Selection on the Reconfigurable Mesh*

Eric Hao† Philip D. MacKenzie‡ Quentin F. Stout§

Advanced Computer Architecture Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122

Abstract

Our main result is a $\Theta(\log n)$ time algorithm to select the kth smallest element in a set of n elements on a reconfigurable mesh with n processors. This improves on the previous fastest algorithm's running time by a factor of $\log n$. We also show that some variants of this problem can be solved even faster. First we show that a good approximation to the median of n elements can be found in $\Theta(\log\log n)$ time. This can be used to solve two-dimensional linear programming over n equations in $\Theta(\log n \log\log n)$ time, an improvement of $\log n/\log\log n$ time over the previous fastest algorithm. Next, we show that, for any constant $\epsilon > 0$, selecting the kth smallest element in a set of $n^{1-\epsilon}$ elements evenly spaced throughout the mesh can be done in constant time. We also show that one can select the kth smallest element from n b-bit words in $\Theta((b/\log b)\max\{\log^* n - \log^* b, 1\})$ time, which implies that if the elements come from a polynomial range, one can select the kth smallest element in $\Theta(\log n/\log\log n)$ time. In addition, we show that the expected time to select the kth smallest element from n elements, assuming that all possible orderings of the elements are equally likely, is the time to perform bit-addition, which is currently known to be $O(\log^* n)$. Finally, we show an $\Omega(\log\log n)$ time lower bound for finding the maximum of n elements on the rmesh. This implies an $\Omega(\log\log n)$ time lower bound for selection, and is the first known non-trivial lower bound on the rmesh which does not rely on the bandwidth constraints of the mesh and does not restrict the instruction sets of the processors.

*A preliminary version of this paper appeared in the *4th Symposium on the Frontiers of Massively Parallel Computation*.

†Supported by a University of Michigan College of Engineering Benton Fellowship.

‡Supported by an AT&T Fellowship and by NSF/DARPA grant CCR-9004727.

§Supported by NSF/DARPA grant CCR-9004727.

1 Introduction

A reconfigurable mesh, or *rmesh*, consists of a bus in the shape of a mesh which connects a set of processors, but which can be split dynamically by local switches at each processor. By setting these switches, the processors partition the bus into a number of subbusses through which the processors can then communicate. Thus the communication pattern between processors is flexible, and moreover, it can be adjusted during the execution of an algorithm.

The rmesh has begun to receive a great deal of attention as both a practical machine to build, and a good theoretical model of parallel computation. On the practical side, rmeshes have been constructed for use in vision and VLSI applications (see [9]). On the theoretical side, the rmesh has been shown to be able to solve some fundamental theoretical problems very quickly. In fact, the rmesh can solve some problems faster than the PRAM, the model most often used for theoretical parallel computation. For instance, an n processor rmesh can find the parity of n bits in constant time [10], which is significantly faster than an n processor PRAM, which requires $\Omega(\log n/\log\log n)$ time [1]. Of course the rmesh still suffers from the bandwidth constraints of the standard mesh, so that an n processor rmesh needs $\Omega(\sqrt{n})$ time to sort n items, whereas an n processor PRAM can sort n items in $\Theta(\log n)$ time.

In this paper, we will examine the problem of selection on the rmesh. The selection problem is to find the kth smallest element from a totally ordered set of n elements. A special case of this problem is to find the median. The first linear time sequential algorithm for selection was given by Blum et. al. [2]. Parallel solutions were explored in [4, 5, 18, 19, 20, 21]. Of these, ElGindy and Węgrowicz [5] obtained the best previous running time on the rmesh, $\Theta(\log^2 n)$. Their algorithm is a parallelization of the serial selection algorithm of Munro and Paterson [15]. The main bottleneck in their algorithm is the time to find good approximations to the kth element. Using the special properties of the reconfigurable mesh, we dramatically improve the time to find good approximations. Furthermore, we also improve the accuracy of these approximations. Thus we are able to reduce the running time of the selection algorithm to $\Theta(\log n)$.

We also show that one can find an approximate median in $\Theta(\log\log n)$ time. As mentioned in [5], an approximate median can be used in place of an exact median in a parallelized version of Megiddo's two-dimensional linear programming algorithm [12]. ElGindy and Węgrowicz [5] find an approximate median in $\Theta(\log n)$ time, which gives a $\Theta(\log^2 n)$ time algorithm for two-dimensional linear programming. Our faster algorithm for finding an approximate median results in a $\Theta(\log n\log\log n)$ time algorithm for two-dimensional linear programming.

We also examine the selection problem when the number of processors is greater than the number of inputs. Specifically, for any constant $\epsilon > 0$, given $n^{1-\epsilon}$ elements spaced evenly throughout the mesh, we present a constant time selection algorithm.

We also present an algorithm to select the kth smallest element from n b-bit words that works in $\Theta((b/\log b)\max\{\log^* n - \log^* b, 1\})$ time, which implies that if the elements come from a polynomial range, then one can select the kth smallest element in $\Theta(\log n/\log\log n)$ time.

Next, we examine the selection problem when the number of inputs equals the number of processors and all input orderings are equally likely. In this case, we present a selection algorithm which is guaranteed with high probability to run in the time to perform bit-

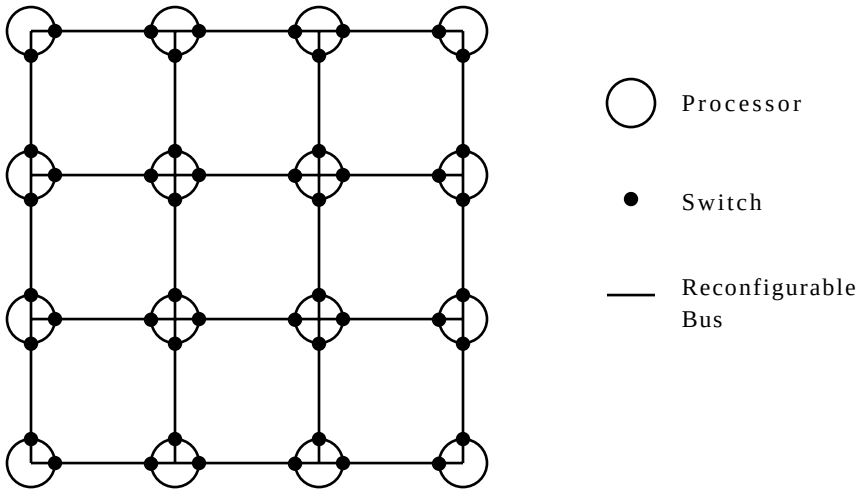

Figure 1: The Reconfigurable Mesh Architecture.

addition. The known $\Theta(\log^* n)$ time algorithm for bit-addition thus implies a $\Theta(\log^* n)$ expected time algorithm for selection.

