Randomized Selection on the Mesh-connected Processor Array

Danny Krizanc
Lata Narayanan
Department of Computer Science
University of Rochester
Rochester, NY 14627

Abstract

We show that selection on an input of size \( N = n^2 \) can be performed by a \( N \) node mesh-connected processor array in \( 2n + o(n) \) steps, with high probability. The best previously known algorithm for this involved sorting, and required \( 3n + o(n) \) steps.

Keywords: Randomized algorithms, parallel algorithms, selection, mesh-connected array.

1 Introduction

Selection, the problem of selecting the \( k \)th largest (or smallest) out of \( N \) elements, is an important and well-studied problem in computer science. In the sequential setting, Blum et. al. [BFP+72] were the first to show that selection could be done in linear time. Valiant [Val75] introduced the parallel computation tree (PCT) model to study the parallel complexity of selection and other comparison-based problems. He showed an \( \Omega(n/p + \log \log n / \log (1 + p/n)) \) lower bound for finding the maximum (and therefore for selection in general) of \( N \) elements using \( p \) processors in this model. Azar and Pippenger [AP90] showed that the above bound was tight for selection. A number of researchers have given optimal algorithms for selection in PRAM models of computation including Vishkin [Vis87] and Cole [Col88].

In the fixed connection network model we view the computation as taking place on a graph where the nodes correspond to processors and the edges correspond to communication channels. On a \( p \) processor network selection may be described as follows: initially each processor contains \( N/p \) of the input elements and at the end of the computation a designated processor contains the \( k \)th largest element. Recently, Plaxton [Pla89b] proved an \( \Omega((n/p) \log \log p + \log p) \) lower bound for selection on
networks that satisfy a particular low expansion property. The class of networks satisfying this property includes such common network families as trees, multi-dimensional meshes, hypercubes, butterfly and shuffle-exchange networks. The algorithms in [Pla89a] and [CP90] imply that the selection problem can be solved in 

\[ O(\min \{ (n/p) \log p (\log \log p)^2, (n/p) \log \log p + \log^2 p \log (n/p) \} ) \]

on the \( p \)-processor hypercube, which matches the lower bound for \( n \geq p \log^2 p \).

The use of randomization in selection algorithms was pioneered by Floyd and Rivest [FR75]. The basic idea of their algorithm is to choose a random sample of the elements which is used to find elements close to and bracketing the \( k \)th largest with high probability. These bracketing elements are then used to eliminate a large fraction of the elements from consideration and selection is performed recursively on those remaining. The resulting algorithm runs in expected linear time. Using the same technique, Reishuk [Rei85] and Meggido [Meg82], give algorithms for selection on a \( N \) processor PCT requiring \( O(1) \) steps with high probability. Recently, Rajasekaran [Raj90] applied these ideas to achieve optimal time selection on a hypercube network.

The mesh-connected processor array (or mesh) has been the object of a great deal of study due to its simplicity which makes it ideal for VLSI implementation. In this paper we are interested in the problem of selecting the \( k \)th largest of \( N = n^2 \) elements on a \( n \times n \) mesh where initially each processors contains exactly 1 element. Due to the mesh's relatively large diameter, we consider the case where the selected element must reach the middle processor (i.e., the one labeled \( [\frac{n}{2}], [\frac{n}{2}] \)) in the labeling described below). In a restricted model of the mesh, Schnorr and Shamir [SS86] gave an \( 3n + o(n) \) algorithm for sorting into snake-like row major order. This immediately implies an \( O(n) \) time algorithm for selection in general and a \( 3n + o(n) \) algorithm for selecting the median (the element of rank \( \lfloor \frac{n}{2} \rfloor \)) at the middle processor. It is clear that there exist inputs on which any general selection algorithm must take \( n \) steps. The results of Kunde [Kun89] imply a lower bound of \( 2n - o(n) \) steps for selecting the median at the middle processor in a model encompassing the upper bound model of Schnorr and Shamir. In this paper, we give a randomized algorithm for selection at the middle processor which runs in \( 2n + o(n) \) steps, with high probability. The model used for this result is the commonly used, less restricted model of the mesh where processors can store \( O(1) \) elements between steps. The best previous result for this problem was the upper bound based on the Schnorr and Shamir sorting result which required greater than \( 3n \) steps. (Recently we have extended the techniques of this paper to achieve a \( 1.5n + o(n) \) bound.)

