A Probabilistic Pipeline Algorithm For K-Selection on the Tree Machine

by

Albert G. Greenberg
Udi Manber

Computer Sciences Technical Report #558

September 1984

A PROBABILISTIC PIPELINE ALGORITHM FOR K-SELECTION ON THE TREE MACHINE

Albert G. Greenberg

AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974

Udi Manber

Department of Computer Science
1210 W. Dayton Street
University of Wisconsin
Madison, WI 53706

ABSTRACT

The tree machine is a synchronous parallel computer in which the underlying communication graph is a complete binary tree. We consider the problem of selecting the k-th largest of n inputs, where initially the inputs are stored in the n leaf processors of the $2n-1$ processor tree machine. A probabilistic algorithm is presented that implements a type of pipelining to solve the problem efficiently. Specifically, on the any problem instance, the algorithm's running time is $< c \frac{\log^2 n}{\log\log n}$, for some constant $c > 0$, with overwhelming probability. Throughout the course of execution, each processor operates in a simple data driven fashion, maintaining just a constant amount of state information.

1. INTRODUCTION

The tree machine is a synchronous parallel computer in which the underlying communication graph is a complete binary tree. The computational power of the tree machine has been the subject of much research, owing in part to its potential for an efficient VLSI implementation [1,2]. A prototype tree machine involving 1023 processors is being built at Columbia University [7]. We consider the problem of *k-selection*, finding the k-th largest of n inputs, where initially the inputs are stored in the n leaf processors of a $2n-1$ processor tree machine.

Tanimoto considered the problem of finding the median value of n^2 pixels in a pyramid machine whose lowermost processor level coincides with the pixels [8]. Operations involving the median pixel are useful in certain filtering and smoothing tasks in image processing [8]. The underlying communication graph of a pyramid machine is a 4-way complete tree so, apart from superficial detail, this is a special case of the problem considered here. Tanimoto gave a number of deterministic algorithms for selecting the median, the fastest of which has running time $O(n^2 \log n)$ [8]. Fredrickson [4] presented a clever adaptation of an algorithm of Munro and Patterson [5] to show that k-selection in a complete binary tree can be accomplished deterministically in $O(\log^3 n)$ time.

Shrira et al. [6] recently reported a probabilistic algorithm for k-selection on the tree machine that runs in expected time $O(\log^2 n)$ on the worst problem instance. The algorithm is based on the following straightforward procedure. Let *value(x)* denote the value of an input x with respect to some total order, *less(x)* the number of inputs with value less than that of x, and *equal(x)* the number of inputs with value equal to that of x. (Thus, $less(x) \geq 0$ and $equal(x) \geq 1$.) The procedure works in stages, with each stage operating on a *candidate set* C. Initially C is the set of all n inputs, and on termination C is empty.

A stage has three parts:

1. Pick a member x of C uniformly at random.
2. Compute *less*(x) and *equal*(x).
3. Compute a new candidate set C' according to the rules:
   a. If *less*$(x) < k \leqslant$ *less*(x) + *equal*(x) then *value*(x) is the k-th largest, so $C' = \varnothing$.
   b. Otherwise, for every y in C, include y in C' if either $k \leqslant$ *less*(x) and *value*$(y) <$ *value*(x), or $k >$ *less*(x) + *equal*(x) and *value*$(y) >$ *value*(x).

If $C' \neq \varnothing$ then this is repeated with C' replacing C. Shrira et al. presented an $O(\log n)$ implementation of each of the three operations constituting a stage. With high probability, a stage cuts the size of the candidate set down by at least a constant fraction, whence the $O(\log^2 n)$ expected running time bound.

In this paper, we give a different algorithm for k-selection based on the same procedure. Our main innovation is to pipeline the solutions of the first two subproblems within a stage to achieve a significant speedup and a simpler algorithm to implement. Our algorithm has expected running time $< c \dfrac{\log^2 n}{\log\log n}$, for some constant $c > 0$.* Simulation results indicate that c is small. The probability distribution for the running time is peaked, in the sense that, for all $t > t_0$, the probability of the running time exceeding $t\, c \dfrac{\log^2 n}{\log\log n}$ goes to 0 as $n^{-dt\,/\,\log\log n}$, where t_0 and d are positive constants. Each processor operates in a data driven fashion, maintaining just a constant amount of state information throughout execution. In view of the simplicity and low order of complexity of our algorithm, it appears to be the algorithm of choice.

