Optimal Parallel Algorithms for Multiselection on Mesh-Connected Computers

Hong Shen[†][*] Yijie Han[‡] Yi Pan[*] David J. Evans[¶]

[†] Graduate School of Information Science
Japan Advanced Institute of Science and Technology
Tatsunokuchi, Ishikawa, 923-1292, Japan

[‡] Computer Science and Telecommunication Program
University of Missouri at Kansas City
Kansas City, MO 64110-2499, USA

[*] Department of Computer Science
Georgia State University
Atlanta, GA 30303, USA

[¶] Department of Computing
The Nottingham Trent University
Burton Street, Nottingham NG1 4BU, U.K.

Abstract

Multiselection is the problem of selecting multiple elements at specified ranks from a set of arbitrary elements. In this paper, we first present an efficient algorithm for single-element selection that runs in $O(\sqrt{p}+\frac{n}{p}\log p\log(kp/n))$ time for selecting the kth smallest element from n elements on a $\sqrt{p}\times\sqrt{p}$ mesh-connected computer of $p\leq n$ processors, where the first component is for communication and second is for computation (data comparisons). Our algorithm is more computationally efficient than the existing result when $p\geq n^{\frac{1}{2}+\epsilon}$ for any $0<\epsilon<\frac{1}{2}$. Combining our result for $p=\Omega(\sqrt{n})$ with the existing result for $p=O(\sqrt{n})$ yields an improved computation time complexity for the selection problem on mesh $t^{sel}_{comp}=O(\min\{\frac{n}{p}\log p\log(kp/n),(\frac{n}{p}+p)\log(n/p)\})$. Using this algorithm as a building block, we then present two efficient parallel algorithms for multiselection on the mesh-connected computers. For selecting r elements from a set of n elements on a $\sqrt{p}\times\sqrt{p}$ mesh, $p,r\leq n$, our first algorithm runs in time $O(p^{1/2}+t^{sel}_{comp}\min\{r\log r,\log p\})$ with processors operating in the SIMD mode, which is time-optimal when $p\leq r$. Allowing

[*]Questions regarding this paper should be sent to shen@@jaist.ac.jp

processors to operate in the MIMD mode, our second algorithm runs in $O(p^{1/2} + t^{sel}_{comp} \log r)$ time and is time-optimal for any r and p.

Key words: Computation time, mesh, multiselection, parallel algorithm, routing, selection.

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 $i = 1, \ldots, r$.

Multiselection generalizes two fundamental and important problems — conventional (single-element) selection and sorting: selection is a special case of multiselection when $r = 1$ and sorting is another special case when $r = n$. It has important applications in order statistics and set theory [8, 12]. While single-element selection has been studied extensively on both uni-processor environment [2, 9, 10, 17, 19, 26] and parallel computers [1, 3, 5, 4, 6, 7, 16, 18, 24, 21], not much has been done on multiselection. This is largely because multiselection can be solved directly by either repeatedly applying single-element selection or by sorting. It is our interest to find efficient solutions that have a better complexity than these direct approaches.

For sequential multiselection, Fredman et al. [8] 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 [23, 22] on PRAM and hypercube respectively, where the PRAM algorithm and MIMD hypercube algorithm are cost (product of time and number processors) optimal.

Let $t^{sel} = t^{sel}_{comm} + t^{sel}_{comp}$ be the best time complexity for single-element selection, where t^{sel}_{comm} and t^{sel}_{comp} are the time required for communication and computation respectively, on the underlying parallel computers concerned. The above results showed that multiselection can be done in $O(t^{sel} \log r)$ time on sequential RAM, PRAM and hypercube operating in MIMD mode. Since communication steps in a $\sqrt{p} \times \sqrt{p}$ mesh is bounded by $\Theta(\sqrt{p})$ which is the dominating factor for selection and sorting in the traditional mesh-connected computers when a unit communication costs more than a unit computation, our interest is hence to find out whether we can achieve $O(t^{sel}_{comm} + t^{sel}_{comp} \log r) = O(\sqrt{p} + t^{sel}_{comp} \log r)$ time complexity, that is, to increase only the computation time for data comparisons of single-element selection by factor $\log r$ while maintaining the same order of communication time, for multiselection on mesh. We say such time complexity *time-optimal.* Finding time-optimal multiselection algorithms to minimize the computation time on mesh is not only interesting in theory but also significant for future generations of mesh-connected computers in which communication can be achieved through optical wave guides or radio frequencies and thus is less important than computation.

In this paper we present a set of efficient parallel algorithms for selection and multiselection in mesh-connected computers, and give an affirmative answer to the above question for meshes that operate in MIMD mode, namely *MIMD mesh.* We start with developing an efficient algorithm for single-element selection on mesh and then move on to develop efficient algorithms for multiselection that call single-element selection. The main contributions of this paper are:

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

All our algorithms are explicit in time complexity and contain no hidden cost. They are readily implementable on the specified mesh-connected computers.