The rest of the paper is organized in the following way. The next section gives some preliminaries, including details of the model of the mesh that we use. Section 3 gives our randomized algorithm for the general problem of selection. We close with some discussion of future work and open problems.
2 Preliminaries

The mesh network contains $N = n^2$ nodes and is $G = (V, E)$ where $V = \{(x, y) \mid x, y \in \langle n \rangle\}$ and $E = \{((x, y), (x, y+1)) \mid x \in \langle n \rangle, y \in \langle n-1 \rangle \} \cup \{((x, y), (x+1, y)) \mid x \in \langle n-1 \rangle, y \in \langle n \rangle\}$. The degree of the mesh is 4 and the diameter is $2n - 2$. We refer to the processor labeled $([n, n], [n, n])$ as the middle processor. The torus is a mesh with the wrap-around edges put in. The diameter of the torus is $n - 1$. The $d$-dimensional mesh and torus are the logical extensions of the two-dimensional versions.

The selection problem on the mesh is stated as follows. Given $N = n^2$ elements placed one per processor on a mesh, an integer $1 \leq k \leq N$, and a specified processor $(i, j)$, move the $k$-th largest element to the processor $(i, j)$. In what follows we will consider only the case where the specified processor is the middle processor. It is generally straightforward how to modify the algorithm for the case of another designated processor.

We use the following model of the mesh: At any given step, each processor can communicate with all of its neighboring processors, and can send and receive a packet along each edge. Processors can also store packets in their local queues, which are of bounded size. They can perform simple operations on the queue in each step. (For our applications, we need a simple priority queue.)

3 Selection on the Mesh

We now describe an algorithm to find $k$-th largest of $N = n^2$ elements in $2n + o(n)$ steps, using constant sized queues with high probability. To motivate our final algorithm, we first present an algorithm that is simple, but inefficient, and then show how to speed it up to the claimed time bounds, as well as reduce the queue size.

3.1 Preliminary Facts

In the following material, when we use the term "with high probability", we mean with probability greater than $1 - n^{-c}$, for some constant $c$.

By the middle diamond we refer to the diamond shaped area in the mesh consisting of the nodes which are not more than $n/2$ away from the center processor.

In our routing algorithms we will use the following two queuing disciplines.

$\mathcal{Q}$: Any queuing discipline with the property that "packets that have already started to move have higher precedence over those that have not". On the linear array, this implies that when a packet starts to move, it will flow at constant speed and never be delayed again.

$\mathcal{Q}':$ Packets that have farther to go (from their current location) have higher priority.
Fact 1: Consider an n-node linear array. The edges are assumed to be bidirectional. Suppose that n packets are distributed arbitrarily in the nodes of the graph, each having a distinct destination. Then, using queuing discipline $Q'$, a packet starting at node $i$ will reach its destination within $b(i)$ steps at most, where $b(i)$ denotes the distance between $i$ and the boundary in the direction the packet is moving (i.e., $n - i$ or $i - 1$ depending on whether the packet is moving from left to right or from right to left). If, furthermore, $L$ is the maximum over all origin-destination distances, each packet will reach its destination in at most $L$ steps. (See [KRT88] for a proof.)

We shall need the following probabilistic bounds for the analysis of our algorithms.

Fact 2: (Bernstein-Chernoff bound) Let $S_{N,p}$ be a random variable having binomial distribution with parameters $N$ and $p$ (i.e., $S_{N,p}$ is the sum of $N$ independent Bernoulli random variables each with mean $p$). Then, for $m \geq Np$

$$P(S_{N,p} \geq m) \leq \exp[-m \ln(Np) - m \ln m + m - Np].$$

3.2 A 3.5$n + o(n)$ Algorithm with Logarithmic Queues

Theorem 1 Given $n^2$ elements located at the processors of a mesh, the $k$-th largest element can be routed to the center processor in $3.5n + 26n + o(n)$ steps, where $\delta$ is a constant such that $0 < \delta < 1$, and the size of the queues does not exceed $c \log n$ for some constant $c$ with high probability.