* When not given explicitly, the base of the logarithm is e.

In section 2 we present the algorithm and its analysis. In section 3 we describe an improvement that lessens the expected value of running time by about 15% and the corresponding standard deviation by at least that much, as evidenced by simulation data also presented in this section. In section 4, we summarize the results and mention an application to sorting.

2. THE ALGORITHM

We assume that there are $2n-1$ processors and that $n = 2^p$ for some integer p. The underlying communication graph is a complete binary tree. Processors are identified with nodes in the tree and communication links with edges. Processors operate synchronously in steps. A step consists of two sets of actions:

1. A message may be received from each adjacent processor. (There are at most 3).
2. A message may be transmitted to each adjacent processor. These messages may depend on those just received.

Thus, a message transmitted at step t is received and acted on at step $t+1$. It is reasonable to consider the operations of receiving up to 3 messages and then transmitting up to 3 new ones as one step, provided the local computations that go along with generating the new messages are simple. In our k-selection algorithm, these computations are simple.

Our aim is to find an efficient implementation for the tree machine of the procedure for k-selection described in section 1. Recall that the k-selection procedure operates iteratively on a candidate set consisting of at least one input whose value coincides with the k-th largest. An iteration involves two main operations: picking a member x of the candidate set uniformly at random, and computing $less(x)$ and $equal(x)$, the number of inputs with value less than x and the number with value equal to that of x (respectively). We refer to x as a *splitter*, and the pair, $less(x)$ and $equal(x)$, as the *rank* of x.

In the k-selection problem, the n inputs initially reside in the local memories of the n leaf processors. Thus, the candidate set can easily be maintained in a distributed fashion: Have each of the n leaf processors maintain a bit in its local memory indicating whether or not its input is a candidate. A natural way to implement the k-selection procedure is as follows.

1. Pick a splitter x and broadcast it to the leaves.
2. Compute $less(x)$ and $equal(x)$ and broadcast the triple $(value(x), less(x), equal(x))$ to the leaves.
3. In parallel, each leaf processor compares its input with x and decides whether its input should remain in the candidate set. Check whether the candidate set is now empty.

If, as a result, the candidate set is not empty, then the three operations are repeated. Each of the three operations can be carried out in $O(p)$ steps. (p is the height of the tree.) This implementation of the k-selection procedure is similar to the one proposed by Shrira et al. [6]. We refer to it as the straighforward algorithm. (In [6] this algorithm is presented to illustrate a technique for transforming a sequential algorithm into a distributed one.)

To broadcast a message, the root can transmit the message to its two sons, which relay it to their sons, and so forth. The values $less(x)$ and $equal(x)$ can be computed in the obvious way, accumulating the sums from leaves to root. Shrira et al. suggested a top down way to pick splitters. For our purposes, a bottom up method is more appropriate: Have each leaf processor send the message (v,s) to its father, where

- v is the value of its input, and
- s is 1 if the input belongs to the candidate set and 0 otherwise.

Upon receipt of messages (v_1,s_1) and (v_2,s_2), a processor associated with an internal node determines v and s, where $s = s_1 + s_2$ and v is either v_1 or v_2. If $s_1 = s_2$ then the choice of v is settled on the basis of an unbiased coin toss. Otherwise $v = v_1$ with probability $\frac{s_1}{s_1+s_2}$

and $v = v_2$ with the complementary probability. If this processor is not associated with the root, it sends (v,s) to its father. If the processor is associated with the root, it must be that v is the value of some x chosen uniformly at random from the candidate set, which must have size s.

One indication that this implementation of the k-selection procedure is probably not optimal is that each processor is busy just one out of every p steps. By utilizing the processors more fully, the splitter selection and rank computation operations can be pipelined, so that the new operations can begin at each step. The details are as follows.

There are three types of messages:

1. A special *start* message.
2. A *splitter* message (v, s), where v is the value of the splitter and s is the size of the set from which the splitter was chosen.
3. A *rank* message (v, l, e), where v is the value of the candidate, and l and e are the number of inputs known to be less than and equal to (respectively) v.

All processors, excepting the root, are initially quiescent, and are activated upon receiving the start message. To begin the computation the root sends the start message to its two sons. Thereafter each processor operates under the following rules, which depend on the type of node it is associated with. We call this algorithm the *basic pipeline* algorithm.