Finally, we show that the proof of the $\Omega(\log \log n)$ time lower bound for finding the maximum on the PRAM given in Fich, Meyer auf der Heide, Ragde and Wigderson [6] can be converted to give an equivalent lower bound proof for the rmesh. This implies an $\Omega(\log \log n)$ time lower bound for selection, and is the first known lower bound on the rmesh which does not rely on the bandwidth constraints of the mesh and does not restrict the instruction sets of the processors.

2 Reconfigurable Mesh

The reconfigurable mesh [13] of size n consists of a reconfigurable bus in the shape of a $\sqrt{n} \times \sqrt{n}$ grid, with four switches at each intersection point of the grid (three for points on the sides, and two for points in the corners) and a processor at each intersection point which controls the switches and can read data from or write data to the bus. See Figure 1. By dynamically setting the switches, the bus can be subdivided into independent connected components called subbusses. All processors connected to a subbus can read the data on it, but only one processor can write data to a subbus at a time.

The processors operate synchronously. In one time step a processor may

- perform a single operation on words of size $O(\log n)$,
- set any of its switches, or
- write or read data from the bus.

Data placed on a bus reaches all processors connected to the bus in unit time. When the processors in the rmesh can independently connect their north-south switches together and their east-west switches together at the same time, we call this the *cross-over* model. Otherwise we call it the *non-cross-over* model. Li and Stout [9] discuss the difference between

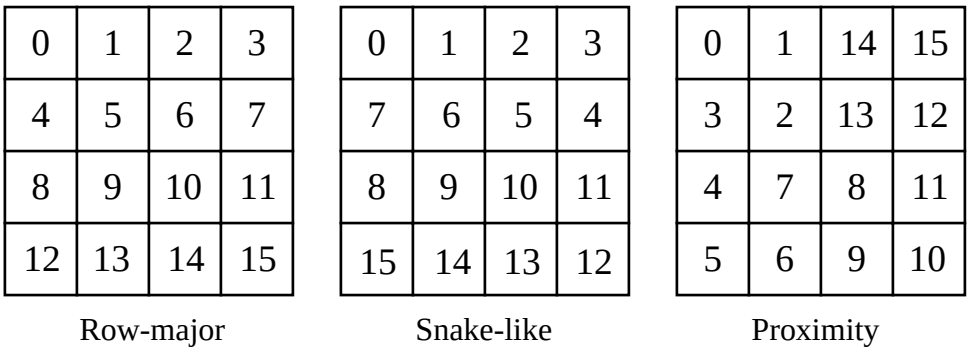

Figure 2: Three Orderings for Mesh Processors.

these two models, and MacKenzie [11] proves an $\Omega(\log^* n)$ time separation between them. Unless otherwise stated, we will assume the non-cross-over model.

We assume each processor knows its row and column indices. In addition we will also want processors to know their rank in some total ordering on the mesh. For our algorithms, we will use three types of orderings, *row major* ordering, *snake-like* ordering, and *proximity* ordering. In row major ordering, we number processors in each row left to right, beginning with the top row, and ending with the bottom row. In snake-like ordering we simply reverse every other row, so that processors with consecutive indices are adjacent. In proximity ordering, we are guaranteed that processors with consecutive indices are adjacent, and that the first quarter of the processors are in the first quadrant, the second quarter in the second quadrant, and so on, and that this property holds recursively (with a suitable change in axes) within the quadrants. See Figure 2.

Often it is assumed that a processor initially knows its position in row-major order, snake-like order, and proximity order. If not, a processor can compute its rank in row-major order or snake-like order from its indices in constant time and its rank in proximity order from its indices in $\Theta(\log n)$ time. Also, by working together, the processors in the rmesh can compute their ranks in proximity order from their indices in $\Theta(\log \log n)$ time.

We now list some known algorithms for the $\sqrt{n} \times \sqrt{n}$ rmesh which will be used in our selection algorithm.

- One processor can broadcast a value to all other processors in constant time.
- Two lists of size $O(\sqrt{n})$ can be merged in constant time.
- $\sqrt{n}$ elements can be sorted in constant time provided they are located in a single row or column, or are located on the diagonal [8, 17].
- n elements can be sorted into row-major order in $\Theta(\sqrt{n})$ time using a standard mesh sorting algorithm [21].
- Prefix bit-addition over n bits stored one per processor can be performed in $\Theta(\log \log n)$ time, using a $\sqrt{n}$ divide-and-conquer technique similar to the one given for finding the minmum of n values in Miller, Prasanna-Kumar, Reisis and Stout [14].
- Bit-addition over n bits stored one per processor can be accomplished in $\Theta(\log^* n)$ time on the cross-over model [7, 16].

- A histogram over h values stored one per processor counts the number of occurances of each value. (We assume h is less than $\sqrt{n}$ and that a histogram is stored in a single row.) Assuming the mesh has been preprocessed so that each submesh of size $h^4 \times h^4$ contains a histogram of its own processors' values, then a histogram for the whole mesh can be found in $O(\max\{\log^* n - \log^* h, 1\})$ time on the cross-over model [7].

3 Selection Algorithm

Our algorithm is based on the rmesh algorithm of El Gindy and Węgrowicz [5], which is in turn based on the serial algorithm of Munro and Paterson [15]. Munro and Paterson's algorithm was designed to be very frugal in memory usage. Though we do not need to worry about memory usage on the rmesh, it turns out their algorithm parallelizes very well. Here we give a simplified explanation of the algorithm of Munro and Paterson.

3.1 Serial Selection Algorithm

This algorithm runs in $O(\log n)$ phases. Each phase is designed to reduce by a constant fraction the number of elements which still have to be considered in selecting the kth element. This is done by forming a sorted sample of size $s = \Theta(\log n)$ from which two good approximations to the kth element can be taken, one which is guaranteed to be below the kth element, and one which is guaranteed to be above the kth element. These approximations will be called the low and high *filters*. To find the sample, first we partition the elements into sublists of size s and sort those sublists. Then we perform a procedure similar to a mergesort on these sublists, except that as sublists are merged, they are also thinned by removing every second element. Thus the sizes of the sorted sublists remains at s. For $n = s2^r$, there are r levels of merges. Munro and Paterson show that the elements $\lceil k/2^r \rceil - r$ and $\lceil k/2^r \rceil$ in the final sample can be used as the filters, and that the number of elements between these filters is at most $(2r-1)2^r$. For $s = 4\log n$, this is less than $\frac{3}{4}n$.

