Randomized Parallel Selection

Sanguthevar Rajasekaran
Department of Computer and Information Sciences
University of Pennsylvania
Philadelphia, PA 19104

ABSTRACT

We show that selection on an input of size N can be performed on a P-node hypercube ($P = N/(\log N)$) in time $O(N/P)$ with high probability, provided each node can process all the incident edges in one unit of time (this model is called the *parallel model* and has been assumed by previous researchers (e.g., [**?**])). This result is important in view of a lower bound of Plaxton that implies selection takes $\Omega((N/P)\log\log P + \log P)$ time on a P-node hypercube if each node can process only one edge at a time (this model is referred to as the *sequential model*).

1 Introduction

Given a set of N keys, and an integer k ($1 \leq k \leq N$), the problem of selection is to find the kth smallest key in the set. This important comparison problem has an elegant linear time sequential algorithm [**?**]. Optimal algorithms also exist for certain parallel models like the CRCW PRAM, the comparison tree model etc.

We are interested in solving the selection problem on the hypercube interconnection network. On any parallel model that uses P processors, one would like to know if there exists a selection algorithm that runs in time $O(N/P)$. Plaxton has proved a lowerbound of $\Omega((N/P)\log\log P + \log P)$ for the hypercube (under the assumption that each processor can process at most one incident edge per time step). An interesting open question was: 'Is there an $O(N/P)$ selection algorithm for the hypercube if each processor can handle all its incident edges in a single time step?' (The routing algorithm proposed originally by Valiant and Brebner [**?**] runs on the parallel hypercube model).

We give a randomized selection algorithm in this paper that acheives this $O(N/P)$ time bound (for the worst case input with overwhelming probability).

1.1 Model Definition

Any fixed connection network can be represented as a graph $G(V, E)$ where the vertices correspond to processing elements and the edges correspond to communication links. Examples of fixed connection networks include the hypercube, the butterfly, the shuffle exchange etc. Real computers have been built based on fixed connection models.

A hypercube of dimension n consists of $N = 2^n$ nodes (or vertices) and $n2^n$ edges. Thus each node in the hypercube can be named with an n-bit binary number. If x is any node in V, then there is a bidirectional link from x to a node y if and only if x and y (considered as binary numbers) differ in one bit position (i.e., the hamming distance between x and y is 1.) Therefore, there are exactly n edges going out of (and coming into) any vertex.

If a hypercube processor can handle only one edge at any time step, this version of the hypercube will be called the *sequential model*. Handling (or processing) an edge here means either sending or receiving a key along that edge. A hypercube model where each processor can process all its incoming and outgoing edges in a unit step is called the *parallel model* [**?**].

In this paper we assume the parallel version of the hypercube.

1.2 Previous Results

Plaxton [**?**] has presented an algorithm that runs on the sequential hypercube in time $O((N/P)\log\log P + (T_1 + T_2 \log P)\log(N/P))$ using P processors, where T_1 is the time needed for sorting N keys (located one per processor) on an N-processor hypercube, and T_2 is the time needed for broadcasting and summing on an N-node hypercube. He [**?**] has also proved a lowerbound of $\Omega((N/P)\log\log P + \log P)$ for selection. For $N \geq P\log^2 P$ the lowerbound matches the upperbound (to within a multiplicative constant). The only operations allowed on the keys are copying and comparison (for both the upperbound and the lowerbound).

Several results are known regarding selection on PRAM models. Vishkin's [**?**] algorithm is optimal with a time bound of $O(\log n \log\log n)$. Cole [**?**] has given an $N/\log N$ processor, $O(\log N)$ time CREW PRAM algorithm. His algorithm also runs in time $O((N/P)\log\log P + \log P \log^* P)$ on the EREW PRAM. It is optimal for
$N = \Omega(P \log P \log^* P)$.

Selection has also been well studied on the parallel comparison tree model. Ajtai, Kómlos, Steiger, and Szemeredi's [**?**] n processor, $O(\log\log n)$ time algorithm, Cole and Yap's [**?**] $O((\log\log n)^2)$ and n processor algorithm are some of the well known algorithms.