Proof. In [FR75] a probabilistic sequential algorithm of small average running time is described. Reischuk [Rei85] and Megiddo [Meg82] describe parallel algorithms in the parallel comparison tree model which work in expected constant time. Using some of these ideas, we will construct a parallel algorithm on the mesh and prove that it has the properties stated above.

Let $X = \{x_1, x_2, \ldots, x_{n^2}\}$ be the given set. As in [Rei85] and [Meg82], we choose a sample of size $s$, and find two elements $u$ and $v$ in the sample that bracket the $k$-th largest element in the sample. Define

$$A = \{x \in X | x < u\}$$
$$B = \{x \in X | u \leq x \leq v\}$$
$$C = \{x \in X | v < x\}$$

If $|A| < k$ and $|A| + |B| > k$, then the $k$-th largest element lies in the set $B$. We claim that for a good choice of $s$, $u$ and $v$, the $k$-th largest will indeed lie in $B$ with high probability, and we will show efficient ways of finding the elements $u$ and $v$ from the sample, and finding the $k$-th largest from the set $B$.

Let $N = n^2$. Let $1/3 < \epsilon < 1/2$ and choose $s = O(N^{1/2+\epsilon/2}) = O(n^{1+\epsilon})$. Let $f = N^\epsilon$. Define

$$rank(u) = t_1 = max\{0, k(n + 1)/(n^2 + 1) - f\}$$
$$rank(v) = t_2 = min\{n + 1, k(n + 1)/(n^2 + 1) + f\}$$
Let $E_0$ be the event that the $k$-th largest element does not lie in the set $B$ and let $E_1$ be the event that $|B| > 4(f + 1)(N + 1)/(s + 1)$. Using lemma 2 from [Rei85], which we reproduce below, we are able to show that both $E_0$ and $E_1$ are unlikely events.

Lemma 1 [Rei85]

i) $P(E_0) \leq \exp(-f^2/(4(s + 1)) + O(\ln N))$

ii) $P(E_1) \leq \exp(-f^2/(4(s + 1)) + O(\ln N))$

For our choice of parameters, it is clear that the $k$-th largest element lies in the set $B$ with high probability and $|B| = O(n^{1+\epsilon})$. Let $s'$ be $n^{1+2\epsilon}$. Define "sub-mesh $M$" to be the square submesh of size $s'$ centered at the center processor.

Algorithm 1

1. Select a sample of size $s$. Each processor selects itself with probability $1/n^{1-\epsilon}$. With high probability, the size of the sample is $O(n^{1+\epsilon})$. Each packet that is at a selected processor picks a random destination in the same quadrant in sub-mesh $M$, and begins to move towards it.

2. Move all the remaining packets into the middle diamond.

3. Sort sub-mesh $M$ using a standard algorithm for sorting into snake-like order. Find elements $u$ and $v$ defined above.

4. Broadcast the values of $u$ and $v$ to all the processors in the middle diamond.

5. Collect the information about the sizes of sets $A$, $B$ and $C$ defined above, in the center processor.

6. Packets with values in the set $B$ choose a random destination in the sub-mesh $M$. Route these packets to their chosen destinations.

7. If the $k$-th largest element does indeed lie in $B$, and if $|B| \leq n^{1+\epsilon}$, sort $B$ and find the element of appropriate rank, which is the $k$-th largest of the set $X$, and we are done. Otherwise, broadcast a message to the whole mesh, and restart. Sort the input configuration, so that the $k$-th largest reaches the center processor.

end

The correctness of Algorithm 1 follows from [Rei85]. We will prove that it works in the claimed number of steps and with logarithmic queues.
Lemma 2 Step 1 can be done in $n$ steps with constant sized queues, with high probability.