3.2 New Parallel Selection Algorithm

Now we give our rmesh algorithm for selection. We assume that we have a set of elements distributed at most one per processor on an rmesh with n processors, and would like to find the kth element of this set. Initially all the processors contain an element and are *active*. As elements are eliminated as possible candidates, the processors which hold them become *inactive*.

Let $s = \log n$. Procedure Select will iterate until the number of active elements becomes less than s^2, at which time it will sort the remaining elements, and choose the correct candidate.

Procedure Select(k)

Stage 1 Split the mesh into n/s^2 submeshes of size $s \times s$ and sort the elements in these submeshes into row major order. This step takes $O(s)$ time using the sorting algorithm of Thompson and Kung [21].

Stage 2 Number the active processors in row major order within the $s \times s$ submeshes and in proximity order between submeshes. This can be done using a prefix bit-addition to find the proximity ordered ranks, and then altering the ranks within each submesh to correspond to the row-major order. We notice that the ranks of the active elements in each submesh are contiguous in the row-major order, which makes this alteration very simple. The time required by this step is determined by the time to perform prefix bit addition, which is $\Theta(\log \log n)$. Let m be the number of active processors. If $m \bmod 2s^2 \neq 0$, add enough active processors with value ∞ so that $m \bmod 2s^2 = 0$.

Stage 3 If $m = 2s^2$, then arrange the m processors along the first row in $\Theta(s)$ time, sort them in constant time, pick the kth element, and exit.

Stage 4 Compute the sample by calling procedure Sample(k).

Stage 5 Choose the filters and broadcast them to all the processors. Perform a prefix bit addition algorithm to find how many active elements fell below the low filter. Call this value b. Make each processor which contains an element below the low filter or above the high filter inactive.

Stage 6 Let $k \leftarrow k - b$ and go to stage 2.

End Select

To compute the sample, we will first partition the active elements into consecutive groups of size s according to the numbering performed in step 2. These groups will form the base of our "merge tree." We proceed as described below.

Procedure Sample(k)

Stage 1 Partition the elements into consecutive groups of size s. Move the items of a group to the submesh which contains the first element of that group. We do this by moving elements using subbusses formed along the proximity ordering between the submeshes. Note that we have enough bandwidth to do this, and that the items can be moved to the correct busses because they are arranged linearly in each submesh. Now each submesh has an integral multiple of s elements.

Stage 2 If a submesh contains new data (at most $s-1$ new elements), then it can sort these elements in constant time. Thus all the groups it contains are sorted.

Stage 3 For $1 + \log s$ steps, we will thin and merge consecutive pairs of groups in parallel, just as Munro and Paterson did in serial. We will merge the first and second groups in the submesh where the first group resides, the third and fourth group in the submesh where the third group resides, and so on. Any two groups which originally resided in the submesh are already sorted, so we thin and merge them by simply selecting every other item from them. When one of the two groups or part of one of the two groups originally resided in another submesh, we must thin and merge these using the constant time merge algorithm. Note that this can only occur with the last group (and thus at most one group) in each submesh. When the second group in the pair which needs to be merged is not in the submesh, it will need to be transported there from

its own submesh. It can get there by the busses which travel in proximity ordering between the submeshes, and it can be merged in constant time as above. Thus each of these thinning and merging steps takes constant time, and after $1 + \log s$ steps, every consecutive pair of submeshes will have at most one group of s items.

Stage 4 (The basic structure of this stage is similar to other $O(\log\log n)$ algorithms, such as finding the maximum [14].) For $\log\log(m/s^2)$ steps, at each step i merge $2^{2^{i-1}}$ consecutive groups of elements and use every $2^{2^{i-1}}$th element for the new group, thus thinning the large sorted list down to a new sorted group of s elements. We do this as follows. Form $m/2s^2 2^{2^{i-1}}$ supergroups, each consisting of $2^{2^{i-1}}$ consecutive groups. Also form $n/s^2 2^{2^i}$ super-submeshes, each of size $2^{2^{i-1}}s \times 2^{2^{i-1}}s$. We would like to move each supergroup of elements into the super-submesh which contains the first group in the supergroup. To do this, in each super-submesh, we sort all elements which have their leader in the super-submesh where they reside. Then we sort those which will need to travel to another super-submesh. Now we simply put the traveling elements onto the correct subbusses formed in proximity order between these super-submeshes Each super-submesh will then have the correct $2^{2^{i-1}}$ groups of elements, and it can sort and thin these in constant time.

End Sample

Stages 1 and 2 in the procedure Sample take constant time. Stage 3 uses $1 + \log s = O(\log\log n)$ steps which each take constant time, and stage 4 uses $\log\log(m/s^2) = O(\log\log n)$ steps which each take constant time. Thus, procedure Sample takes $O(\log\log n)$ time. We now analyze how good the resulting sample is. Let L_{ij} and M_{ij} be the least and most number of elements which can precede element j at step i of stage 3. From Munro and Paterson [15], $L_{ij} = j2^i - 1$ and $M_{ij} = (i + j - 1)2^i$. Then at the start of the stage 4, $L_{1+\log s,j} = 2js - 1$ and $M_{1+\log s,j} = 2s(\log s + j)$.

Now let L'_{ij} and M'_{ij} be the least and most number of elements which can precede element j at step i of stage 4. We present the following lemma.