1. [Internal node, not the root] Any message received from its father is relayed to its two sons. For messages received from below there are two cases:
   a. The two messages are rank messages of the form (v, l_1, e_1) and (v, l_2, e_2). In this case, the processor transmits (v, l, e) to its father, where $l = l_1 + l_2$ and $e = e_1 + e_2$.
   b. The two messages are splitter messages of the form (v_1, s_1) and (v_2, s_2). In this case, the processor transmits (v', s) to its father, where $s = s_1 + s_2$ and v' is either v_1 or v_2; the decision is made probabilistically as described above.
2. [Root] The processor associated with the root behaves just like one associated with any other internal node, except that all messages are sent to the two sons instead of to the father.

3. [Leaf] Suppose that the processor holds an input with value w. After receiving the start message and before termination, the processor always transmits a message to its father. There are three cases.

   a. It receives either the start message, or receives no message at all after having received the start message earlier. In this case, the processor transmits $(w,1)$. (This initiates a splitter selection, which results in a splitter being chosen uniformly at random from the n inputs.)

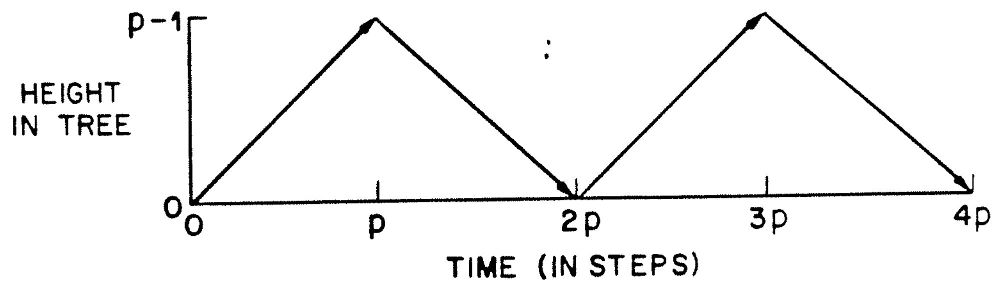
   b. It receives a rank message of the form (v, l, e). In this case, the processor decides whether its input still belongs to the candidate set, and transmits $(w,1)$ if so and $(w,0)$ if not. (Thus, the splitter x reduces the candidate set, and initiates a new splitter selection, which results in a new splitter being chosen uniformly at random from the current candidate set.)

   c. It receives a splitter message of the form (v', s). (This represents a newly selected splitter x with value v', of undetermined rank.) In this case, the processor transmits (v', l, e) where $e = 1$ if $w = v'$ and 0 otherwise, and $l = 1$ if $w < v'$ and 0 otherwise. (This initiates the rank computation for x.)

Before discussing termination, let us consider the behavior of the algorithm over the course of the first several steps. At step 1, the start message is issued, copies of which are relayed down the tree and are received at the leaves at step $p+1$. As a result, at step $p+1$ each leaf promotes its value, entering it in a splitter selection. Each leaf does the same at steps $p+2$ through $3p$, because it receives no messages at these steps. This puts underway $2p$ splitter selections. At steps $3p+1$ through $5p$, $2p$ newly selected splitters arrive at the leaves, triggering the computation of their ranks. At steps $5p+1$ through $7p$, the $2p$ splitters, now appropriately bound to their ranks, return to reduce the candidate set and initiate a new phase of $2p$ splitter selections.

Thus, we may think of the algorithm as proceeding in *stages*, each consisting of $4p$ steps, where a stage is composed of two *phases*, these consisting of $2p$ steps: a splitter selection phase, followed by a rank computation phase. Figure 1 depicts how the information associated with a splitter travels up and down the tree during a stage.

Figure 1



A little reflection indicates that with probability 1, a splitter is eventually selected whose value v coincides with the k-th largest. This leads the root to send (v, l, e) to its two sons, with $l < k \leqslant l + e$. The message is relayed on to every other node. The computation terminates gracefully under the assumption that an individual processor terminates at the step following the receipt of this message. We measure the running time of the algorithm as the number of steps that elapse after the leaves have received the start message up to the step at which the root receives the message (just described) signaling termination.

