October 1993

Deterministic Selection on the Mesh and Hypercube

Sanguthevar Rajasekaran
University of Pennsylvania

Shibu Yooseph
University of Pennsylvania

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-93-85.

Deterministic Selection on the Mesh and Hypercube

Abstract

In this paper we present efficient deterministic algorithms for selection on the mesh connected computers (referred to as the mesh from hereon) and the hypercube. Our algorithm on the mesh runs in time $O([n/p] \log\log p + \sqrt{p} \log n)$ where n is the input size and p is the number of processors. The time bound is significantly better than that of the best existing algorithms when n is large. The run time of our algorithm on the hypercube is $O([n/p] \log\log p + Ts/p \log$ nM/em>), where Ts/p is the time needed to sort p element on a p-node hypercube. In fact, the same algorithm runs on an network in time $O([n/p] \log\log p + Ts/p \log)$, where Ts/p is the time needed for sorting p keys using p processors (assuming that broadcast and prefix computations take time less than or equal to Ts/p.

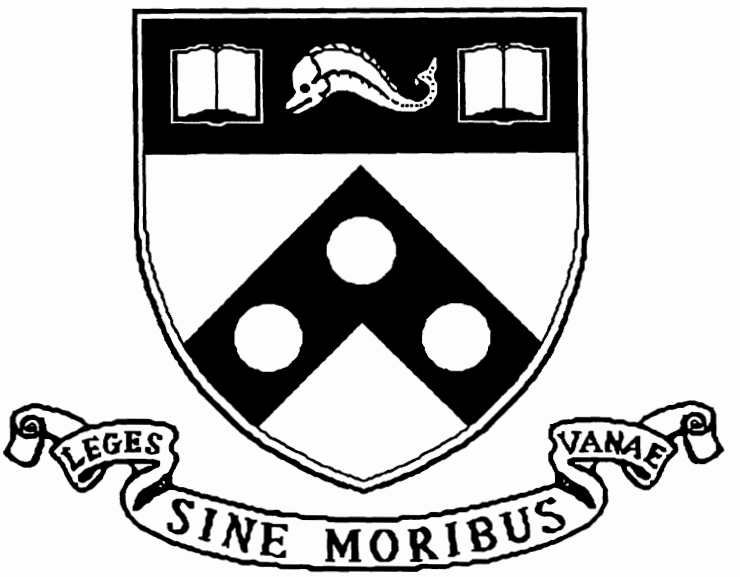
Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-93-85.

Deterministic Selection on the Mesh and the Hypercube

MS-CIS-93-85
GRASP LAB 363

Sanguthevaar Rajasekaran
Shibu Yooseph



University of Pennsylvania
School of Engineering and Applied Science
Computer and Information Science Department
Philadelphia, PA 19104-6389

October 1993

Deterministic Selection on the Mesh and the Hypercube[1]

Sanguthevar Rajasekaran
Shibu Yooseph

Dept. of CIS, Univ. of Pennsylvania
Philadelphia, PA 19104

Abstract. In this paper we present efficient deterministic algorithms for selection on the mesh connected computers (referred to as the mesh from hereon) and the hypercube. Our algorithm on the mesh runs in time $O(\frac{n}{p}\log\log p + \sqrt{p}\log n)$, where n is the input size and p is the number of processors. This time bound is significantly better than that of the best existing algorithm when n is large. The run time of our algorithm on the hypercube is $O(\frac{n}{p}\log\log p + T_p^s \log n)$, where T_p^s is the time needed to sort p elements on a p-node hypercube. In fact, the same algorithm runs on any network in time $O(\frac{n}{p}\log\log p + T_p^s \log n)$, where T_p^s is the time needed for sorting p keys using p processors (assuming that broadcast and prefix computations take time less than or equal to T_p^s).

1 Introduction

Given a set of n keys, and an integer i ($1 \leq i \leq n$), the problem of selection is to find the ith smallest key in the set. This important comparison problem has an elegant linear time sequential algorithm [1]. 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 mesh connected computers and the hypercube.

1.1 Models Definition

A mesh connected computer is a $\sqrt{p} \times \sqrt{p}$ square grid where there is a processor at each grid point. Each processor is connected to its four or less neighbors through bidirectional links.

[1]This research was supported in part by an NSF Research Initiation Award CCR-92-09260 and an ARO grant DAAL03-89-C-0031.