In the next section, we introduce some background knowledge. We then present our single-element selection algorithm in Section 3 and multiselection algorithms in Section 4. Section 5 concludes the paper with some remarks. A preliminary short version of this paper was reported in [20].

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 given by Batcher [11]:

Theorem 1 (Batcher's Theorem) *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*

$$MIN : \min(a_0, a_{\frac{m}{2}}), \min(a_1, a_{\frac{m}{2}+1}), \ldots, \min(a_{\frac{m}{2}-1}, a_{m-1});$$
$$MAX : \max(a_0, a_{\frac{m}{2}}), \max(a_1, a_{\frac{m}{2}+1}), \ldots, \max(a_{\frac{m}{2}-1}, a_{m-1}).$$

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

Parallel merging and sorting can be achieved by simply performing a series of BCE operations [11, 25]. They are called *bitonic merging* and *bitonic sorting* respectively.

It was observed [5] that, like merging and sorting [25], parallel selection can also be done by performing a sequence of parallel comparisons, 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.* The following theorem gives the pivot sequence for parallel comparisons for selecting k smallest elements:

Theorem 2 *[5] For the problem of selecting k smallest elements from n elements in n processors, the pivot sequence for parallel comparisons is*

$$I_0, I_1, \ldots, I_{t-1}, t, I_{t-1}, t+1, I_{t-1}, \ldots, h-2, I_{t-1}, h-1, \tag{1}$$

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} ($0 \le i \le t-1$).

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. For the pivot sequence in Theorem 1, the order setting of data comparisons is determined as follows:

1. For comparisons at pivots in I_j ($0 \le j \le t-1$), the order at processor $i = i_{h-1} i_{h-2} \cdots \imath_0$ is ascending (same as processor's index order) if $i_{h-1} \oplus i_{h-2} \oplus \cdots i_{j+1} = 0$ and descending otherwise, where $\oplus$ is the "exclusive-OR" operator.

2. For comparisons at other pivots than those in I_j ($0 \le j \le t-1$), the order is always ascending.

Data comparison over a pair of processors is carried out in the following way: the processor with larger address (P_i) sends its data to the other processor ($P_{i'}$) that will carry out the comparison and send the larger element back to P_i if the order is *ascending* or the smaller one to P_i otherwise.

Clearly transferring a fixed-sized data (packet) between two neighbouring processors in a mesh-connected computer requires one single routing step and can hence be done in constant time.

For a $\sqrt{p} \times \sqrt{p}$ mesh-connected computer (MCC), there are three main schemes for processor indexing: *row-major*, *shuffled row-major* and *snake-like.* For processor at position (x, y), $\mathrm{P}(x, y)$, in the MCC, $0 \le x, y \le \sqrt{p} - 1$, let $x = x_{q-1} x_{q-2} \cdots x_0$ and $y = y_{q-1} y_{q-2} \cdots y_0$ be respectively the binary indexing for the column and row where $\mathrm{P}(x, y)$ stays. The row-major indexing on $\mathrm{P}(x, y)$ is $y_{q-1} y_{q-2} \cdots y_0 x_{q-1} x_{q-2} \cdots x_0$. The shuffled row-major indexing on $P(x, y)$ shuffles its row-major indexing, that is, it uses $y_{q-1} x_{q-1} y_{q-2} x_{q-2} \cdots y_0 x_0$ to index $\mathrm{P}(x, y)$. The snake-like indexing is the same as row-major indexing, except that the order of indices of processors on each even-indexed row (i.e. even y) is reversed. Positions of cells in the mesh that hold processors are statically indexed in row-major order.

We say that processors are indexed in scheme I (any of the above three) if their indices are generated in (row-major) order of their resulting binary indices after applying scheme I. Note that in the above we index processor indices rather than mesh cell positions, where the latter generates processor indices on their binary codes before applying indexing scheme and then place them to mesh positions defined by the resulting binary codes after applying the indexing scheme. It is not difficult to see that these two opposite approaches on the same indexing scheme are inverse to each other.

Figure 1 shows these three processor indexing schemes on an MCC.

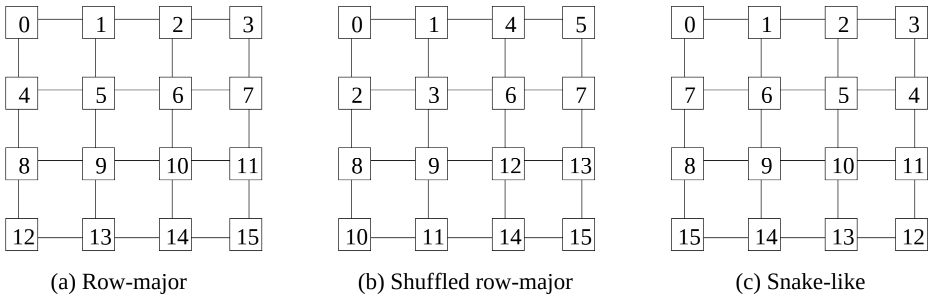

Figure 1: Three processor indexing schemes for mesh-connected computers

3 Single-element selection

We now consider the problem of selecting the kth element from n given elements on an MCC with $p \leq n$ processors. We will later use this algorithm as a building block to devise efficient algorithms for multiselection. Following the literature for ease of analysis, we assume that processors in the MCC are connected in 2-dimensional $\sqrt{p} \times \sqrt{p}$ square mesh.

