, K, $\sqrt{p}$, $\sqrt{p}$, M) to the following procedure:

Algorithm MSelectMesh2 (S, K, u, v, M)
{*Select all elements in S whose ranks are specified in K on a $u \times v$ MIMD MCC M, where $|S| = n$, $|K| = r$, and $u \leq v$.*}

1. **if** $r = 1$ **then**
   SelectMesh (S, k_1, p, x); **exit**;
2. **if** $p = 1$ **then**
   Select K using a sequential algorithm; **exit**;
3. SelectMesh (S, $k_{r/2}$, p, x);
   {*Select the $k_{r/2}$th element in S.*}

4. Partition S and K each into two subsets;
$S_L = \{x \mid x \leq x\}$; $K_L = \{k \mid k < r/2\}$;
$S_G = \{x \mid x \geq x\}$; $K_G = \{k \mid k > r/2\}$;

5. Move S_i to $u_i \times v_i$ submesh M_i of M for $i = L, G$, where $u_i = \min\{\frac{|S_i|}{u}, u\}$ and $v_i = \max\{\frac{|S_i|}{u}, u\}$;
{*In case if $\frac{|S_i|}{u}$ is not an integer, round it up (and the other down).*}

6. Balance load within M_i for $i = L, G$ in parallel.
{*Each processor holds n/p data after load balancing.*}

7. Do in parallel for $i = L, G$
MSelectMesh1 (S_i, K_i, u_i, v_i, M_i).
{*Select K_i in S_i on $u_i \times v_i$ submesh M_i in parallel, where $u_i \leq v_i$.*}

Let $p = uv$. Since $|K_i| = |K|/2$ and all other steps except the recursion in Step 6 can be completed in $O(\sqrt{p} + \frac{n}{p}\log^2 p)$ time, the time complexity $T(n, r, p)$ of the algorithm can be easily expressed as the the following recurrence.

$$T(n,r,p) = \begin{cases} O(\sqrt{p} + \frac{n}{p}\log^2 p)), \text{ if } r \leq 3, \\ O(n \log r), \text{ if } p \leq 3, \\ O(v + \frac{n}{p}\log^2 p) + \max\{T(n_L, r/2, p_L), \\ T(n_G, r/2, p_G)\}), \text{where } n_L + n_G = \\ n - 1, p_L + p_G = p, r, p > 3. \end{cases} \quad (6)$$

Since $\max\{p_L, p_G\} \leq p$, $\max\{v_L, v_G\} \leq v \leq \sqrt{p}$ and $n_i/p_i = |S_i|/|M_i| = n/p$, clearly $\max\{T(n_L, r/2, p_L), T(n_G, r/2, p_G)\})\} \leq T(n, r/2, p)$. This simplifies the above recurrence to

$$T(n,r,p) = \begin{cases} O(\sqrt{p} + \frac{n}{p}\log^2 p)), \text{ if } r \leq 3, \\ O(n \log r), \text{ if } p \leq 3, \\ O(\sqrt{p} + \frac{n}{p}\log^2 p) + T(n, r/2, p))\}, \\ \text{if } r, p > 3. \end{cases} \quad (7)$$

The solution to the above recurrence is

$$T(n,r,p) = O(\log r(\sqrt{p} + \frac{n}{p}\log^2 p)). \quad (8)$$

Since the known best algorithm for selecting a single element from n elements in an MCC of p processors requires $O(\sqrt{p} + \frac{n}{p}\log^2 p)$ time when $p \geq n^{\frac{1}{2}+\epsilon}$, algorithm MSelectMesh2 is hence asymptotically optimal. We have therefore the following theorem:

Theorem 4.2 *Selecting r elements at specified ranks in an arbitrary set of n elements can be completed on an MIMD mesh-connected computer with p processors in $O(\log r(\sqrt{p} + \frac{n}{p}\log^2 p))$ time, for any $p \leq n$.*

If we use the algorithm of [5] instead of SelectMesh in the case when $p < n^{\frac{1}{2}+\epsilon}$, we will make MSelectMesh2 optimal across the whole range of p. So we have

Corollary 4.2 *There is an optimal algorithm for selecting r elements at specified ranks in an arbitrary set of n elements on an MIMD mesh-connected computer with $p \leq n$ processors that runs in $O(\log r(\sqrt{p} + \min\{\frac{n}{p}\log^2 p, (\frac{n}{p} + p)\log(n/p)\}))$ time.*

References

[1] K. E. Batcher. Sorting networks and their applications. In *Proc. AFIPS 1968 Spring Joint Computer Conference*, pages 307–314. AFIPS Press, 1968.

[2] G. L. Chen and H. Shen. Bitonic selection algorithm on simd machines. In *Proc. 2nd Intern. Conference on Computers and Applications*, pages 176–82. IEEE CS Press, 1987.

[3] G. L. Chen and H. Shen. Bitonic selection network and bitonic selection algorithm on multiprocessors. *Computer Studies and development*, 24:1–10, 1987.

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

[5] D. Krizanc and L. Narayanan. Multiple-packet selection on mesh-connected processor arrays. In *Proc. 1992 Intern. Parallel Processing Symp. (IPPS'92)*, pages 602–605. IEEE CS Press, 1992.

[6] 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.

[7] M. Kunde. l-selection and related problems on grids of processors. *J. of New Generation Computer Systems*, 2:129–143, 1989.

[8] C. Schnorr and A. Shamir. An optimal sorting algorithm for mesh-connected computers. In *Proc. 1996 Symp. Theory of Computing (STOC'86)*, pages 255–263. ACM, 1986.

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

[10] H. Shen. Optimal parallel multiselection on EREW pram. *Parallel Computing*, V. 23, p. 1987-92, 1997.

[11] H. Shen. Parallel multiselection in hypercubes", In *Proc. 1997 International Symposium on Parallel Architectures, Algorithms and Networks (I-SPAN'97)*, Taipei, Taiwan, 1997.

[12] H. Stone. Parallel processing with the perfect shuffle. *IEEE Trans. Comput.*, C-20(2):153–161, 1971.