It is assumed that in one unit of time a processor can perform a local computation and/or communicate with all its neighbors.

A hypercube of dimension ℓ consists of $p = 2^\ell$ nodes (or vertices) and $\ell 2^{\ell-1}$ edges. Thus each node in the hypercube can be named with an ℓ-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 exactly one bit position (i.e., the hamming distance between x and y is 1.) Therefore, there are exactly ℓ 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.* We assume the sequential model in this paper.

1.2 Previous Results

Krizanc and Narayanan [6] have presented efficient algorithms for selection on the mesh. Their algorithm runs in time $O(\min\{p \log \frac{n}{p},\ \max\{\frac{n}{p^{2/3}}, \sqrt{p}\}\})$. However, they only account for the communication steps in the algorithm. In particular, they discount local computations performed at individual nodes.

Plaxton [11] has presented an algorithm for selection out of n elements that runs on a p-node sequential hypercube in time $O((n/p) \log\log p + (T_p^s + T_p^b \log p) \log(n/p))$, where T_p^s is the time needed for sorting p keys (located one per processor) on a p-processor hypercube, and T_p^b is the time needed for broadcasting and summing on a p-node hypercube. He [11] has also proved a lower bound of $\Omega((n/p) \log\log p \ + \ \log p)$ for selection. For $n \geq p \log^2 p$ the lower bound matches the upper bound (to within a multiplicative constant). The only operations allowed on the keys are copying and comparison (for both the upper bound and the lower bound).

Meggido's [9] algorithm does maximal and median selection in constant time using a linear number of processors on the comparison tree model. Reischuk's [17] selection algorithm runs in $O(1)$ time using n comparison tree processors. Floyd and Rivest's [4] sequential algorithm takes $n + \min(i, n-i) + o(n)$ time. In [12], Rajasekaran has presented randomized algorithms for selection on the hypercube (on both the sequential and parallel versions). Rajasekaran and Sen [15] give an $O(1)$ time n processor maximal selection algorithm for the CRCW PRAM model. Krizanc and Narayanan [5] have presented optimal algorithms for selection

on the mesh connected computers. All these results hold for the worst case input with high probability. For an extensive survey of randomized selection algorithms, see [14]. In this paper we present deterministic algorithms for selection on the mesh and the hypercube.

1.3 New Results

Our algorithm for selection on the mesh runs in time $O(\frac{n}{p}\log\log p \ + \ \sqrt{p}\ \log n)$, taking into account all local computations performed. Since $\Omega(\frac{n}{p} + \sqrt{p})$ is a trivial lower bound, our algorithm is very nearly optimal. If we neglect the time spent on local computations, the run time of our algorithm will be $O(\sqrt{p}\ \log n)$. Clearly, this time bound is close to the trivial lower bound of $\Omega(\sqrt{p})$. For all $n > p^{7/6}\log p$, our algorithm will have a much better run time than that of [5].

Our algorithm for selection on the hypercube runs in time $O(\frac{n}{p}\log\log p \ + \ T_p^s \log n)$, where T_p^s is the time needed for sorting on a p-node hypercube with one key per node. The best known value for T_p^s is $O(\log p \log\log p)$ [3]. With this value for T_p^s, the run time of our algorithm very nearly matches that of Plaxton [11]. But if a better sorting algorithm is discovered, the run time of our algorithm will improve somewhat, whereas [11]'s algorithm does not seem to improve.

In fact, our algorithm could be implemented on any network to obtain a run time of $O(\frac{n}{p}\log\log p \ + \ T_p^s \log n)$, where T_p^s is the time needed for sorting p numbers on a network of size p.

2 Preliminary Facts

2.1 Sorting

We make use of existing sorting algorithms. The following theorem is due to Schnorr and Shamir [18]:

Lemma 2.1 *Sorting on a p-node mesh can be completed in time $O(\sqrt{p})$, the queue size being $O(1)$ if there is a single key input at each node.*

Cypher and Plaxton [3] have proven the following:

Lemma 2.2 *Sorting on a p-node hypercube can be completed in time $O(\log p \log\log p)$ [3].*

2.2 Broadcasting and Summing

Broadcasting is the operation of a single processor sending some information to all the other processors. The *prefix sums problem* is this: Processor v in a p-node hypercube has an integer k_v, for $1 \leq v \leq p$. Processor v has to compute $\sum_{j=1}^{v} k_j$.