Randomization has been used to solve selection on a wide variety of parallel models. Meggido's [**?**] algorithm does maximal and median selection in constant time using a linear number of processors on the comparison tree model. Reischuk's [**?**] selection algorithm runs in $O(1)$ time using N comparison tree processors. Floyd and Rivest's [**?**] sequential algorithm takes $N+\min(k, N-k)+o(N)$ time. Rajasekaran and Sen [**?**] give an $O(\log N)$ time $N/\log N$ processor maximal selection algorithm for the CRCW PRAM model. All these results hold for the worst case input with high probability. The underlying idea behind all these algorithms is to sample $o(N)$ keys at random, and to eliminate keys (from future consideration) in stages.

For an extensive survey of randomized selection algorithms, see [**?**]. In this paper we present a randomized algorithm for selection (on an input of N keys) on the parallel hypercube model that uses $N/\log N$ processors (with $\log N$ keys initially residing at each processor) and runs in $O(\log N)$ time on the worst case input with high probability. We also assume that the only operations allowed on the keys are copying and comparison.

2 Preliminary Facts

Definitions. Let $B(n,p)$ stand for a binomial random variable with parameters n and p. By *high probability* we mean a probability $\geq (1-N^{-\alpha})$ for any $\alpha \geq 1$ (N being the input size). We let 'w.h.p.' stand for 'with high probability'. We say a randomized algorithm has a resource (like time, space etc.) bound of $\widetilde{O}(f(n))$ if there exists a constant c such that the amount of resource used is no more than $c\alpha f(n)$ with probability $\geq (1-n^{-\alpha})$ on any input of size n. A random variable X is said to upper bound another random variable Y if for any x, $Prob.[Y \geq x] \leq Prob.[X \geq x]$.

The following results related to packet routing and sorting on the hypercube will be used in our algorithm.

The problem of *routing* is this: Given a network and a set of packets of information, a packet being an <origin,destination> pair. To start with the packets are placed in their origins. Only one packet can be sent along any edge at any time. The problem is to send all the packets to their correct destinations as quickly as possible. The restriction of

this problem where exactly one packet originates from any node and exactly one packet is destined for any node is called *permutation routing*. A routing algorithm is given by specifying a path that each packet should take, together with a *queueing discpline*, i.e., a rule for resolving contentions for the same edge. The *run time* of a routing algorithm is the time needed for the last packet to reach its destination, and the *queue size* is defined to be the maximum number of packets that any node will have to store during the entire algorithm.

Valiant and Brebner [**?**] gave a randomized permutation routing algorithm for the parallel hypercube that runs in $O(\log N)$ time on an N-node network, the queue size being $O(\log N)$. Their algorithm also works for *partial permutations*, i.e., routing with at the most one packet originating at any node and at the most one packet destined for any node.

Upfal [**?**] calls a routing problem to be a *q-bounded communication request* if intially at most q packets are located at any node, and no more than q packets will have to be sent to any node.

The following theorem is due to Ranade [**?**]

Theorem 2.1 *There exists a randomized algorithm for routing partial permutations on an N-node butterfly that runs in time $O(\log N)$ and needs only $O(1)$ sized queues.*

As a consequence of the above theorem, we can prove the following

Lemma 2.1 *Any $O(\log N)$-bounded communication request can be routed on an N-node parallel hypercube in $O(\log N)$ time, with a queue size of $O(\log N)$.*

Proof. We can trivially embed an $N \log N$-node butterfly on an N-node parallel hypercube [**?**]. Each node in the hypercube will correspond to $\log N$ butterfly nodes.

Also, Ranade's algorithm runs in the same time bound even for an $O(1)$-bounded communication request. These two facts imply the lemma. □

This proof assumes that the $O(\log N)$ packets that are destined for any hypercube node are distinct, i.e., they have addresses of the corresponding butterfly nodes ($O(1)$ packets destined for each butterfly node). This assumption can be eliminated for the following special case.