Lemma 3.1 *$L'_{ij} = js2^{2^i} - 1$ and $M'_{ij} \le (i + j + \log s)s2^{2^i}$.*

Proof: Obviously $L'_{0j} = 2js - 1$ and $M'_{0j} = 2s(j + \log s)$. From the way we construct these groups in the algorithm, we can see that

$$L'_{ij} = \min\{L'_{i-1,p_1} + L'_{i-1,p_2} + \ldots + L'_{i-1,p_{2^{2^{i-1}}}} + 2^{2^{i-1}} - 1 \quad | \quad p_1 + p_2 + \ldots + p_{2^{2^{i-1}}} = 2^{2^{i-1}}j; \; p_1 > 0; p_2, p_3, \ldots, p_{2^{2^{i-1}}} \ge 0\}$$

and

$$M'_{ij} = \max\{M'_{i-1,p_1} + M'_{i-1,p_2+1} + \ldots + M'_{i-1,p_{2^{2^{i-1}}}+1} \quad | \quad p_1 + p_2 + \ldots + p_{2^{2^{i-1}}} = 2^{2^{i-1}}j; \; p_1 > 0; p_2, p_3, \ldots, p_{2^{2^{i-1}}} \ge 0\}.$$

From these equations, the lemma follows inductively. □

Now let $r = \log\log(m/s^2)$, the number of steps in Stage 4. We must choose the low and high filters from the sample after step r of Stage 4. We will call these u and v respectively. We must have

$$k - 1 \geq M'_{ru} = (r + u + \log s)s2^{2^r}$$

and

$$k - 1 \leq L'_{rv} = vs2^{2^r} - 1,$$

so we will choose $u = \lceil (k-1)/s2^{2^r} \rceil - \log s - r$, and $v = \lceil k/s2^{2^r} \rceil$. The kth element must then be one of u or v or lie between them in the order. Given $s = \log n$, the number of elements between the uth and vth elements is at most

$$\begin{aligned} M'_{rv} - L'_{ru} - 1 &= (r + \log s + v - u)s2^{2^r} \\ &= 2(r + \log s)s2^{2^r} + 1 \\ &\leq 2(\log\log\frac{m}{s^2} + \log s)\frac{sm}{s^2} + 1 \\ &\leq O(m/\sqrt{\log n}) \end{aligned}$$

Theorem 3.1 *In $\Theta(\log n)$ time, one can select the kth of n items on an n processor reconfigurable mesh.*

Proof: Use procedure Select. Each iteration of the main loop in procedure Select, which includes the call to Sample and a constant number of prefix bit-additions, takes $\Theta(\log\log n)$ time, and reduces the number of active elements by a factor of $\sqrt{\log n}$. Therefore this loop is executed $O(\log n/\log\log n)$ times, giving a total of $O(\log n)$ time. Stages 1 and 3 of Select are each executed only once, and each takes $\Theta(\log n)$ time. Therefore the procedure Select is completed in $\Theta(\log n)$ time. □

Theorem 3.2 *Given m elements stored at most one per processor on a $\sqrt{n} \times \sqrt{n}$ rmesh, an approximate median (an element with rank between $m/4$ and $3m/4$) can be found in $\Theta(\log\log n)$ time.*

Proof: First partition the mesh into submeshes of size $8\log\log n \times 8\log\log n$ and sort them. Then use procedure Sample, but with $s = 8\log\log n$. Now examine how many elements are below and above element $s/2$ in the sample after step $r = \log\log m/s^2$.

$$\begin{aligned} L'_{r,s/2} &= (s/2)s2^{2^r} - 1 \\ &= m/2 - 1 \end{aligned}$$

and

$$\begin{aligned} M'_{r,s/2} &= (r + (s/2) + \log s)s2^{2^r} \\ &\leq 3m/4 \end{aligned}$$

Thus at least $m/4$ elements are below and $m/4$ elements are above element $s/2$ in the sample. We choose this element to be the approximate median, and the theorem follows. □

Theorem 3.3 *We can solve the two-dimensional linear programming problem with n equations on an n processor rmesh in $\Theta(\log n \log\log n)$ time.*

Proof: We use a simple parallelization of Megiddo's algorithm [12] for two-dimensional linear programming. This results in $O(\log n)$ calls to a median algorithm. This median algorithm is simply used to remove a constant fraction of the equations from consideration. Thus we can also use an approximate median, namely, an element which is guaranteed to have a fixed fraction of inputs below it, and a fixed fraction above it, in the total order. ElGindy and Węgrowicz [5] also use an approximate median in place of an exact median in their algorithm. From the previous theorem, we can find this approximate median in $\Theta(\log\log n)$ time, which results in a $\Theta(\log n \log\log n)$ time algorithm for two-dimensional linear programming. □

We say m elements are *evenly spaced* throughout the rmesh if there is one element in each $\sqrt{n/m} \times \sqrt{n/m}$ submesh. We examine here the case of the selection problem, when the number of elements is much fewer then the number of processors, and these elements are evenly spaced throughout the rmesh. It turns out that in this case we can perform selection much faster.

Theorem 3.4 *For any constant $\epsilon > 0$, one can select the kth of $n^{1-\epsilon}$ items on an n processor reconfigurable mesh in constant time, assuming the items are spaced evenly throughout the mesh.*

Proof: First we note that we can count the active items in constant time. To do this we simply count the active items inside all $n^\epsilon \times n^\epsilon$ submeshes in parallel in constant time, since there are $\leq n^\epsilon$ items initially evenly spaced in each. Then we start the $\Theta(\log\log n)$ time bit-addition algorithm from the point where the subsquares are of size $n^\epsilon \times n^\epsilon$. Then in $\log(1/2\epsilon)$ steps, the algorithm will be finished.

The selection algorithm proceeds similar to procedure Select except groups of n^ϵ can initially be sorted in constant time, and we will use a sample of size $n^{\epsilon/2}$ in a modified Sample procedure. We will omit Stage 3 from procedure Sample, and begin Stage 4 by merging n^ϵ groups together. Note that procedure Sample will then take $\log(\log m/\epsilon \log n) \leq \log(\frac{1}{\epsilon} - 1)$ steps. The corresponding equations for L'_{ij} and M'_{ij} will be

$$L'_{ij} = n^{\epsilon(2^i-1)} n^{\epsilon/2} j - 1$$

and

$$M'_{ij} = (j + i - n^{-\epsilon/2}) n^{\epsilon(2^i-1)} n^{\epsilon/2}.$$

Now let $r = \log(\log m/\epsilon \log n)$. We must choose the low and high filters from the sample after step r of Stage 4. We will call these u and v respectively. We must have

$$k - 1 \geq M'_{ru} = (r + u - n^{-\epsilon/2}) n^{\epsilon/2} n^{\epsilon(2^r-1)}$$

and

$$k - 1 \leq L'_{rv} = n^{\epsilon(2^r-1)} n^{\epsilon/2} v - 1,$$

so we will choose $u = \lceil (k-1)/n^{\epsilon/2} n^{\epsilon(2^r-1)} \rceil - n^{-\epsilon/2} - r$, and $v = \lceil k/n^{\epsilon(2^r-1)} n^{\epsilon/2} \rceil$. The kth element must then be one of u or v or lie between them in the order. The number of elements between the uth and vth elements is at most

$$\begin{aligned} M'_{rv} - L'_{ru} - 1 &= (r - n^{-\epsilon/2} + v - u) n^{\epsilon(2^r-1)} n^{\epsilon/2} \\ &= 2(r - n^{-\epsilon/2}) n^{\epsilon(2^r-1)} n^{\epsilon/2} + 1 \\ &\leq 2(\log m/\epsilon \log n - n^{-\epsilon/2}) m n^{-\epsilon/2} + 1 \\ &\leq O(m n^{-\epsilon/3}) \end{aligned}$$