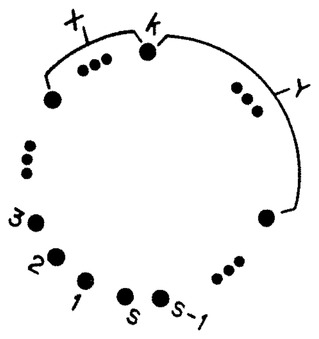
We now present a pessimistic analysis of the algorithm, which leads to upper bounds on the expected running time and the probability of the running time exceeding its expectation by a constant factor.

Suppose that the candidate set C has size $s > 0$ just before the i-th stage. Let s' denote the size of the candidate set just before the rank computation phase of the $(i+1)$-st stage.

Lemma 1: $s' \leqslant \frac{s}{p}$, with probability $> 1-\frac{2}{e}$, for all $i \geqslant 1$.

Proof:

Figure 2



Think of the s members of C as being arranged in sorted order about a circle as indicated in Figure 2. An input of smallest value is at the bottom, and the values increase going counterclockwise around the circle. Starting at the input marked k (one of the k-th largest), scan counterclockwise around the circle until hitting, for the first time, an input picked as a splitter in stage i. Let x be the number of inputs, counting the one marked k, passed over before hitting the splitter. Similarly, scan clockwise around the circle until hitting a splitter and let y denote the number of inputs passed over.

Suppose for the moment that the stage index $i = 1$, so that the computation is just beginning, and the candidate set C is the set of all inputs. Each of the $2p$ splitters generated at the first stage are chosen uniformly at random with replacement from C. Thus, $s' = 0$ if the value of at least one of these splitters coincides with the k-th largest, and $s' \leqslant x+y-1$ otherwise. Hence $s' > \frac{s}{p}$ with probability at most

$$Prob(x+y > \frac{s}{p}) \leqslant Prob(x > \frac{s}{2p}) + Prob(y > \frac{s}{2p})$$
$$\leqslant 2\,(1-\frac{1}{2p})^{2p} < \frac{2}{e},$$

as desired.

Now, suppose that the stage index $i > 1$. An added complication is that the $2p$ splitters generated at the $(i-1)$-st stage arrive at the leaves during the the first $2p$ steps of the i-th stage, influencing the generation of the new group of $2p$ splitters. However, the bounds given above for $Prob(x > \frac{s}{2p})$ and $Prob(y > \frac{s}{2p})$ still hold. To see this for x, note that $x > z$ holds if, at each of the first $2p$ steps of the i-th stage, no member of a certain subset of C of size z is selected to be a splitter. Letting c $(0 \leqslant c \leqslant s)$ denote the candidate set size at any given one of these steps, the probability of not picking one of the z is $1-\frac{z}{c} \leqslant 1-\frac{z}{s}$, provided $c > z$, and is 0 otherwise. Thus,

$$Prob(x > z) \leqslant (1-\frac{z}{s})^{2p}.$$

■

Theorem 1: The expected running time of the k-selection algorithm is less than

$$c\,\frac{\log^2 n}{\log\log n},$$

for some constant $c > 0$. Moreover, for all $t > t_0$, the running time exceeds $t\,c\,\frac{\log^2 n}{\log\log n}$ with probability less than $n^{-dt\,/\log\log n}$, where t_0 and d are positive constants.

Proof: At a given step, if the candidates set size s satisfies $\frac{n}{p^i} \geqslant s > \frac{n}{p^{i+1}} > 0$, then we say the algorithm is in state $i+1$, and if $s = 0$ then we say the algorithm is in state $m = 1 + \left\lfloor \frac{p}{\log p} \right\rfloor$. The algorithm starts in state 1 and terminates in state m. Let T be the running time of the algorithm. We observe the state of the algorithm just before the odd stages (1,3,5,...). By Lemma 1, if the algorithm is in state u just before the i-th stage then, with probability $> 1 - \frac{2}{e}$, it will be in some state $v < u$ just before the $(i+2)$-nd stage. Define a new random variable W to be the number of coin tosses that elapse before obtaining m heads, where each toss results in heads independently with probability $1-\frac{2}{e}$. It follows that the running time is dominated by $8pW$, in the sense that $Prob(8pW \geqslant z) \geqslant Prob(T \geqslant z)$, for all $z \geqslant 0$. (The factor of $8p$ accounts for the cost of two consecutive stages.) The expected value of W is $m \frac{e}{e-2}$, so the expected running time is at most

$$8pm \frac{e}{e-2} < c \frac{\log^2 n}{\operatorname{loglog} n},$$

for some constant $c > 0$. This establishes the first part of the theorem.