For selection on MCC when $p = n$, there have been several algorithms [4, 13, 15, 18] 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 [4] based on the *bitonic selection network* [5] is the first selection algorithm on MCC to our knowledge. 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 selection on MCC in the general case when $p \leq n$, the only known result is [13] 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 [4], we can derive a more efficient algorithm for this general case than that of [13] when p is large.

3.1 Selection when $p = n$

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) defined by Theorem 2.

After the above data comparisons, the k smallest elements are stored in processors from 0 to $k-1$ as a bitonic sequence, where the two monotonic sequences of length $k/2$ of opposite order are stored in processors with interleaved indices (one sequence at even indices and the other at odd indices). With the properties of bitonic sequence, we know that the kth element is the larger one between the two largest elements of the two monotonic sequences held by processors $k/2-1$ and $k/2$ respectively. This results in the following general paradigm for parallel selection, where default order setting is ascending.

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

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

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

3. **for** $i = 0$ **to** $p-1$ **do in parallel**
       Set order "ascending";
       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$_0$.

**end.**
```

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

Given a $\sqrt{n} \times \sqrt{n}$ MCC, we use *shuffled row-major* indexing scheme to index the processors in the MCC. It was shown in [4] that among the three popular indexing schemes of row-major, shuffled row-major and snake-like, shuffled row-major indexing results in the least number of routing steps for carrying out data comparisons following the pivot sequence (1). This can be easily seen from the fact that among these three schemes only shuffled row-major indexing results in the number of routing steps for data comparisons at pivot j non-increasing on the value of j.

Lemma 1 *In shuffled row-major indexing of mesh the number of routing steps required for carrying out data comparisons at pairs of processors whose indices differ at pivot j is $2^{\lfloor j/2 \rfloor}$.*

Proof

In the row-major indexing of cell positions (addresses) in a $2^q \times 2^q$ mesh $i = i_{2q-1} i_{2q-2} \cdots i_1 i_0$, it is clear that position i is 2^u steps away from both $i^{(u)}$ along the x-coordinate and $i^{(q+u)}$ along the y-coordinate for any $0 \leq u \leq q-1$. Because shuffled row-major indexing shuffles processor $i = i_{2q-1} i_{2q-2} \cdots i_1 i_0$ to $i' = i_{2q-1} i_{q-1} i_{2q-2} i_{q-2} \cdots i_q i_0 = i'_{2q-1} i'_{2q-2} \cdots i'_1 i'_0$, equivalently it places processor $i' = i'_{2q-1} i'_{2q-2} \cdots i'_1 i'_0$ under the shuffle row-major indexing scheme to mesh position $i = i_{2q-1} i_{q-1} i_{2q-2} i_{q-2} \cdots i_q i_0$ which is always in row-major order. Thus the distances from i' to $i'^{(j)}$ and $i'^{(j+1)}$ for even j equal the distances from i to $i^{(\lfloor j/2 \rfloor)}$ and $i^{(q+\lfloor j/2 \rfloor)}$ in the mesh respectively, which is $2^{\lfloor j/2 \rfloor}$. Since for even j $\lfloor j/2 \rfloor = \lfloor (j+1)/2 \rfloor$, the distance from i' to $i'^{(j)}$ is $2^{\lfloor j/2 \rfloor}$ for any $0 \leq j \leq 2q-1$. This proves the lemma. $\Box$

By Lemma 1 we have the following complexity for algorithm ParSelect implemented on an MCC:

- Routing steps:

$$R_{n,k} = 2\left(\sum_{i=0}^{t-1} \sum_{j=0}^{i} 2^{\lfloor j/2 \rfloor} + (h-t-1) \sum_{i=0}^{t-1} 2^{\lfloor j/2 \rfloor} + \sum_{i=0}^{h-1} 2^{\lfloor j/2 \rfloor}\right) = O(\sqrt{p}). \tag{2}$$

- Data comparisons:

$$C_{n,k} = \sum_{i=0}^{t-1} \sum_{j=0}^{i} + (h-t-1) \sum_{i=0}^{t-1} + \sum_{j=0}^{i} = O(\log(n/\sqrt{k}) \log k). \tag{3}$$