Thus each iteration, which takes a constant number of steps, reduces the number of active elements by a factor of $n^{\epsilon/3}$. Therefore this loop is executed $\leq \frac{3}{\epsilon}(1-\epsilon)$ times, and thus the whole procedure takes constant time. □

4 Selection on b-bit words

To select the kth of n b-bit words, we can use a different type of algorithm. This algorithm will find the kth word in $5b/\log b$ stages by honing in on the kth word $(\log b)/5$ bits at a time. Using a preprocessing phase which takes $\Theta(b/\log b)$ time, we can set up the rmesh so that each of the $5b/\log b$ stages takes $O(\max\{\log^* n - \log^* b, 1\})$ time (in the cross-over model). Thus the time of the algorithm will be $O((b/\log b)\max\{\log^* n - \log^* b, 1\})$. This is asymptotically faster than the comparison based selection algorithm in Section 3 when $b = o(\log n \log\log n)$. For the special case of $b = O(\log n)$ (i.e. the words come from a polynomial range), the algorithm takes $O(\log n/\log\log n)$ time.

Our algorithm is outlined in procedure Selectword, which consists of four stages. Stages 1, 2 and 3 are involved in preprocessing. This preprocessing divides the rmesh into submeshes and does all the rest of its work in parallel in each submesh. Its purpose is to create records in each submesh which will be useful in stage 4. These records are formed as follows. For each level i from 1 to $5b/\log b$ consider the elements of each submesh partitioned into groups according to their top $i(\log b)/5$ bits. From one level to the next, one can think of groups as subdividing into one or more groups depending on the next $(\log b)/5$ bits. The preprocessing phase forms records which store the number of elements which belong to each group at each level i. Because of the limited storage in the submesh, we will only form records for those groups with more than $5b/\log b$ elements. If a group doesn't subdivide at some level, we will not form a new record for that group at that level, but simply modify the record from the previous level to indicate that the group has not subdivided. The records we create will be of the form [level,bits,number,hold]. The level field indicates to which level this record belongs, from 1 to $5b/\log b$. The bits field indicates the group to which this record belongs. The number field indicates how many elements in the submesh are in this group, and the hold field indicates for how many levels this group has gone without changing. Zero records are formed in stage 4 to fill in for groups which have no elements in the submesh. A zero record at level i has the form $[i, x, 0, 0]$, where x is a b-bit word with the correct top $(i-1)(\log b)/5$ bits of the kth word plus one of the $b^{1/5}$ possibilities for the next $(\log b)/5$ bits, depending on which group the record is used for.

Stage 4 hones in on the kth word $(\log b)/5$ bits at a time. At each iteration i of stage 4, for all the elements with the same upper $(i-1)(\log b)/5$ bits of the kth element, we will form a histogram over the $b^{1/5}$ possible values of the next $(\log b)/5$ bits. We use the records created in the first three stages to form the required histogram within each submesh quickly, and then use the algorithm of Jang, Park, and Prasanna [7] to form the histogram for the whole mesh. By performing a prefix calculation over this histogram, we can decide between which values the kth element would fall, and thus we can discover the next $(\log b)/5$ bits of the kth element.

Procedure Selectword(k, b)

Stage 1 Let $s = 5b/\log b$ and $h = (\log b)/5$. Split the mesh into n/s^2 submeshes of size $s \times s$ and sort the elements in each submesh in parallel into snake-like ordering. Let the element at processor i be e_i. Let all the processors be *active*.

Stage 2 For $i = 1$ to $5b/\log b$ perform stages 2a and 2b.

Stage 2a Within each submesh, form a bus between the processors in the snake-like order. Then disconnect the processors into groups according to the top ih bits of their stored elements, and perform the rest of stage 2a and stage 2b in parallel in each disconnected group. Let j and j' be the highest and lowest indices of processors in the group, respectively. Have processor j' send its index to processor j, and let $d = j - j' + 1$. Then in this group, the highest processor knows that there are d processors containing elements with the same upper ih bits as its element, e_j.

Stage 2b If processor j in the group is active, then do the following.

If $d \leq 5b/\log b$, then store a new record $[i, e_j, d, 0]$ and send a deactivate message to all the processors in the group. This processor and all other processors in this group now make themselves *inactive*.

If $d > 5b/\log b$ and there is no record stored at this processor, then create a new record $[i, e_j, d, 0]$.

If $d > 5b/\log b$ and there is already a record R stored in this processor, check to see if R.number $= d$. If so, then simply increment R.hold. If R.number $\neq d$, then send a message along the bus for all the processors with a record to move it to its predecessor. The records can be moved to their predecessors in one step. Then create a new record, $[i, e_j, d, 0]$, at the highest processor.

Stage 3 Sort all the records in each submesh into row-major order by the first two fields. This guarantees that the records for each level will be spread out enough so that they can be accessed quickly when needed.

Stage 4 For $i = 1$ to $5b/\log b$ perform stages 4a to 4c. After the jth iteration, we will know the top jh bits of the kth word, and this information will be used in Stage 4a to decide which groups to consider at the next iteration. After all iterations we will know all the bits of the kth word.

Stage 4a Stage 4a is performed in each $s \times s$ submesh in parallel. For each position q in the first $b^{1/5} (= 2^h)$ positions along the bottom row of the submesh, we will place a record which contains the number of elements in the submesh which have

the same top $(i-1)h$ bits as the kth word and whose next h bits equal q in binary form.

If $i = 1$, we simply move the first level records to the positions on the bottom row corresponding to the top h bits of their number fields. Note that these records can be moved in constant time because no two records originate from the same column. (This will apply to all the records which need to be moved at any level.) Now we fill the remaining processors on the bottom row with zero records.

If $i > 1$, then assume the record at the previous iteration which contained the correct $(i-1)h$ bits of the kth word is R. If R.number $= 0$, then this submesh does not have any elements with the same upper $(i-1)h$ bits as the kth element, so we fill in all the processors on the bottom row with zero records.

If R.number $\neq 0$, then what we need to do can be broken up into separate cases, depending on other fields of R.