Proof. With high probability, there are $O(n^{1+\varepsilon})$ packets in the sample. (In the case that there are too many or too few packets, we will consider the algorithm to have failed, and simply sort in order to find the selected element.) There are $n^{1+2\varepsilon}$ destinations, each sample packet chooses a destination with probability $1/n^{1+2\varepsilon}$. Using Fact 2, we can prove that there exists a constant $c > 1/e$ such that with high probability, at most $c$ packets choose the same destination. We will route the sample using a greedy algorithm, where each packet travels to the right row and then to the right column. Note that we only need to analyze the sample packets in one quadrant. Clearly no collisions occur while the packets are traveling up the column to the right row. The queue size can increase only when two packets enter the queue for an edge at the same time, and this means at least one of the packets must have “turned” at the node. Since each packet turns only once, the expected number of packets that turn at any node, over all time is less than one. Consider any column and the half of it that lies in the quadrant. With high probability, the number of sample packets in the column does not exceed $n^{2\varepsilon}$. Each of these packets picks a row to turn into with probability $1/n^{1/2+\varepsilon}$. By Fact 2, $\exists b > 1$ such that with high probability, the number of packets turning into a given node is not more than $b$.

Lemma 3 Overlapping the packets can be done in $n/2$ steps.

Proof. We would like each node in the middle diamond to end up with two packets. Since our model allows constant sized queues, this is legal. To do the overlap, each packet simply marches in the horizontal direction to the boundary of the middle diamond in its own quadrant and then in the vertical direction towards the center row and goes as far as it can go till it either hits the center row or can't go further without hitting a node which has two packets already. It is easy to see that this operation can be completed in $n/2$ steps.

Lemma 4 (i) Sorting the sample takes $O(n^{1/2+\varepsilon})$ steps.

(ii) Step 3 (broadcasting the values of $u$ and $v$) takes $n/2 + 1$ steps.

(iii) Collecting information about sizes of the sets $A$, $B$, and $C$ takes $n/2 + 2$ steps.

(iv) Sorting set $B$ takes $O(n^{1/2+\varepsilon})$ steps.

Proof. (ii) follows from the fact that all the packets are in the middle diamond. We need to send two packets (the values of $u$ and $v$) to each processor in the inner diamond. These packets travel down along all the center column and out along row edges. The two packets can be routed in a pipelined fashion, thus taking $n/2 + 1$ steps. The proof of (iii) is similar. (i) and (iv) are consequences of the sorting algorithm in [SS86].
Lemma 5  Routing the set $B$ can be done in time $n + 2\delta n + o(n)$ using logarithmic sized queues with high probability.

Proof. We adapt the algorithm described in [KRT88] for permutation routing. To motivate our final algorithm better, we first present an algorithm that takes $n$ steps and requires logarithmic queues. Then we show how to modify this algorithm in order to get our claimed bounds.

Algorithm 2: ROUTE

Divide the mesh up into $1/\delta$ slices containing $\delta n$ rows each. If packet $(i,j)$ wants to go to $(r,s)$ eventually:

Phase 1: Choose a random row in its own slice, say $k$ and go to $(k,j)$.

Phase 2: Go to $(k,s)$ (correct the column).

Phase 3: Go to $(r,s)$ (correct the row).

Note that we could have two packets at each node to start with and all the nodes do not necessarily participate. Also our final routing pattern is not a permutation, that is, as many as $\log n$ packets may want to go to the same destination.

Queue size analysis: At Phase 1, processor $(k,j)$ can receive two packets from $\delta n$ processors (the ones at the same strip and column as $(k,j)$), each with probability $1/(\delta n)$. Let $E_m$ be the event that more than $m$ packets will be stored at $(k,j)$ at the end of Phase 1. Then, using Fact 2, $P(E_m) = B(m; 2\delta n, \frac{1}{\delta n}) \leq \exp(m \ln 2 - m \ln m + m - 2) = \exp(cm - m \ln m - 2)$. Therefore, the probability that any one processor will have more than $m$ packets at the end of Phase 1, is less than $n^2 \exp(cm - m \ln m - 2) = \exp(2\ln n - m \ln m + cm - 2)$. By choosing $m = \Theta(\log n)$ we can make this probability smaller than the inverse of some polynomial in $n$.

Consider a given node $(k,s)$ at the end of Phase 2. With high probability, not more than $n^{1/2 + c}$ packets will choose destinations in column $s$. Each of these picks node $(k,s)$ with probability less than $1/\delta n$. Using Fact 2, we can prove that with high probability, at most $O(\log n)$ packets accumulate at any processor. Since the maximum of the queue sizes at the end of Phase 1 and Phase 2 is an upper bound on the sizes of queues at any step during the algorithm, we have proved that with high probability, the queue size grows to at most $O(\log n)$.