Lemma 2.2 *If no more than $O(\log N)$ packets originate from any node and no more than $O(1)$ packets are destined for any node, routing can be done in $O(\log N)$ time on an N-node hypercube. The queue size will not exceed $O(\log N)$ at any node.*

Consider the following problem. In a hypercube with $N/\log N$ nodes there are at the most $\log N$ packets per node to start with. Each packet is chosen to be in a sample with probability $N^{-\epsilon}$ (where ϵ is a constant < 1), independently of the other packets. (This means that the number of packets in the sample is $\widetilde{O}(N^{1-\epsilon})$). We want to concentrate the packets in the sample in a subcube of size N^{δ} where $(1-\epsilon+\frac{1}{6}) < \delta < 1$. This task of concentration is performed as follows. Each packet in the sample chooses a random node in the subcube and goes there along the shortest path. How long does it take to perform this routing step on the sequential hypercube? The next theorem answers this question:

Theorem 2.2 *The above routing step can be completed in $\widetilde{O}(\log N)$ time.*

Proof. Let $v_1, v_2, \ldots, v_m$ be the sequence of nodes in the path taken by an arbitrary packet π in the sample. Then, m is at the most $\log N$. In order to compute the time taken by this packet to reach its destination on the sequential model, it suffices to count how many other packets will ever visit any of the nodes in π's path. We show that w.h.p. no more than $O(1)$ packets will ever visit any node. This will immediately imply that the time taken by any packet to reach its destination is $O(\log N)$ with high probability. Let X' stand for the number of packets that will ever visit a given node v.

Now look at a slightly different routing problem. Every node in an $N/\log N-$node hypercube has $\leq \log N$ packets to start with. Each packet is chosen to be in a sample with probability $N^{-\epsilon+(1-\delta)}(= N^{-\gamma}$,say). Every packet in the sample is sent to a random node in the whole cube along the shortest path. Let X be the number of packets that will ever visit the node v in this routing.

It is easy to see that X' is upperbounded by X. Thus to prove the lemma it is enough to show that X is $O(1)$ with high probability for any node v.

Realize that γ satisfies the condition $\frac{1}{6} < \gamma < \epsilon$. To prove an upper bound on X we make use of the fact that a hypercube with $N = 2^n$ nodes can be laid out as a butterfly with $(n+1)2^n$ nodes (see e.g., [**?**]). The butterfly nodes are arranged in $n+1$ 'levels' (or 'ranks') with 2^n nodes in each rank. Each node in the butterfly is labelled with a tuple $< rank, address >$, where *rank* is the level in which the node is located and *address* is an n-bit binary number. Each node in rank $i (0 \leq i \leq n)$ is connected to exactly two nodes in rank $i-1$ (if this rank exists) and exactly two nodes in rank $i+1$ (if this rank exists). For any node $< i, a >$, its two neighbors in rank $i+1$ are $< i+1, a >$ and $< i+1, a' >$, where a and a' (treated as n-bit binary numbers) differ only in the ith most significant bit.

For any n-bit binary number a, let 'column a' stand for all the $(n+1)$ nodes with address a. If each column of an $(n+1)2^n$-node butterfly is collapsed into a single node, we get a hypercube with $N = 2^n$ nodes. Our routing problem can be analyzed assuming that packets originate from rank 0 and have destinations in rank n of the butterfly. The following analysis of the routing algorithm on the butterfly is similar to the one given in [?] (which has been employed in [?] also).

Let l be any column in the butterfly. Let C_i be the number of packets that meet column l for the first time in rank i (i.e., at node $< i, l >$). Then $\sum_{i=0}^{n} C_i$ will be the number of packets that will ever meet column l (and hence the corresponding hypercube node).

Next we compute the distribution of C_i (i.e., we compute Prob.$[C_i = d]$ for each d). There are at the most $\binom{2^{i-1}n}{d}$ ways to choose the origins for the d packets (in rank 0). Probability that the d packets in the chosen origins belong to the sample is $\leq N^{-\frac{1}{6}d}$. Also, probability that each one of these packets goes through node $< i, l >$ is 2^{-id}. Therefore,

$$Prob.[C_i = d] \ \leq \ \binom{2^{i-1}n}{d} 2^{-id} N^{-d/6} \ \leq \frac{(n/2)^d N^{-d/6}}{d!}.$$

The probability generating function for (an upper bound of) C_i is given by $G_i(z) = \sum_d z^d Prob.[C_i = d] = e^{nN^{-1/6}z/2}$. Hence, the generating function for (an upper bound of) $\sum_{i=1}^{n} C_i$ is given by, $G(z) = \Pi_{i=1}^{n} G_i(z) = e^{n^2 N^{-1/6} z/2}$.