Case 1: (R.hold $\neq 0$) The processor which holds R should decrement this hold field. Then this record should be sent to the processor determined by the next h bits in its bits field corresponding to this level. All the other processors in the first $b^{1/5}$ locations along the bottom row should create zero records.

Case 2: (R.hold $= 0$ **and** R.number $> s$) Consider those records for this group found in stage 2 and sorted in stage 3. Move them to their correct places along the bottom row and create zero records at the unfilled positions.

Case 3: (R.hold $= 0$ **and** R.number $\leq s$) There are at most $5b/\log b$ remaining elements in this category. Using the sorted elements from the first stage of preprocessing, find the number of elements for each group of elements with different next h bits by splitting the elements into groups as in Stage 2a and simply creating a record with the correct number of items in the group at the highest processor in the group. Then move the records to the correct places along the bottom row, and fill in the rest of the positions with zero records.

Stage 4b Use the data in the $b^{1/5}$ records formed in stage 4a along the bottom row in each submesh as a histogram. This gives the number of elements with the correct top $(i-1)h$ bits and the $b^{1/5}$ different possible next bits. Compute the histogram for the whole mesh using the algorithm given in Jang, Park and Prasanna [7], starting from the $b^{1/5}$ values in each submesh of size $5b/\log b \times 5b/\log b$.

Stage 4c Perform a prefix sum on the $b^{1/5}$ resulting values using Lemma 1 in Jang, Park and Prasanna [7]. For prefix sums p_i, where $1 \leq i \leq b^{1/5}$ and $p_0 = 0$, the kth element will be in the group j where $p_{j-1} < k \leq p_j$. Let the new k be $k - p_{j-1}$. Then broadcast j and the new k to all the processors. In each submesh, the jth processor along the bottom row will hold the record which contains the next h bits of the kth word. In other words, the next h bits of the kth word are the binary representation of j.

End Selectword

Proving the correctness of the Selectword procedure is relatively straightforward, except for showing that the records formed in Stage 2 are stored in such a way that they can be sorted in Stage 3. The following lemma provides this fact.

Lemma 4.1 *After Stage 2, each processor will hold at most two records, one with a number field greater than $5b/\log b$ and one with a number field less than $5b/\log b$.*

Proof: Obviously, once a processor forms a record with a number field less than $5b/\log b$, it becomes inactive and never gets another record. For the records with number fields greater than $5b/\log b$, a simple proof by induction shows that at any step i, at most i records of this type will be stored in any active group, and they will be stored at the highest processors in the group. The fact that each active group has $\geq 5b/\log b$ processors implies that these records will be stored at most 1 per processor. □

Now we analyze the time of the procedure Selectword. Stage 1 is a sort which takes $\Theta(b/\log b)$ time. Stage 2 performs $5b/\log b$ steps, each of which takes constant time. Stage 3 is another sort which takes $\Theta(b/\log b)$ time. Stage 4 perform $5b/\log b$ steps, each of which involve a histogram procedure and a prefix sum. The histogram is over $b^{1/5}$ elements and starts with a histogram in each submesh of size $5b/\log b \times 5b/\log b$, so it takes $O(\max\{\log^* n - \log^* b, 1\})$ time in the cross-over model. The prefix sum is over $b^{1/5}$ numbers of at most b bits each. This takes constant time by Lemma 1 from Jang, Park and Prasanna [7]. Therefore Stage 4 takes a total of $\Theta((b/\log b)\max\{\log^* n - \log^* b, 1\})$ time.

The analysis above proves the following theorem.

Theorem 4.1 *One can select the kth of n b-bit words in $\Theta((b/\log b)\max\{\log^* n - \log^* b, 1\})$ time on a $\sqrt{n} \times \sqrt{n}$ rmesh in the cross-over model.*

5 Average Case Selection

Here we show how to find the kth element of a set of n elements stored one per processor on a $\sqrt{n} \times \sqrt{n}$ rmesh, in which all the possible orderings of elements are equally likely. In this section, we define "high probability" as probability $\geq 1 - 1/n$. First we present some useful lemmas.

Lemma 5.1 *Given a sorted sample of r elements chosen randomly from s ordered elements, the probability that the kth of the s elements is not*

1. *below element $\log^5 s$ in the random sample if $k \leq (s/r)\log^3 s$, or*
2. *between element $rk/s - \sqrt{rk/s}\log s$ and element $rk/s + \sqrt{rk/s}\log s$ in the random sample if $k > (s/r)\log^3 s$*

is $\leq s^{-3}$.

Proof: The probability of a randomly chosen element of s being equal to or below the kth is k/s. Thus the number of items in our random sample below the kth element will behave like the probability distribution $X = B(r, k/s)$. If $rk/s \leq \log^3 s$, then by Chernoff's bound, $P(X > \log^5 s) \leq 2^{-\log^5 s} \leq s^{-3}$. If $rk/s > \log^3 s$, then by Chernoff's bound $P(X < (1 - \sqrt{s/rk}\log s)rk/s) \leq e^{-(log^2 s)/2} \leq s^{-3}/2$, and $P(X > (1 + \sqrt{s/rk}\log s)rk/s) \leq e^{-(log^2 s)/3} \leq s^{-3}/2$. □

Lemma 5.2 (Clarkson, [3]) *Let S be a set of elements of size s, and let R be a sorted random sample of S of size r. Then the probability that over $O(s \log s/r)$ elements of S fall between any two adjacent elements of R is $\leq s^{-3}$*

Proof: Consider all pairs of points in R that define an interval in which $\geq \alpha s$ elements of S fall. For each of these intervals the probability of no other points in R falling in that interval is $(1-\alpha)^{r-2}$. Then the probability that any of these intervals are defined by two adjacent points of R is less than $O(r^2)(1-\alpha)^{r-2}$. By setting $\alpha = 5(\log s)/(r-2)$, the lemma follows. □

Lemma 5.3 *Given s elements, for large enough k, if we select each element with probability $s^{-(1+b)}$, then the probability that k/b elements will be chosen is $\leq s^{-k}$.*

Proof: The probability of choosing k/b elements is $\leq \binom{s}{k/b} s^{-(k/b)(1+b)} \leq s^{-k}$. □

The algorithm works as follows. The input is a set S of at most n^α, $.5 \leq \alpha \leq 1$ ordered items distributed at most one per processor on a $\sqrt{n} \times \sqrt{n}$ rmesh, with each row having $O(n^{\alpha-.5})$ items, and any order of the items equally likely. The AverageSelect procedure finds, with high probability, a reduction of the selection problem on its input to either

1. if $\alpha > .6$, an equivalently constrained selection problem with α replaced by $\alpha - .1$, or
2. if $\alpha \leq .6$ an equivalently constrained selection problem with α replaced by $\alpha - .1$ but with at most a constant number of items per row.