Hence the time complexity of the algorithm is

$$T(n,k) = O(R_{n,k} + C_{n,k}) = O(\sqrt{p} + \log(n/\sqrt{k}) \log m). \tag{4}$$

Figure 2 depicts an example of the implementation of algorithm ParSelect on a 4×4 MCC.

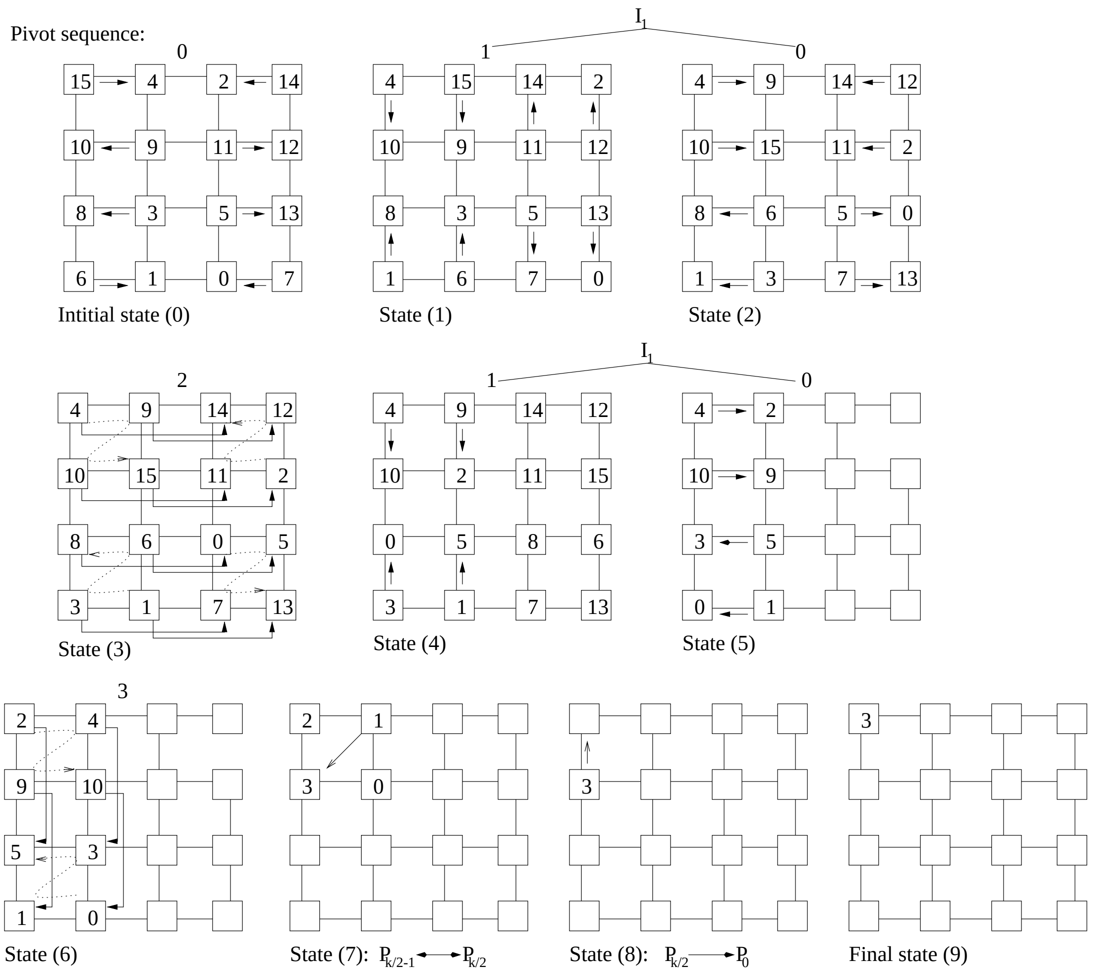

Figure 2: Parallel selection on a 4 × 4 MCC ($k = 4$, $n = 16$)

3.2 Selection when $p \leq n$

For the general case when $p \leq n$, the only known result is [13] 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 [4], we can derive a more efficient algorithm for this general case than that of [13].

When $p \leq n$, 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. We can compute $\min(B_i, B_j)$ and $\max(B_i, B_j)$ as follows:

1. Processor j sends B_j to processor i;
2. Processor i selects the median of the elements in (B_i, B_j) and use it to partition all elements into $\min(B_i, B_j)$ and $\max(B_i, B_j)$;
3. Processor i sends back $\max(B_i, B_j)$ to processor j.

In an MCC with shuffled row-major indexing, for $j = i^{(w)}$ clearly this can be done in $2^{\lfloor w/2 \rfloor + 1}$ routing steps and $O(n/p)$ computation steps.

A sequence $\{B_0, B_1, \ldots, B_{m-1}\}$ is said *block-monotonic* if no element in B_i is greater than any element in block B_{i+1} for $0 \leq i < m-1$ [21]. A *block-bitonic* sequence consists of two block-monotonic sequences arranged in opposite orders, one ascending and the other descending.

Theorem 3 ([21]) *Let $B_0, B_1, \ldots, B_{m-1}$ be a block-bitonic sequence and*

$$\overline{MIN} = \{\min(B_0, B_{\frac{m}{2}}), \min(B_1, B_{\frac{m}{2}+1}), \ldots, \min(B_{\frac{m}{2}-1}, B_{m-1})\};$$
$$\overline{MAX} = \{\max(B_0, B_{\frac{m}{2}}), \max(B_1, B_{\frac{m}{2}+1}), \ldots, \max(B_{\frac{m}{2}-1}, B_{m-1})\}.$$

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.* Same as bitonic merging, carrying out a series of block-BCE operations on a block-bitonic sequence will result in a block-sorted sequence. We call this procedure *block-bitonic merging.* Our selection in this setting can hence be realized by an interleaved sequence of block-BCE operations and block-bitonic mergings. We present our parallel selection algorithm for the general case when $p \leq n$ as follows.

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

1. if $k \leq n/p$ 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 $k$th element must be within the collection of $k$ smallest numbers of each local data
   block.*}

2. if $k > n/p$ then
       for $i = 0$ to $p - 1$ do in parallel
           for $j = 0$ to $t - \log(n/p) - 1$ do
               if $i_{h-1} \oplus i_{h-2} \oplus \cdots \oplus i_{j+1} = 0$ then
                   then set order "ascending"
                   else set order "descending"
               for $q = j$ downto 0 do
                   Block-BCE operation between $\mathrm{P}_i$ and $\mathrm{P}_{i(q)}$;
   {*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-monotonic $k$-sequences, where all successive pairs form
   a series of block-bitonic $2k$-sequences.*}

3. for $i = 0$ to $p - 1$ do in parallel
       for $j = \max\{t - \log(n/p), 0\}$ to $h - 1$
           Set order "ascending";
           Block-BCE operation between $P_i$ and $\mathrm{P}_i^{(j)}$;
           if $(t > \log(n/p))$AND$(j < h - 1)$ then
               if $i_{h-1} \oplus i_{h-2} \oplus \cdots \oplus i_{t-\log(n/p)} = 0$
                   then set order "ascending"
                   else set order "descending"
               for $q = t - \log(n/p) - 1$ downto 0 do
                   Block-BCE operation between $P_i$ and $\mathrm{P}_{i(q)}$;
   {*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-monotonic, until
   there is only one $\overline{MIN}$ sequence containing the $k$ smallest numbers left.*}

4. Compare the two data block in $\mathrm{P}_{\frac{kp}{2n}-1}$ and $\mathrm{P}_{\frac{kp}{2n}}$ and select the maximum (store it at $\mathrm{P}_0$
   and $x$).

end.
```