Therefore,

$$Prob.[X = d] \leq \left(\frac{n^2}{2N^{1/6}}\right)^d \frac{1}{d!}.$$

This implies

$$Prob.[X \geq 7\alpha] \ \leq \ 2\left(\frac{n^2}{2N^{1/6}}\right)^{7\alpha} \frac{1}{(7\alpha)!} \leq N^{-\alpha}$$

for any α.

This completes the proof that X is $\widetilde{O}(1)$ and the proof of theorem 2.2.□

Corollary 2.1 *Say we choose the sample in the above problem in the following way. The number of packets in the sample is specified to be r (where r is $\theta(N^{1-\epsilon})$). Each possible distribution of these r packets among the $N/\log N$ hypercube nodes is equally likely. Even if the sample is chosen this way, the above routing problem can be completed in $O(\log N)$ time w.h.p.*

Proof. is very similar to that of theorem 2.2 and hence is omitted.

We also use the following sorting algorithms on the hypercube.

Theorem 2.3 *(Nassimi and Sahni [?]) [Sparse Enumeration Sort] Sorting of n keys can be performed in $q \log n$ time on a hypercube using $n^{1+1/q}$ processors. (Initially the keys are located in a subcube of size n, one key per processor).*

Broadcasting is the operation of a single processor sending some information to all the other processors. The *summing* problem is this: Each processor in a hypercube has an integer. We need to obtain the sum of all these integers at some designated processor (say $00\ldots0$). Given two sorted sequences, the problem of *merging* is to obtain a sorted list of elements from both the lists.

Lemma 2.3 *Both broadcasting and summing can be completed in $O(\log P)$ steps on a P-node hypercube. Also, two sorted sequences of length P each can be merged in $O(\log P)$ time, given P hypercube processors.*

3 The Selection Algorithm

3.1 Summary

The algorithm we present can be thought of as an extension of existing randomized selection algorithms (in particular Floyd and Rivest's [?]) to the hypercube. Given a set X of N keys and an integer k, $1 \leq k \leq N$. Assume the keys are distinct (without loss of generality). We need to identify the kth smallest key. Like in the previous randomized selection algorithms we sample a set S of $o(N)$ keys at random. Sort the set S. Let l be the key in S with rank $m = \lceil k(|S|/N) \rceil$. We will expect the rank of l in X to be roughly k. We identify two keys l_1 and l_2 in S whose ranks in S are $m - \delta$ and $m + \delta$ respectively, δ being a 'small' integer, such that the rank of l_1 in X is $< k$, and the rank of l_2 in X is $> k$, with high probability.

Next we eliminate all the keys in X which are either $< l_1$ or $> l_2$. The remaining keys are sorted and the key that we are looking for will be one such with high probability. We do a trivial selection on the remaining keys to identify the right key.

3.2 Detailed Algorithm

Assume that the hypercube has $P = N/\log N$ processors. At the beginning each processor has $\log N$ keys. There is a queue associated with each edge incident on every node. There are 8 simple steps in the algorithm:

step1

Each processor flips an $N^{1/3}$-sided coin (for each of its $\log N$ keys) to decide if the key is in the 'random sample, S' with probability $N^{-1/3}$. This step takes $\log N$ time and with high probability $O(N^{2/3})$ keys (from among all the processors) will be selected to be in the random sample. Also no more than $O(1)$ packets will be selected from any node.

step2

P processors perform a prefix operation to compute the number of keys that succeeded in step 1. Let q be this number.

step3

Concentrate the keys that survived in step 2 in a subcube (call it C) of size $N^{3/4}$. This is done as follows. Each surviving key chooses a random node in the subcube. No more than $O(1)$ keys will choose any node of C w.h.p. Packets are routed to these destinations using Ranade's algorithm (see lemma 2.2).

step4

Sort the subcube C using sparse enumeration sort. Pick keys l_1 and l_2 from S with ranks $\left\lceil \frac{kq}{N} \right\rceil - \sqrt{N^{2/3} \log N}$ and $\left\lceil \frac{kq}{N} \right\rceil + \sqrt{N^{2/3} \log N}$ respectively.