Then it recursively calls itself until $\alpha \leq .5$ (at most five times), at which point it simply sorts the remaining items and outputs the correct answer.

Procedure AverageSelect(S, k)

Stage 1 Count the items in S using bit-addition. Let s be the number of items, and let $\alpha = \log s / \log n$. If $k < 0$ or $k \geq s$, then revert to the deterministic selection algorithm of the previous section. (Note that this condition can only occur on recursive calls to AverageSelect.)

Stage 2 If $\alpha \leq .5$, then there should be at most a constant number of items per row. If so, then sort these items, output item k, and exit. If not, revert to the deterministic selection algorithm of the previous section.

Stage 3 Have each processor with an item in S choose its item to be in the random sample with probability $n^{.3-\alpha}$. By lemma 5.3, with high probability at most a constant number of items will be chosen in each row. If not, revert to the deterministic selection algorithm of the previous section. If so, then compress, count, and sort these items. Let r be the number of chosen items. By Chernoff's bound, $r = O(n^{.3})$ with high probability.

Stage 4 In this stage, we first broadcast the low filter, and then broadcast the high filter. If $k \leq (s/r)\log^3 s$, then

1. broadcast $-\infty$,

2. if $\log^5 s \leq r-1$ broadcast the value at this rank in the random sample, else broadcast $+\infty$.

else

1. If $\lfloor rk/s - \sqrt{rk/s}\log s \rfloor \geq 0$, then broadcast the value at this rank in the random sample, else broadcast $-\infty$,
2. If $\lceil rk/s + \sqrt{rk/s}\log s \rceil \leq r-1$, broadcast the value at this rank in the random sample, else broadcast $+\infty$.

Stage 5 Count the number of items below the low approximation using bit-addition, and let b be this sum. Let S' be the set of items between the two approximations.

Stage 6 Call AverageSelect$(S', k-b)$.

End AverageSelect

By lemma 5.1, the median is guaranteed to be between the two approximations with high probability. Also, since $k < s$ and $s \leq n$, we see that the number of items between the approximations is at most $2\sqrt{r}\log s \leq O(n^{.15}\log n)$. By lemma 5.2, with high probability there will be at most $O(n^{\alpha-.3}\log n)$ items between any two consecutive items in the random sample, and thus with high probability there will be at most $O(n^{\alpha-.15}\log^2 n) \leq O(n^{\alpha-.1})$ items between the two approximations.

Since the original items were ordered arbitrarily among the processors they occupied, the items between the approximations will be too, and it is not too hard to check using Chernoff bounds and lemma 5.3 that the conditions on the number of items per row are also satisfied.

The procedure and analysis above proves the following theorem.

Theorem 5.1 *Given a totally ordered set of n items which are stored one per processor on an rmesh, and in which all possible orderings are equally likely, then with high probability, the time required to select the kth item is the time required to perform bit-addition.*

The $\Theta(\log\log n)$ algorithm for bit-addition thus implies a $\Theta(\log\log n)$ time expected time algorithm for selection in the non-cross-over model. The $\Theta(\log^* n)$ time algorithm for bit-addition implies a $\Theta(\log^* n)$ expected time algorithm for selection in the cross-over model.

6 Lower Bound

Here we prove a lower bound of $\Omega(\log\log n)$ time on finding the maximum of n elements on an n processor rmesh. This implies an equivalent lower bound on the selection problem. We mention that Valiant [22] showed that any comparison-based algorithm for finding the maximum requires $\Omega(\log\log n)$ time. The lower bound presented here places no restrictions on the instruction sets of the processors. We also mention that the maximum of n elements can be found in $\Theta(\log\log n)$ time on an n processor rmesh, so the lower bound is tight.

Theorem 6.1 *Given n elements distributed one per processor on an n processor rmesh, finding the maximum of these elements requires $\Omega(\log\log n)$ time on the cross-over model.*

To prove this theorem we apply the techniques and analysis of Fich, Meyer auf der Heide, Ragde and Wigderson [6], in which the equivalent lower bound is proven for the CRCW PRAM. This lower bound makes no restrictions on the instruction sets of the processors, and is simply a lower bound on the number of communication steps required to find the maximum. The proof is actually simpler for the rmesh because one doesn't have to deal with infinite memory, where processors might have read and write functions with infinite ranges. A processor in the rmesh only has a constant number of possible actions at each step, namely, which local switches to set, and whether to write or not. Because of this, the Ramsey Theoretic arguments in the PRAM proof can all be replaced by simple Pigeonhole Principle arguments. The proof proceeds as follows.

Let MAX be any algorithm which finds the maximum of n inputs on a $\sqrt{n} \times \sqrt{n}$ rmesh. We will show that MAX requires $\Omega(\log \log n)$ steps by an adversary argument. At each step, the adversary will fix the values of certain inputs and maintain a set of allowed values for the non-fixed inputs such that each processor will know only one non-fixed input. We make this more precise as follows.

Consider step t of MAX. Let $V_t \subseteq \{1, \ldots, n\}$ be the set of indices of inputs which could still be the maximum. These are the *live* inputs. Let $S_t \subseteq N = \{1, 2, \ldots\}$ be the set of possible values for the live inputs, restricted by the adversary. Let $F_t = \{f_i | i \in \{1, \ldots, n\} - V_t\}$ be the adversary's assignment of values to fixed variables.

At the start of the algorithm, $V_0 = \{1, \ldots, n\}$, $S_0 = N$ and $F_0 = \emptyset$. Now we state the main lemma.

Lemma 6.1 *We can construct an adversary such that after step t of MAX, the following properties hold.*

1. *$V_t \subseteq V_{t-1}$, and $|V_t| \geq |V_{t-1}|^2/(|V_{t-1}| + 2n)$.*
2. *Each processor knows only one variable in V_t.*
3. *$S_t \subseteq S_{t-1}$ and S_t is infinite.*
4. *$F_{t-1} \subseteq F_t \subseteq N - S_t$.*

First we show why this lemma implies the theorem. For MAX to be finished in T steps, it must be the case that $V_T = 1$. Then by property 1 of the lemma,

$$1 = |V_T| \geq \frac{|V_{T-1}|^2}{3n},$$

since $V_t \leq n$ for any t. By recursively applying property 1 of the lemma, we get

$$1 \geq \frac{|V_0|^{2^T}}{(3n)^{2^T-1}} = \frac{n^{2^T}}{(3n)^{2^T-1}} = \frac{n}{3^{2^T-1}}.$$

Then $3^{2^T-1} \geq n$, which implies $T = \Omega(\log \log n)$.