Analysis of time complexity of the above algorithm is easy:

- The algorithm has exactly the same number of routing steps as algorithm ParSelect — the difference is only that it has a packet size n/p instead of 1 in ParSelect; So

$$R_{n,k,p} = \sqrt{p}. \tag{5}$$

- Instead of $O(1)$ time a single data comparison in ParSelect, it requires $O(n/p)$ time for median-selection and block-splitting. So its computation time dominated by the two loops in Steps 2 and 3 is

$$\begin{aligned} C_{n,k,p} &= O(\frac{n}{p}(\sum_{j=1}^{\log\frac{kp}{2n}}\sum_{q=0}^{j} + \sum_{j=\log\frac{kp}{n}}^{\log(p/2)}(1+\sum_{q=0}^{\log\frac{kp}{2n}}))) \\ &\leq O(\frac{n}{p}(\log^2(kp/n) + \log(n/k)\log(kp/n))) = O(\frac{n}{p}\log p\log(kp/n)). \end{aligned}$$

Hence the time complexity of the algorithm is

$$T(n,k,p) = O(R_{n,k,p} + C_{n,k,p}) = O(\sqrt{p} + \frac{n}{p}\log p\log(kp/n)) \tag{6}$$

As a result, we have the following theorem:

Theorem 4 *Selecting the kth element from n unordered elements on a $\sqrt{p}\times\sqrt{p}$ MCC for any $p \leq n$ 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 [13]. Since $n/p^{2/3} = \Omega(\frac{n}{p}\log p\log(kp/n))$, and $p\log(n/p) = \Omega(\sqrt{p} + \frac{n}{p}\log p\log(kp/n))$ when $p \geq n^{\frac{1}{2}+\epsilon}$ for any fixed constant $0 < \epsilon < \frac{1}{2}$, our time complexity is clearly better than that of [13] when p is greater than $\sqrt{n}$.

Note that [13] 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 time 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 comes from the $\Omega(n)$ lower time bound for selection.

Combining our algorithm with that of [13], we may apply our algorithm when $p \geq n^{\frac{1}{2}+\epsilon}$ and the algorithm of [13] when $p < n^{\frac{1}{2}+\epsilon}$. This leads to the following result immediately:

Corollary 1 *Selecting the kth element from n unordered elements on a $\sqrt{p}\times\sqrt{p}$ MCC, $p \leq n$, 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 a 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. In the following we will present two efficient algorithms using different approaches, one running in SIMD mode and the other in MIMD mode. Both of them employ our general-case single-element selection algorithm described in the previous section.