Routing Time Analysis: Phase 1 can be accomplished in $2\delta n$ steps, simply by making two passes to account for two packets at every node. In Phase 2, we use queuing discipline $Q$, and using a similar analysis to [RT90], we can show that the probability that the delay is more than $n^c$ is at most $\exp(-Cn^{2c-1})$. For Phase 3,
we use queuing discipline $Q'$. Unfortunately, the packets are distributed through the entire length of the column and this implies that Phase 3 takes as many as $n/2$ steps.

For each $0 < \delta < 1$, we can route the set $B$ in $n + 2\delta n + o(n)$ steps using $O(\log n)$ size queues.

The total time required is therefore $n + n/2 + n/2 + n + 2\delta n + o(n) = 3.5n + 2\delta n + o(n)$ steps, for $0 < \delta < 1$.

### 3.3 Coalescing the phases

In this section we show that some of the steps in our algorithm for selection can be combined, thus significantly lowering the number of time steps.

**Lemma 6** Collecting information about the sizes of sets $A$, $B$ and $C$, as well as routing of the set $B$ can be done simultaneously.

**Proof.** The packets belonging to set $B$ take lower precedence. Then each such packet can be delayed at most 3 time units (at the same place) because of this.

**Lemma 7** The set $B$ can be routed in $n/2 + o(n)$ steps.

**Proof.** We notice here that the actual distance traveled by any packet is always less than $n/2 + \delta n + n^{1/2+\epsilon}$. We consider the effect on the running time if the three phases are coalesced. The only possible conflict is between packets doing their Phase 3 and Phase 1. In such a case, the packet doing its Phase 1 is given preference. If a packet $q$ doing its Phase 3 contends for an edge with a packet that is doing its Phase 1, then it needs to go a maximum of $n/2$ steps to get to its destination. Since after $2\delta n$ steps, $q$ will only have to contend with packets doing their Phase 3, it will reach its destination in $n/2 + 2\delta n$ steps. No other packets suffer additional delays due to coalescing the phases.

Packets now travel a maximum of $2\delta n + n/2 + n^{1/2+\epsilon}$, but if $\delta$ is a constant, this is not good enough. To overcome this, we try and ensure that all packets are less than $n/2 - 2\delta n$ away from their final random destinations inside the sub-mesh $M$. For these purposes we define $I$ to be the inner diamond, that is all the nodes that are less than $n/2 - c\delta n$ away from the final destination, where $c > 4$ is a constant. While performing the overlap operation, we will try to get all the packets into the inner diamond in such a way that there are a maximum of 4 packets per node. In Phase 2, using an analysis similar to before, we can show that every packet has $O(\log n)$ delay with high probability. Using Fact 1, we can show that every packet can finish its Phase 3 in $n/2 - c\delta n$ steps. By coalescing the phases, we can show that the total time we take is $4\delta n + n/2 - c\delta n + o(n)$.

Using Fact 2, and an analysis similar to that used in the proof of lemma 5, we can bound the queue size in the different phases of this algorithm to be $\Theta(\log n)$, with high probability.
We have shown that there is a randomized oblivious algorithm to route the set \( B \) that can be realized in \( n/2 + o(n) \) steps and uses queues of size \( \Theta(\log n) \) with high probability. It remains to show that the new overlap operation can be performed in conjunction with the routing of the sample in a total of \( n \) steps. We prove the following lemma.

Lemma 8 \textit{Routing the sample can be done together with overlapping packets into the inner diamond in} \( n \) \textit{steps.}

\textbf{Proof.} We want to get the sample in sub-mesh \( M \), and perform the overlap, in \( n \) steps. We will route the sample using a greedy algorithm, where each packet travels to the right row and then to the right column.