The second part follows from a standard result from the theory of large deviations (cf. [3], page 17). Let $B(\alpha, N, q)$ denote the probability of α or more tails in N coin tosses, where each toss results in tails independently with probability q:

$$B(\alpha, N, q) = \sum_{\alpha \leqslant i \leqslant N} \binom{N}{i} q^i (1-q)^{N-i}$$

Then for any β, with $0 < \beta < 1$,

$$B(Nq(1+\beta), N, q) \leqslant e^{-Nq\beta^2/3}.$$

Returning to our problem, $W \geqslant mt$ means there are $mt - m$ or more tails in mt tosses, an event with probability $B((1-\frac{1}{t})mt, mt, \frac{2}{e})$. For sufficiently small β and sufficiently large t_0, if $t > t_0$, then $(1 - \frac{1}{t}) > \frac{2}{e}(1 + \beta)$, whence

$$B((1-\frac{1}{t})mt, mt, \frac{2}{e}) \leqslant B(mt\frac{2}{e}(1+\beta), mt, \frac{2}{e})$$
$$\leqslant e^{-2mt\beta^2 / (3e)} < n^{-dt / \text{loglog } n},$$

for some constant $d > 0$. The running exceeds $8pmt$ with probability at most

$$Prob(W \geqslant mt) < n^{-dt / \text{loglog } n}.$$

Since $8pmt < 8\frac{e}{e-2}pmt < c\ t\ \frac{\log^2 n}{\text{loglog } n}$, this completes the proof. ■

3. IMPROVEMENTS AND SIMULATION RESULTS

In this section we first describe some modifications to the basic pipeline algorithm. Theorem 1 remains true under these modifications. Second, we present results from a simulation study indicating the modifications lead to an improvement of about 15% in the running time's expected value and at least that much in its standard deviation. On analyzing the data using the method of least squares, we found that the improvement may be more dramatic for large n. Specifically, the results indicate that the expected running time of the basic pipeline algorithm is $\sim 4.53\frac{\log^2 n}{\text{loglog} n}$ and that of the improved pipeline algorithm is $\sim 2.75\frac{\log^2 n}{\text{loglog} n}$.

It happens that the splitter selection and rank computation operations can be overlapped so that the same group of messages used to accumulate the rank of one splitter can also accomplish the picking of another. In the basic algorithm, messages of the form (v,l,e) combine to accumulate the rank of an input, and messages of the form (w,s) combine to select a new splitter. In the improved algorithm, all messages except the start message are of the form $[(v,l,e),(w,s)]$, that is, all messages have a rank computation component and a splitter selection component. We obtain the following *improved pipeline* algorithm by combining parts of the basic algorithm.

1. [Internal node, not the root] An internal node may receive two messages, $[(v, l_1, e_1), (v_1, s_1)]$ and $[(v, l_2, e_2), (v_2, s_2)]$, from its two sons. As a result, the node transmits to its father a message whose rank computation component is computed as a function of (v, l_1, e_1) and (v, l_2, e_2), as described in rule 1.1.a of the basic algorithm, and whose splitter selection component is computed as a function of (v_1, s_1) and (v_2, s_2), as described in rule 1.1.b of the basic algorithm.
2. [Root] Same as for the basic pipeline algorithm.
3. [Leaf] A leaf receives either the start message, no message, or a message, $[(v, l, e), (v', s)]$, from its father. In the first two cases, the leaf transmits $[(-\infty, 0, 0), (w, 1)]$, where $-\infty$ represents a special value less than the value of all inputs, and, w the value of the input stored in the leaf in question. In the second case, the leaf transmits the message whose rank computation component is determined as a function of (v, l, e), as described in rule 3.b of the basic algorithm, and whose splitter selection component is determined as a function of (v', s), as described in rule 3.c of the basic algorithm.

As mentioned above, Theorem 1 holds (with a similar proof) for this algorithm. Thus, its running time is less than $c\frac{\log^2 n}{\log\log n}$ with high probability, for some constant $c > 0$.