Before we prove the lemma, we state the Pigeonhole Principle (which is a special case of the Ramsey Theory lemma used in [6]) and Turán's Theorem.

Pigeonhole Principle Let $f : W \to D$ be any function defined on an infinite domain W and finite range D. Then there is an infinite subset $W' \subseteq W$ such that $f|_{W'}$ is constant.

Turán's Theorem Given a finite graph $G(V, E)$, there exists an independent set of vertices of size $|V|^2/(|V| + 2|E|)$.

Proof: (*of Lemma 6.1*) We prove the lemma by induction. Assume the lemma holds for all steps before step t. Then in step t, a processor P_i either writes or does not write depending on the live input it knows, and connects to its neighbors depending on this same input. Call the write functions and connect functions w_i^t and c_i^t respectively. Each is a function from an infinite domain S_{t-1} to a constant range. By applying the Pigeonhole Principle to each write function and connect function in turn, for all i from 1 to n, the adversary can restrict the allowed inputs such that all write functions and connect functions are constant. Let $S' \subseteq S_{t-1}$ be this restriction. This fixes the communication pattern on the rmesh. Each processor connected to a bus then learns only the variable which was known by the processor which wrote to that bus. Note that if more than one processor wrote to the bus, they must have written the same value, so no more information can be gained then by a single processor writing to the bus.

At this point each processor knows at most two inputs in V_{t-1}. Now consider the graph $G(V_{t-1}, E)$ with $(i, j) \in E$ if and only if a processor knows both input i and input j. We know $|E| \leq n$, and thus by Turán's theorem, we can construct an independent set of size $|V_{t-1}|^2/(|V_{t-1}| + 2n)$. Let $V_t \subseteq V_{t-1}$ be this independent set. This satisfies properties 1 and 2. Now for each $i \in V_{t-1} - V_t$, let f_i be the smallest value $s \in S'$, and add this to F_{t-1} to get F_t. Also let $S_t = S' - \{s\}$. This satisfies properties 3 and 4, and thus the lemma holds. □

Plaxton [18] gives a lower bound of $\Omega((n/p) \log \log p + \log p)$ for selection on a fixed network of p processors which satisfies a particular low expansion property. His proof assumes that each processor can only compare or copy keys, so it is not as general as the proof above. Also, the $\log p$ term does not apply to the rmesh because that term comes from the diameter of the network, which for the rmesh is constant. Thus there is still a large separation between the upper and lower bounds on the selection problem. Narrowing this gap is an open problem.

References

[1] P. Beame and J. Hastad. Optimal bounds for decision problems on the CRCW PRAM. *J. Assoc. Comput. Mach.*, 36(3):643–670, July 1989.

[2] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *J. Comput. System Sci.*, 7(4):448–461, 1972.

[3] K. L. Clarkson. New applications of random sampling in computational geometry. *Discrete Comput. Geom.*, pages 195–222, 1987.

[4] R. Cole. An optimally efficient selection algorithm. *Inform. Process. Lett.*, 26:295–299, 1988.

[5] H. ElGindy and P. Węgrowicz. Selection on the reconfigurable mesh. In *Proc. 20th Intl. Conf. on Parallel Processing*, pages III 26–33, 1991.

[6] F. E. Fich, F. Meyer auf der Heide, P. Ragde, and A. Wigderson. One, two, three ...infinity: Lower bounds for parallel computation. In *Proc. 17th Symp. on Theory of Computing*, pages 48–58, 1985.

[7] J. Jang, H. Park, and V. K. Prasanna. A fast algorithm for computing histogram on reconfigurable mesh. Technical Report IRIS#290, Univerisity of Southern California, 1992.

[8] J. Jang and V. K. Prasanna. An optimal sorting algorithm on reconfigurable mesh. Technical Report IRIS#277, Univerisity of Southern California, 1991.

[9] H. Li and Q. F. Stout. *Reconfigurable Massively Parallel Computers.* Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[10] H. Li and Q. F. Stout. Reconfigurable SIMD massively parallel computers. *Proceedings of the IEEE*, pages 429–443, 1991.

[11] P. D. MacKenzie. A separation between reconfigurable mesh models. in preparation.

[12] N. Megiddo. Linear time algorithm for linear programming in R^3 and related problems. *SIAM J. Comput.*, 12(4):759–776, 1983.

[13] R. Miller, V. K. Prasanna-Kumar, D. I. Reisis, and Q. F. Stout. Data movement operations and applications on reconfigurable VLSI arrays. In *Proc. 17th Intl. Conf. on Parallel Processing*, volume 1, pages 205–208, 1988.

[14] R. Miller, V. K. Prasanna-Kumar, D. I. Reisis, and Q. F. Stout. Parallel computations on reconfigurable meshes. *IEEE Trans. Comput.*, 41, 1992. to appear.

[15] J. I. Munro and M. S. Paterson. Selection and sorting with limited storage. *Theor. Comput. Sci.*, 12:315–323, 1980.

[16] K. Nakano, T. Masuzawa, and N. Tokura. A sub-logarithmic time sorting algorithm on a reconfigurable array. *IEICE Transactions*, E74(11):3894–3901, 1991.

[17] M. Nigam and S. Sahni. Sorting n numbers on $n \times n$ reconfigurable meshes with buses. Technical Report TR-92-04, Univerisity of Florida, 1992.

[18] C. G. Plaxton. On the network complexity of selection. In *Proc. 30th Symp. on Found. of Comp. Sci.*, pages 396–401, 1989.

[19] V. K. Prasanna-Kumar and C. S. Raghavendra. Array processors with multiple broadcasting. *J. Parallel and Distributed Comput.*, 4:173–190, 1987.

[20] Q. F. Stout. Mesh-connected computers with broadcasting. *IEEE Trans. Comput.*, 32(9):826–830, September 1983.

[21] C. D. Thompson and H. T. Kung. Sorting on a mesh connected parallel computer. *Comm. ACM*, 20(4):263–271, 1977.

[22] L. G. Valiant. Parallelism in comparison problems. *SIAM J. Comput.*, 4:348–355, 1975.