Lemma 2.3 *Both broadcasting and prefix sums problem can be completed in $O(\sqrt{p})$ steps on a p-node mesh.*

Lemma 2.4 *Both broadcasting and prefix sums problem can be completed in $O(\log p)$ steps on a p-node sequential hypercube.*

3 Summary of our Techniques

The basic idea behind our algorithm is the same as the one employed in [2]. The sequential algorithm of [2] partitions the input into groups (of say 5), finds the median of each group, and computes recursively the median (call it M) of these group medians. Then the rank r_M of M in the input is computed and as a result, all the elements from the input which are either $\leq M$ or $> M$ are dropped, depending on whether $i > M$ or $i \leq M$, respectively. Finally, an appropriate selection is performed from out of the remaining keys recursively. An easy analysis will reveal that the run time of this algorithm is $O(n)$.

The same algorithm can be used in parallel, for instance on a PRAM, to obtain an optimal algorithm. If one has to employ this algorithm on a network, it seems like one has to perform periodic load balancing (i.e., distribute remaining keys uniformly among the processors). In [11], an algorithm is given which identifies an M for splitting the input upon, which automatically ensures (approximate) load balancing. That is, at least one half of the keys from any node will be eliminated every time the remaining keys are splitted.

In this paper we introduce a different approach. We employ the same algorithm as that of [2], with a twist. To begin with each node has exactly $\frac{n}{p}$ keys. As the algorithm proceeds, keys get dropped from future consideration. We never perform any load balancing. The remaining keys from each node will form the groups. We identify the median of each group. Instead of picking the median of these medians as the splitter key M, we choose a weighted median of these medians. Each group median is weighted with the number of remaining keys in that node. This simple algorithm (with some minor modifications) yields the stated results.

4 Selection on the Mesh

In this section we show that selection can be done in time $O(\frac{n}{p} \log\log p \ + \ \sqrt{p} \ \log n)$ on a $\sqrt{p} \times \sqrt{p}$ mesh, the input size being n. To begin with, there are exactly $\frac{n}{p}$ keys at each node. We need to find the ith smallest key.

Algorithm I

$N := n$

Step 0. *if* $\log(n/p)$ is $\leq \log\log p$ *then*

sort the elements at each node

else

partition the keys at each node into $\log p$ equal parts.

repeat

Step 1. In parallel find the median of keys at each node. Let M_q be the median and N_q be the number of keys at node q, $1 \leq q \leq p$.

Step 2. Find the weighted median of $M_1, M_2, \ldots, M_p$ where key M_q has a weight of N_q, $1 \leq q \leq p$. Let M be the weighted median.

Step 3. Count the rank r_M of M from out of all the remaining keys.

Step 4. *if* $i \leq r_M$ *then*

eliminate all remaining keys that are $> M$

else

eliminate all keys that are $\leq M$.

Step 5. Compute E, the number of keys eliminated.

if $i > r_M$ *then* $i := i - E$; $N := N - E$.

until $N \leq c$, c being a constant.

Output the ith smallest key from out of remaining keys.

Analysis. Step 0 takes time $\frac{n}{p} \min\{\log(n/p) \ , \ \log\log p\}$. Assume here that $\log p$ is an integral power of 2; if not take the nearest power of 2 larger than $\log p$. At the end of Step 0, the keys in any node have been partitioned into nearly $\log p$ nearly equal parts. Call each such part as a block. During the algorithm, a node will delete more and more blocks until it is left with just a single block. From then on median will be found at this node using a linear time sequential algorithm.

In Step 1, we could find the median at any node in $O(1)$ time up to at least the end of the first $T = \min\{\log(n/p) \ , \ \log\log p\}$ runs of the *repeat* loop, i.e., up to the time that the

node has only one block left. From this time on, finding the median will take time $O(\frac{n}{p \log p})$. Thus in the algorithm, after T runs, we'll allow $O(\frac{n}{p \log p})$ time for Step 1, so that each node is guaranteed to find its median.

In Step 2, we could sort the medians and thereby compute the weighted median. If $M'_1, M'_2, \ldots, M'_p$ is the sorted order of the medians, then, we need to identify j such that $\sum_{k=1}^{j} N'_k \geq \frac{N}{2}$ and $\sum_{k=1}^{j-1} N'_k < \frac{N}{2}$. Such a j can be computed with an additional prefix computation. Thus M, the weighted median, can be identified in time $O(\sqrt{p})$ (c.f. Lemmas 2.1 and 2.3). Step 3 takes $O(\sqrt{p})$ time. Step 4 also takes $O(\sqrt{p})$ time, since it only involves the broadcast of r_M; the deletion takes $O(1)$ time if the elements are stored in an array. Step 5 takes $O(1)$ time.