step5

Broadcast l_1 and l_2 to all the processors in the hypercube.

step6

All the keys with a value $< l_1$ or $> l_2$ in the input drop out. The kth smallest key of the original input will be located in the range $[l_1, l_2]$ w.h.p. Count the number of surviving keys and call it r. Also count the number of keys in X with a value $< l_1$. Let this count be s.

step7

Concentrate the surviving keys in a subcube (say C') of size $N^{3/4}$ (similar to step 3).

step8

Sort C' using sparse enumeration sort. Find the key in C' with rank $k - s$ and output.

Analysis.
step1. Number of keys succeeding in step 1 is $B(N, N^{-1/3})$. Using Chernoff bounds, this number is $\leq c\alpha N^{2/3}$ with probability $\geq (1 - N^{-\alpha})$, for some constant c. Also, since there are $\log N$ keys per node at the beginning, the number of keys suceeding at any node is $B(\log N, N^{-1/3})$. Using Chernoff bounds again, this number is $O(1)$ w.h.p.

step2. This step takes $O(\log N)$ time since each node has $O(1)$ packets w.h.p.

step3. Given that the number of keys succeeding in step 1, q, is $O(N^{2/3})$, probability that more than e keys will choose the same node in C is $O\left(\frac{N^{2/3}}{N^{3/4}}\right)^e \leq N^{-\alpha}$ for any $\alpha \geq 1$

and some appropriate constant e. The routing step takes $O(\log N)$ steps according to lemma 2.2. This routing step is an $O(1)-$communication request w.h.p.

step4. Sparse enumeration sort takes $O(\log N)$ steps (since there are $O(1)$ packets per node w.h.p.) This can be done in $O(1)$ stages each taking $O(\log N)$ steps. In each stage pick one key per node and sort the whole subcube. Finally, merge these $O(1)$ sorted lists. (Nodes which do not have any key generate a dummy key with value ∞).

Once the sorting is done, l_1 and l_2 can be picked in $O(1)$ time.

step5. Broadcasting takes $O(\log N)$ steps (see lemma 2.3).

step6. If l_1 and l_2 are chosen as in step 4, the kth smallest key will be in the range $[l_1, l_2]$ and also the value of r will be $O(N^{2/3}\sqrt{\log N})$ w.h.p. This can be proved using the sampling lemma given in [**?**].

step7. This step can also be performed in $O(\log N)$ time (using lemma 2.2). In this routing step, there can be as many as $\log N$ packets originating from some of the nodes, but only $O(1)$ packets will be destined for any node w.h.p.

step8. Sorting C' and the final selection can also be completed in $O(\log N)$ steps (similar to step 4).

Thus we have the following

Theorem 3.1 *Selection on an input of size N can be performed on a $P = N/(\log N)$-node parallel hypercube network in time $O(N/P)$.*

4 Expected Behavior on the Sequential Model

In this section we show that the above algorithm has an expected run time of $O(\log N)$ using $N/\log N$ processors even on the sequential model of the hypercube (under the assumption that each possible permutation is equally likely to be the input). Notice that Plaxton's lower bound proof rules out a worst case $O(\log N)$ time deterministic algorithm with $N/\log N$ processors.

In the algorithm given in section 3, at the end of step 6, some nodes of the hypercube can still have as many as $\log N$ keys each. This is the reason why we need the parallel model to perform routing (in step 7). Also, step 7 is the only step in the entire algorithm

where the stronger model is needed. In every other step, a sequential model will do. In particular, step 3 can be completed on the sequential model in $O(\log N)$ steps (see theorem 2.2).

Under the assumption that each possible permutation is equally likely to be the input, and given that only $O(N^{2/3}\sqrt{\log N})$ keys survive after step 6 w.h.p. (see the analysis of step 6), step 7 also can be completed in $O(\log N)$ steps w.h.p. (making use of corollary 2.1). This time the 'high probability' not only involves the space of outcomes for coin flips but also the input space. Another way of stating this result is: Step 7 can be completed quickly on a large fraction of all possible inputs, and on each such input the algorithm terminates in $O(\log N)$ steps w.h.p. (this probability being over the space of outcomes for coin flips).