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 $r \leq \log p / \log\log p$, 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 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** $r \leq 3$ **then**
       SelectMesh ($S$, $k_i$, $p$, $x_i$) 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 $S$ and $K$ each into four subsets:
       $S_{SW} = \{x \mid x \leq x_1\}$;   $K_{SW} = \{k \mid k < n/4\}$;
       $S_{SE} = \{x \mid x_1 \leq x \leq x_2\}$;   $K_{SE} = \{k \mid n/4 \leq k < n/2\}$;
```

```
        $S_{NW} = \{x \mid x_2 \le x \le x_3\}$;   $K_{NW} = \{k \mid n/2 \le k < 3n/4\}$;
        $S_{NE} = \{x \mid x \ge x_3\}$;   $K_{NE} = \{k \mid k \ge 3n/4\}$;
  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.*}
end.
```

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 [8].
- 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 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. Figure 3 depicts the above 4 routing steps for Step 5.

- Using similar approach as above ("folding" and "unfolding"), Step 6 load balancing within each M_i can be completed in $O(\sqrt{p})$ time.

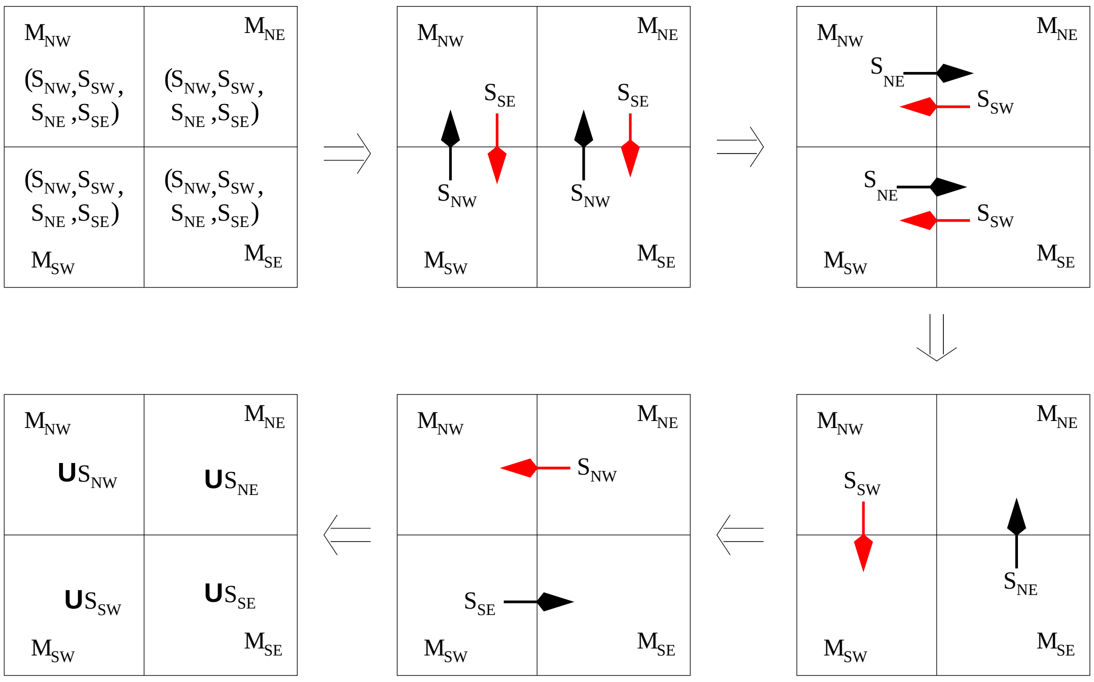

Figure 3: Data redistribution in MCC using permutation routing

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 $r \le \log p/\log\log p$. 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

$$T(n,r,p) = \begin{cases} O(\sqrt{p} + \frac{n}{p}\log^2 p)), & \text{if } r \le 3, \\ O(n\log r), & \text{if } p \le 3, \\ O(\sqrt{p} + \frac{n}{p}\log^2 p) + \min\{T(n-3r/4, r/4, p/4), T(n/4, r, p/4)\}, & \text{if } r, p > 3. \end{cases} \tag{7}$$

The solution to the above recurrence is

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

We have therefore our first result on multiselection on MCC:

Theorem 5 *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 $p \le n$.*

Clearly, for either $p \ge n^{\frac{1}{2}+\epsilon}$ or $r \ge \log p$, $0 < \epsilon < \frac{1}{2}$, 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 [13] to replace SelectMesh when $p < n^{\frac{1}{2}+\epsilon}$, the following corollary holds immediately:

Corollary 2 *Selecting r elements at specified ranks in an arbitrary set of n elements can be completed on an SIMD mesh-connected computer with $p \le n$ 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 $p \le n$.*

4.2 MIMD algorithm

We now show how to improve the time complexity of the above algorithm empowering the processors in the MCC from SIMD execution to MIMD execution. We target to obtain a (time) optimal MCC multiselection algorithm that requires $O(t^{sel}_{comp} logr)$ computation time. The communication time for any algorithm on an $\sqrt{p} \times \sqrt{p}$ MCC is always lower-bounded by $\Omega(\sqrt{p})$. In an MIMD MCC, all processors may execute different instructions at the same time. This lifts our previous requirement of topological identicality on all submeshes in each phase of recursive partitioning, and allows us to partition the mesh more flexibly into any-shape submeshes.

Our strategy for multiselection on MIMD MCC M is to alternatively partition S, K and M on the mid-element of K and the median of S so that all processors in M are fully utilized while each holds a fixed n/p data of S. We call a partition on the mid-element of K k-partition and on the median of S x-partition. The above alternation guarantees that the size of each subset of K and S after each cycle of k-partition and x-partition is smaller than half of the sizes of K and S respectively. In order to keep each submesh's shape as square as possible to save the routing cost, each phase partitioning splits the underlying mesh along the longer side (length) in case if the two sides are not equal. It is not difficult to see that after two cycles of k-partition and x-partition, either (1) both sides of each submesh are smaller than half of their previous lengths before the cycles, or (2) the longer side of each submesh is smaller than one-fourth of its previous length before the cycles, because the x-partition in each cycle halves the mesh along one (longer) side. Clearly, both of these cases result in the size of the submesh after two cycles smaller than one-fourth of the size before the cycles.

Let m is either the mid-element of K or the median of S when appropriate. Each phase partitioning splits S and K each into two subsets S_i and K_i, $i = 1, 2$, on m, 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 $|S_i|$. In the worst scenario M_i is a row-major "strip" in M with side $|S_i|/c_i \times c_i$, where $1 \leq c_i \leq \sqrt{p} - 1$ is a small constant. In order to reduce the size of K_i and S_i most, we use k-partition for the larger subset of K to split it in to two halves and x-partition for the smaller subset to split the subset of S into two halves. In case of $|K_L| = |K_G|$, the direction setting depends on sizes of S_L and S_G: use k-partition for the smaller and x-partition for the larger.

We first give the following procedure for the two partitioning procedures k-partition and x-partition.