We can consider the new overlap as being composed of two phases. First, we do the overlap operation as defined before. Then every packet that is outside the inner diamond simply moves in along column edges. Since each node could have had 2 packets at it, this could take \( 2c_0n \) steps. This approach does not work for the \( 6c_0n \) rows in the middle. We call these packets \textit{special} and deal with them separately. The remaining packets will simply start the two-phase operation described above after waiting out \( n/2 - 2c_0n \) steps. Clearly, the overlapping packets do not interfere with each other. After \( n/2 - 2c_0n \) steps, the sample packets that are still using column rows are \( 2c_0n \) away from the center, and so no collisions occur between sample packets and non-special packets.

We will describe what the special packets in the lower quadrants should do. Packets here do not perform the two phase operation that we just described. This block of packets which are not already inside the inner diamond simply moves down \( 3c_0n \) rows, as a block and then each packet starts the two-phase operation as it would if it had started there (see fig. 2). The overlap operation for these packets takes a maximum of \( 10c_0n \) steps. By choosing \( \delta < 1/20c \), we can ensure that this takes less than \( n/2 \) steps. Since these packets can start moving after \( n/2 \) steps, clearly they do not collide with the sample packets.

Our new algorithm combines Steps 1 and 2 (and overlaps into the inner diamond rather than the middle diamond), and Steps 5 and 6 of Algorithm 1. Using Lemmas 4, 6, 7, and 8, we can show the following:

\textbf{Theorem 2} \textit{There exists an algorithm for finding the} \( k \)-th \textit{largest element out of} \( N = n^2 \) \textit{elements on an} \( n \times n \) \textit{mesh that finishes in} \( 2n + o(n) \) \textit{steps and uses queues of size} \( \Theta(\log n) \), \textit{with high probability.}

\subsection*{3.4 A Constant-size Queue Algorithm}

We will now modify the above algorithm so that it only uses constant-size queues. The only point where we may use large queues in our algorithm is in the routing of
set $B$. In order to reduce the size of the queues, we use a redistribution trick as in [RT90]. We divide the rows (and columns) up into blocks of size $\log n$.

Routing the set $B$: In Phase 1 of the routing algorithm, when a packet wants to go to its random destination within its slice, if the queue for the edge it wants to take is already full, it tries to find another location in the $\log n$ sized block the destination is in. Using Fact 2, we can show that with high probability, there exists a $d$ such that the number of packets wanting to go to any $\log n$ sized block does not exceed $d \log n$. Therefore, if each node has queues of size $d$, then the redistribution can cause a delay of $O(\log N)$.

The same trick is used at the end of Phase 2 while routing the set $B$. We already know that there is only a constant number of packets at each node at the end of Phase 3.

**Theorem 3** There exists an algorithm for finding the $k$-th largest element out of $N = n^2$ elements on an $n \times n$ mesh that finishes in $2n + o(n)$ steps and uses queues of size $O(1)$, with high probability.

4 Discussion

4.1 Higher dimensional meshes

Using efficient randomised routing algorithms for higher dimensional meshes yields algorithms for selection similar to Algorithm 2 better than any previously known (sorting-based) selection algorithms for these meshes. Details will be provided in a later version of the paper.

4.2 Tori

Clearly, this algorithm works on the torus in the same time bounds, but since the diameter of the torus is much smaller, it is possible that there are faster algorithms on an $n \times n$ torus. On the Schnorr-Shamir model of the mesh, a lower bound of $1.5n$ steps in the worst case can be proved.

4.3 More intensive overlapping

By overlapping into the diamond shaped area consisting of those processors which are less than $n/4$ away from the middle processor, we can modify the algorithm to work in $1.5n + o(n)$ steps. We believe that by overlapping into an even smaller area, it is possible to achieve a $(1.25 + \epsilon)n$ bound but this is the limit of the techniques developed in this paper. Details will be provided in a later version of the paper.
4.4 Lower bounds

We can extend this technique of packing packets into small areas close to the center processor to get more efficient algorithms for doing selection. On the other hand, lower bound arguments for selection on the mesh might be derived using the fact that packing beyond a certain point violates our restriction that we can only send one packet across an edge in a single time step. We are interested in proving lower bounds for this problem on some appropriate model of the mesh. The only known deterministic upper or lower bounds are the obvious ones (based on sorting and on distance respectively). It would be interesting to try to close the gaps between upper and lower bounds both in the deterministic and the randomized settings.

References