Thus each run of the *repeat* loop takes $O(\frac{n}{p \log p} + \sqrt{p})$ time.

How many keys will get eliminated in each run of the *repeat* loop? Assume that $i \leq r_M$ in a given run. (The other case can be argued similarly). The number of keys eliminated is at least $\sum_{k=1}^{j} \left\lceil \frac{N'_k}{2} \right\rceil$ which is $\geq \frac{N}{4}$. Therefore, it follows that the *repeat* loop will be executed $O(\log n)$ times. Thus we get (assuming that $\log n$ is asymptotically the same as $\log p$):

Theorem 4.1 *Selection on a p-node square mesh can be performed in time* $O(\frac{n}{p} \log \log p + \sqrt{p} \log n)$.

Often times, the time needed for communication far outweighs the time needed for local computations in a network based computer. Thus it may be reasonable to neglect local computations. In [5], Krizanc and Narayanan make this assumption to derive the run time of their algorithm. It is easy to compute the run time of our algorithm under this assumption and obtain the following:

Theorem 4.2 *If local computations are neglected, the run time of our algorithm is* $O(\sqrt{p} \log n)$.

5 Selection on the Hypercube

Our selection algorithm when applied on the hypercube yields a run time of $O(\frac{n}{p} \log \log p + T_p^s \log n)$, where T_p^s is the time needed to sort p keys on a p-node hypercube. With the currently best known value for T_p^s, the run time of our algorithm very nearly matches that of [11]. However, if a better sorting algorithm is discovered, the run time of our algorithm will improve. Our algorithm is also somewhat simpler than that of [11]'s.

Here also, there are $\frac{n}{p}$ keys to begin with at each node and we have to identify the ith smallest key.

Algorithm II

$N := n$

Step 0. *if* $\log(n/p)$ is $\leq \log\log p$ *then*

sort the elements at each node

else

partition the keys at each node into $\log p$ equal parts.

repeat

Step 1. In parallel find the median of keys at each node. Let M_q be the median and N_q be the number of keys at node q, $1 \leq q \leq p$.

Step 2. Find the weighted median of $M_1, M_2, \ldots, M_p$ where key M_q has a weight of N_q, $1 \leq q \leq p$. Let M be the weighted median.

Step 3. Count the rank r_M of M from out of all the remaining keys.

Step 4. *if* $i \leq r_M$ *then*

eliminate all remaining keys that are $> M$

else

eliminate all keys that are $\leq M$.

Step 5. Compute E, the number of keys eliminated.

if $i > r_M$ *then* $i := i - E$; $N := N - E$.

until $N \leq c$, c being a constant.

Output the ith smallest key from out of remaining keys.

Analysis. Step 0 takes time $\frac{n}{p}$ $\min\{\log(n/p)$, $\log\log p\}$. At the end of Step 0, the keys in any node have been partitioned into nearly $\log p$ nearly equal parts. Call each such part as a block. During the algorithm, a node will delete more and more blocks until it is left with just a single block. From then on median will be found at this node using a linear time sequential algorithm.

Step 1 takes time $O(\frac{n}{p\log p})$ just as in the mesh algorithm. In Step 2, we could employ a sorting followed by a prefix computation in order to identify the weighted median. Thus Step 2 will take time $O(T_p^s + \log p)$ (c.f. Lemmas 2.2 and 2.4). Step 3 takes $O(\log p)$ time each in accordance with Lemma 2.4. Step 4 also takes $O(\log p)$ time, since it only involves the broadcast of r_M; the deletion takes $O(1)$ time if the elements are stored in an array. Step 5 takes $O(1)$ time.