```
Procedure $e$-Partition ($S$, $K$, $u$, $v$, $S_L$, $S_G$, $K_L$, $K_G$, $u_L$, $v_L$, $u_G$, $v_G$, $flag$)
  {*Partition $S$ and $K$ according to $e$ on $u \times v$ mesh $M$, $u \leq v$, $S = \{x_1, x_2, \ldots, x_n\}$,
  $K = \{k_1, k_2, \ldots, k_r\}$.*}

1. **if** $e =' k'$ **then** $m = k_{r/2}$ **else** $m = n/2$;

2. SelectMesh ($S$, $m$, $p$, $x^*$);
   {*Select the $k_{r/2}$th element in $S$ and store it in $x^*$.*}

3. Partition $S$ and $K$ each into two subsets:
      $S_L = \{x \in S \mid x \leq x^*\}$;   $K_L = \{k \in K \mid k < m\}$;
      $S_G = \{x \in S \mid x \geq x^*\}$;   $K_G = \{k \in K \mid k \geq m\}$;

4. Move $S_i$ to $u_i \times v_i$ submesh $M_i$ of $M$ for $i = L, G$, where $u_i = u$, $v_L = \lceil \frac{p/n}{|S_i|} u \rceil$ and
   $v_G = v - v_L$;
   {*Partition $M$ in $u$-major. In case if $\frac{p/n}{|S_i|} u < u$, mask some processors inactive.*}

5. Balance load within $M_i$ for $i = L, G$ in parallel;

6. **if** $e =' k'$ **then** $flag =' x'$ **else** $flag =' k'$.
   {*Set the partition direction for the next phase.*}

**end.**
```

Our algorithm for multiselection is a call MSelectMesh2 (S, K, $\sqrt{p}$, $\sqrt{p}$) to the following procedure:

```
Algorithm MSelectMesh2 ($S$, $K$, $u$, $v$)
  {*Select all elements from $S$ 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. if $r \geq \log p$ then
         $k$-partition($S$, $K$, $\sqrt{p}$, $\sqrt{p}$, $S_L$, $S_G$, $K_L$, $K_G$, $u_L$, $v_L$, $u_G$, $v_G$, $flag$);
       else $x$-partition($S$, $K$, $\sqrt{p}$, $\sqrt{p}$, $S_L$, $S_G$, $K_L$, $K_G$, $u_L$, $v_L$, $u_G$, $v_G$, $flag$);