We do not have a precise analysis of the moments of the running time for any of the algorithms considered here: the basic pipeline algorithm, the improved pipeline algorithm, or the straightforward algorithm. Therefore, we simulated the three algorithms to determine which is better for a range of practical values of n. (In simulating the straightforward algorithm, we used the splitter selection technique described here because it is simpler and never slower than the one proposed in [6].) In the simulations, each leaf processor was initialized with a input value chosen uniformly at random over a large range, so that the expected number of duplicates was small. Simulation runs on data in which all input values were distinct gave nearly identical results.

For each algorithm and n we measured the running time averaged over 200 trials. Table 1 displays the average running time results collected for n = 32, 64, 128, ..., 2048. Figure 3 gives plots of the same data. The data indicates that the improved algorithm is more than 15% more efficient than the basic algorithm. Using the method of least squares, we fit the data for the pipeline algorithms to the expression

$$a \frac{\log^2 n}{\log\log n} + b \log n + c$$

and solved for the real parameters a, b, and c. Similarly for the algorithm of Shrira et al., we fit the data to

$$a \log^2 n + b \log n + c.$$

The dominant parameter a turned out to be 2.75 for the improved pipeline algorithm, 4.53 for the basic pipeline algorithm, and 9.12 for the straightforward algorithm. The smooth curves plotted along with the data correspond to these fitted expressions. We also measured the standard deviation of the running time, and collected the data presented in Table 1.

TABLE 1. Expected Value of the Running Times

n	Improved Pipeline	Basic Pipeline	Straightforward
32	25.73	30.31	105.30
64	33.66	40.96	170.16
128	44.06	50.86	228.06
256	52.14	60.85	308.64
512	63.42	74.72	398.16
1024	77.80	87.47	487.20
2048	86.54	103.86	590.92

TABLE 2. Standard Deviation of the Running Times

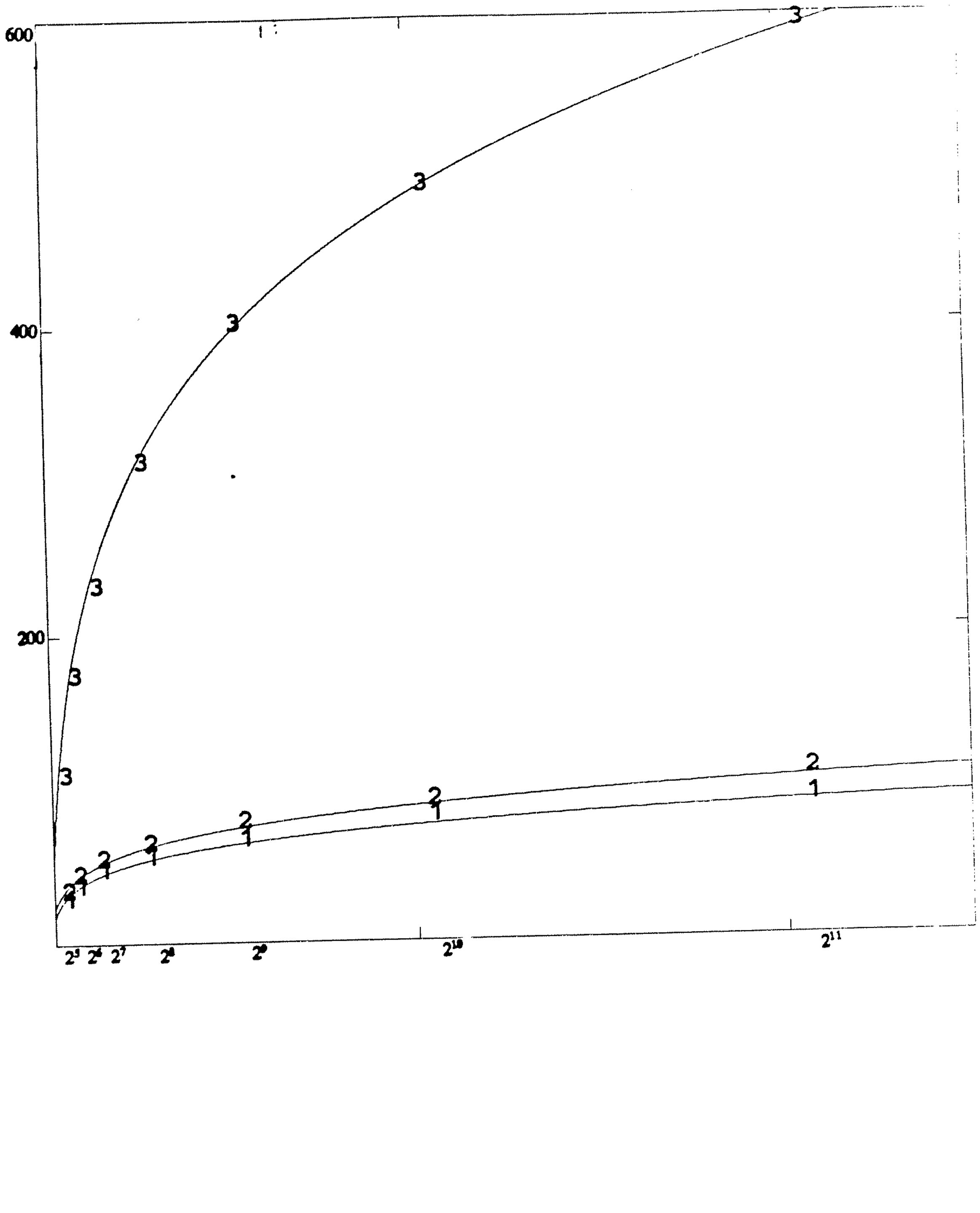
n	Improved Pipeline	Basic Pipeline	Straightforward
32	7.68	10.34	40.60
64	9.65	13.15	62.11
128	11.44	13.95	73.19
256	12.68	16.00	104.42
512	14.04	17.45	124.62
1024	14.85	21.95	139.81
2048	16.34	23.40	152.66