Therefore, each run of the *repeat* loop takes $O(\frac{n}{p\log p} + T_p^s + \log p)$ time. At least a constant fraction of the keys get eliminated during each run of the *repeat* loop (for the same reason as

in the mesh algorithm). Therefore, assuming that $\log n$ is asymptotically the same as $\log p$, the run time of the algorithm is $O(\frac{n}{p} \log \log p + T_p^s \log n)$. As a result, we get:

Theorem 5.1 *Selection on a p-node hypercube can be performed in time $O(\frac{n}{p} \log \log p + T_p^s \log n)$, where T_p^s is the time needed for sorting and n is the input size.*

The following theorem is also clear:

Theorem 5.2 *Selection on a p-node hypercube can be performed in time $O(\frac{n}{p} \log \log p + T_p^w \log n)$, where T_p^w is the time needed for computing the weighted median of p numbers on a p-node hypercube.*

Our selection algorithm can be implemented on any network to obtain a very nearly optimal run time. The following theorem assumes that broadcast and prefix computations take time less than or equal to the time needed for sorting:

Theorem 5.3 *Selection can be performed in time $O(\frac{n}{p} \log \log p + T_p^s \log n)$ on any network with p processors, where T_p^s is the time needed for sorting p numbers using p processors.*

6 Conclusions

We have presented deterministic algorithms for selection on the hypercube as well as the mesh. Our mesh algorithm has a run time significantly better, when n is large, than the best existing algorithm. Also, our hypercube selection algorithm will have a better run time with the discovery of a better sorting algorithm. Our algorithms are very nearly optimal. It is still open to find optimal algorithms. We could implement the selection algorithm presented in this paper on any network to obtain near optimal run times.

Acknowledgements

The first author would like to thank Danny Krizanc for many stimulating interchange of ideas.

References

[1] A. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley Publishing Company, 1974.

[2] M. Blum, R. Floyd, V.R. Pratt, R. Rivest, and R. Tarjan, Time Bounds for Selection, *Journal of Computer and System Science*, 7(4), 1972, pp. 448-461.

[3] R. Cypher and G. Plaxton, Deterministic Sorting in Nearly Logarithmic Time on the Hypercube and Related Computers, in Proc. *ACM Symposium on Theory of Computing*, 1990, pp. 193-203.

[4] R.W. Floyd, and R.L. Rivest, Expected Time Bounds for Selection, *Communications of the ACM*, Vol. 18, No.3, 1975, pp. 165-172.

[5] D. Krizanc, and L. Narayanan, Optimal Algorithms for Selection on a Mesh-Connected Processor Array, Proc. *Symposium on Parallel and Distributed Processing*, 1992.

[6] D. Krizanc and L. Narayanan, Multi-packet Selection on a Mesh-Connected Processor Array, in Proc. *International Parallel Processing Symposium*, 1992.

[7] T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays–Trees–Hypercube*, Morgan-Kaufmann Publishers, 1992.

[8] T. Leighton, F. Makedon, and I. Tollis, A $2N - 2$ Step Algorithm for Routing in an $N \times N$ Mesh, in Proc. *ACM Symposium on Parallel Algorithms and Architectures*, 1989, pp. 328-335.

[9] N. 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.

[10] D. Nassimi, and S. Sahni, Data Broadcasting in SIMD Computers, *IEEE Transactions on Computers*, Vol. C30, No. 2, 1981.

[11] C.G. Plaxton, Efficient Computation on Sparse Interconnection Networks, Ph. D. Thesis, Department of Computer Science, Stanford University, 1989.

[12] S. Rajasekaran, Randomized Parallel Selection, Proc. *Symposium on Foundations of Software Technology and Theoretical Computer Science*, 1990, pp. 215-224.

[13] S. Rajasekaran and R. Overholt, Constant Queue Routing on a Mesh, *Journal of Parallel and Distributed Computing* 15, pp. 160-166, 1992.

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

[15] S. Rajasekaran, and S. Sen, Random Sampling Techniques and Parallel Algorithms Design, in *Synthesis of Parallel Algorithms*, Editor: J.H. Reif, Morgan-Kaufman Publishers, 1993, pp. 411-451.

[16] J.H. Reif and L.G. Valiant, A Logarithmic Time Sort for Linear Size Networks, *Journal of the ACM* 34, 1987, pp. 60-76.

[17] R. Reischuk, Probabilistic Parallel Algorithms for Sorting and Selection, *SIAM Journal of Computing*, Vol. 14, No. 2, 1985, pp. 396-409.

[18] C. Schnorr and A. Shamir, An Optimal Sorting Algorithm for Mesh-Connected Computers, in Proc. *ACM Symposium on Theory of Computing*, 1986, pp. 255-263.

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

[20] P. Varman and K. Doshi, Sorting with Linear Speedup on a Pipelined Hypercube, TR-8802, Department of Electrical and Computer Engineering, Rice University, 1988.