4. for $i = L, G$ do in parallel
       while $|K_i| \geq 3$ do in parallel
           $(a, b) = (L, G)$;
           if $flag =' k'$ then
               $k$-partition($S_a$, $K_a$, $u_a$, $v_a$, $S_L$, $S_G$, $K_L$, $K_G$, $u_L$, $v_L$, $u_G$, $v_G$, $flag$);
               $k$-partition($S_b$, $K_b$, $u_b$, $v_b$, $S_L$, $S_G$, $K_L$, $K_G$, $u_L$, $v_L$, $u_G$, $v_G$, $flag$);
           if $flag =' x'$ then
               $x$-partition($S_a$, $K_a$, $u_a$, $v_a$, $S_L$, $S_G$, $K_L$, $K_G$, $u_L$, $v_L$, $u_G$, $v_G$, $flag$);
               $x$-partition($S_b$, $K_b$, $u_b$, $v_b$, $S_L$, $S_G$, $K_L$, $K_G$, $u_L$, $v_L$, $u_G$, $v_G$, $flag$);
   {*Alternatively partition $(S_L, K_L)$ on $M_L$ and $(S_G, L_G)$ on $M_G$ in parallel.*}
       Call $SelectMesh$ to select $K_i$ from $S_i$ on $M_i$. {*$|K_i| \leq 2$.*}

end.
```

For $p = uv$ and $u \leq v$ at any time, all other steps except the recursion in Step 6 can be completed in $O(v + \frac{n}{p}\log^2 p)$ time. Since each partition in the algorithm is followed by a partition in the alternative direction (in the sense of k- and x-) on each subset (K_i, S_i, M_i), a cycle of two successive partitioning halves K_i and S_i (hence M_i as each processor holds n/p data of S). Since each cycle of of x- and k-partitioning breaks the longer side of the submesh by at least half of its previous length, as shown before any two cycles of k- and x-partition on mesh M_i will split M_i into submeshes whose maximal side is smaller than half of the longer side of M_i and size smaller than $|M_i|/4$. Apparently sizes of K and S are halved by each k-partition and x-partition respectively, and hence quartiled after any two cycles of partitioning. Therefore the time complexity $T(n, r, p)$ of the algorithm can be expressed as the the following recurrence.

$$T(n,r,p) = \begin{cases} O(\sqrt{p} + \frac{n}{p}\log^2 p), \text{ if } r \leq 3, \\ O(n \log r), \text{ if } p \leq 3, \\ O(\sqrt{p} + \frac{n}{p}\log^2 p) + T(n/4, r/4, p/4), \text{ if } r, p > 3. \end{cases} \tag{9}$$

The solution to the above recurrence is

$$T(n,r,p) = O(\sqrt{p} + \frac{n}{p} \log r \log p \log(p/r)). \tag{10}$$

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

Theorem 6 *Selecting r elements at specified ranks in an arbitrary set of n elements can be completed asymptotically on an MIMD mesh-connected computer with p processors in $O(\sqrt{p} + \frac{n}{p} \log r \log p \log(p/r))$ time, for any $p \leq n$.*

If we use the algorithm of [13] instead of SelectMesh in the case when $p < n^{\frac{1}{2}+\epsilon}$, we will be able to make MSelectMesh2 optimal across the whole range of p. Noticing that in any case the recursion in equation (9) executes at most $\log r$ times (until $r = 1$), we have

Corollary 3 *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 $O(\sqrt{p} + \log r \min\{\frac{n}{p} \log p \log(p/r), (\frac{n}{p} + p) \log(n/p)\})$ time.*

5 Concluding remarks

We have presented a set of efficient parallel algorithms for selection and multiselection on a $\sqrt{p} \times \sqrt{p}$ mesh-connected computer. For selecting the kth smallest element from $n \geq p$ elements, our algorithm runs in $O(\min\{\sqrt{p} + \frac{n}{p} \log p \log(kp/n), (\frac{n}{p} + p) \log(n/p)\})$ time and is more computationally efficient than the existing result when $p \geq n^{\frac{1}{2}+\epsilon}$ for any fixed constant $0 < \epsilon < \frac{1}{2}$. For selecting r elements at specified ranks in a given set of n elements, our first algorithm runs 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 with processors operating in the SIMD mode and is time optimal when $p < r$, our second algorithm runs in $O(\sqrt{p} + \log r \min\{\frac{n}{p} \log p \log(p/r), (\frac{n}{p} + p) \log(n/p)\})$ time on MIMD processors and is time optimal for any p and r.

An interesting open problem is how to obtain an optimal multiselection algorithm on an SIMD MCC. This remains to be a topic for our future research.

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

[4] 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.

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

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

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

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

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

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

[13] 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.

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

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

[17] 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.

[18] 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.

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

[20] H. Shen. Efficient parallel algorithms for selection and multiselection on mesh-connected computers. In *Proc. 13th IEEE Int. Parallel Processing Sypm. & 10th IEE Sypm. on Parallel and Distributed Processing (IPPS/SPDP 99)*, Puerto Rico, 1999, pages 426-430.

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

[22] H. Shen. Efficient parallel multiselection in hypercubes. *Parallel Algorithms and Applications*, 14(3): 203-213, 2000.

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

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

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

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