This means that the algorithm of section 3 runs for a large fraction ($\geq 1 - N^{-\beta}, \beta \geq 1$) of all possible inputs, and on each such input the algorithm terminates in $O(\log N)$ steps w.h.p. (this probability being over the space of outcomes for coin flips).

Acknowledgement. The author thanks Michael Palis, Sandeep Sen, Sunil Shende, Hui Wang, and David Wei for many stimulating discussions.

References

[1] Aho, Hopcroft, and Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[2] Ajtai, M., K'omlos, J., Steiger, W.L., and Szemeredi, E., 'Deterministic Selection in $O(\log \log n)$ Parallel Time,' Proc. ACM Symposium on the Theory of Computing, 1986, pp. 188-195.

[3] Cole, R., 'An Optimally Efficient Selection Algorithm,' Information Processing Letters 26, Jan. 1988, pp. 295-299.

[4] Cole, R., and Yap, C., 'A Parallel Median Algorithm,' Information Processing Letters 20(3), 1985, pp. 137-139.

[5] Floyd, R.W., and Rivest, R.L., 'Expected Time Bounds for Selection,' Communications of the ACM, vol. 18, no.3, 1975, pp. 165-172.

[6] Leighton, T., Lecture Notes on Parallel Algorithms, MIT, 1988.

[7] Meggido, 'Parallel Algorithms for Finding the Maximum and the Median Almost Surely in Constant Time', Preliminary Report, CS Department, Carnegie-Mellon University, Pittsburg, PA, Oct. 1982.

[8] Nassimi, D., and Sahni, S., 'Data Broadcasting in SIMD Computers,' IEEE Transactions on Computers, vol. c30, n0.2, 1981.

[9] Palis, M., Rajasekaran, S., and Wei,D., 'General Routing Algorithms for Star Graphs,' Technical Report, Department of Computer and Information Science, Univ. of Pennsylvania, 1989.

[10] Plaxton, C.G., 'Load Balancing, Selection and Sorting on the Hypercube,' Proc. First Annual ACM Symposium on Parallel Algorithms and Architectures, 1989, pp. 64-73.

[11] Plaxton, C.G., 'On the Network Complexity of Selection,' Proc. IEEE Symposium on Foundations Of Computer Science, 1989, pp. 396-401.

[12] Rajasekaran, S., and Reif, J.H., 'Derivation of Randomized Algorithms,' Technical Report, Aiken Computing Lab., Harvard University, March 1987.

[13] Rajasekaran, S., and Sen, S., 'Random Sampling Techniques and Parallel Algorithms Design,' to appear in *Synthesis of Parallel Algorithms*, Editor: Reif, J.H., 1990.

[14] Ranade, A., 'How to Emulate Shared Memory,' Proc. IEEE Symposium on Foundations Of Computer Science, 1987, pp. 185-194.

[15] Reischuk, R., 'Probabilistic Parallel Algorithms for Sorting and Selection,' SIAM Journal of Computing, vol. 14, no. 2, 1985, pp. 396-409.

[16] Upfal, E., 'Efficient Schemes for Parallel Communication,' Journal of the ACM, vol. 31, no. 3, July 1984, pp. 507-517.

[17] Valiant, L.G., and Brebner, G.J., 'Universal Schemes for Parallel Communication,' Proc. ACM Symposium on Theory Of Computing, 1981, pp. 263-277.

[18] Vishkin, U., 'An Optimal Parallel Algorithm for Selection,' Unpublished Manuscript, 1983.

APPENDIX A: Chernoff Bounds

Lemma A.1 *If X is binomial with parameters (n, p), and $m > np$ is an integer, then*

$$Probability(X \geq m) \ \leq \ \left(\frac{np}{m}\right)^m \ e^{m-np}. \tag{1}$$

Also,

$$Probability(X \leq \lfloor (1-\epsilon)pn \rfloor) \ \leq \ exp(-\epsilon^2 np/2) \tag{2}$$

and

$$Probability(X \geq \lceil (1+\epsilon)np \rceil) \ \leq \ exp(-\epsilon^2 np/3) \tag{3}$$

for all $0 < \epsilon < 1$.