4. Conclusions

We presented a new probabilistic algorithm for k-selection for the tree machine, which runs in time less than $c\frac{\log^2 n}{\log\log n}$ with high probability, where c is a small constant. Analytic and simulation results indicate that the algorithm is significantly faster than ones reported previously.

We note that the algorithm can be used to sort n inputs, initially distributed among the n leaf processors. (The task is to rearrange the inputs so that the value stored at the i-th leaf is the i-th largest in a non-decreasing ordering of the input values.) Sorting on the tree machine requires $\Omega(n)$ time in the worst case, and can be accomplished in $O(n)$ time by several methods. With our k-selection algorithm, it is possible to sort in expected time $m + O(\frac{\log^3 n}{\log\log n})$, where m is the number of input values initially out of order. Without going into the details, the idea is as follows. Compute the median and broadcast it to the leaves. Identify the values in the left subtree that should go to the right subtree, and the values in the right subtree that should go to the left subtree. Move the values to the appropriate subtree, and recurse in parallel in the two subtrees. There are several opportunities for pipelining (for example, in the move operation) and other improvements.

Figure 3. Expected Value of the Running Times: Data points obtained from simulating the improved pipeline algorithm are plotted as 1's, those from the basic pipeline algorithm as 2's, and those from the algorithm of Shrira et al. as 3's.



REFERENCES

[1] Bhatt, S.N. and Leiserson,C.E., "How to Assemble Tree Machines (extended abstract)" *Proceedings of the 14-th Annual ACM Symposium on Theory of Computing,* (May 1982), San Francisco.

[2] Browning, S. *The Tree Machine: A Highly Concurrent Computing Environment,* Ph.D. Thesis, Dept. of Computer Science, California Institute of Technology (1980).

[3] Erdos, P. and Spencer, J., *Probabilistic Methods in Combinatorics,* (1974) Academic Press.

[4] Fredrickson, G., "Tradeoffs for Selection in Distributed Networks" (preliminary version), *ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing* (August 1983), Ottawa.

[5] Munro, J.I. and Paterson, M.S., "Selection and Sorting with Limited Storage", *Theoretical Computer Science* 12 (1980), 315-323.

[6] Shrira, L., Francez, N., and Rodeh, M., "Distributed k-Selection: From a Sequential to a Distributed Algorithm", *ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing* (August 1983), Ottawa.

[7] Stolfo, S. J. and Miranker, D. P. "DADO: A Parallel Processor for Expert Systems," *Proceedings of the International Symposium on Parallel Processing* (August 1984), Bellaire, MI.

[8] Tanimoto, S., *Computing Structures for Image Processing,* Academic Press (1983), London. (See Chapter 9: "Algorithms for Median Filtering of Images on a Pyramid Machine".)