An Improved Randomized Selection Algorithm With an Experimental Study

David A. Bader
University of New Mexico

A common statistical problem is that of finding the median element in a set of data. This paper presents an efficient randomized high-level parallel algorithm for finding the median given a set of elements distributed across a parallel machine. In fact, our algorithm solves the general selection problem that requires the determination of the element of rank k, for an arbitrarily given integer k.

Our general framework is an SPMD distributed memory programming model that is enhanced by a set of communication primitives. We use efficient techniques for distributing and coalescing data as well as efficient combinations of task and data parallelism. The algorithms have been coded in the message passing standard MPI, and our experimental results from the IBM SP-2 illustrate the scalability and efficiency of our algorithm and improve upon all the related experimental results known to the author.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems

General Terms: Selection Algorithm, Randomized Algorithms, Parallel Algorithms, Experimental Parallel Algorithmics

1. INTRODUCTION

Selection and median finding in large data sets are important statistical measures needed by a variety of high-performance computing applications, for example, image processing for computer vision and remote sensing, computational aerodynamics and physics simulations, and data mining of large databases. In these applications, the data set typically is already evenly distributed across the processing nodes. Because of the large data volume, solving the problem sequentially surely would overwhelm a single processor.

Given a set of data X with $|X| = n$, the selection problem requires the determination of the element with rank k (that is, the k^{th} smallest element), for an arbitrarily given integer k. Median finding is a special case of selection with $k = \frac{n}{2}$. In previous work, we have designed deterministic and efficient parallel algorithms for the selection problem on current parallel machines [Bader and JáJá 1995; Bader and JáJá 1996; Bader 1996]. In this paper, we discuss a new UltraFast Randomized algorithm for the selection problem

Address: Department of Electrical and Computer Engineering, University of New Mexico, Albuquerque, NM 87131. Email: dbader@eece.unm.edu

This work was supported in part by NSF CISE Postdoctoral Research Associate in Experimental Computer Science No. 96-25668 and U.S. Department of Energy Sandia-University New Assistant Professorship Program (SUNAPP) Award # AX-3006. This research, in part conducted at the Maui High Performance Computing Center, was sponsored in part by the Air Force Research Laboratory, Air Force Materiel Command, USAF, under cooperative agreement number F29601-93-2-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory, the U.S. Government, The University of New Mexico, or the Maui High Performance Computing Center.

which, unlike previous research (for example, [Hao et al. 1992; Krizanc and Narayanan 1992; Rajasekaran and Reif 1993; Berthomé et al. 1993; Rajasekaran et al. 1994; Rajasekaran 1996; Rajasekaran and Sahni 1997; Sarnath and He 1997; Rajasekaran and Wei 1997; Rajasekaran and Sahni 1998]), is not dependent on network topology or limited to the PRAM model which does not assign a realistic cost for communication. In addition, our randomized algorithm improves upon previous implementations on current parallel platforms, for example, [Al-furiah et al. 1997] implements both our deterministic algorithm and the randomized algorithms due to Rajasekaran et al. (e.g., [Rajasekaran and Reif 1993; Rajasekaran 1996]) on the TMC CM-5.

The main contributions of this paper are

(1) New techniques for speeding the performance of certain randomized algorithms, such as selection, which are efficient with likely probability.
(2) A new, practical randomized selection algorithm (UltraFast) with significantly improved convergence.

The remainder of this paper is organized as follows. Both our new and Rajasekaran's randomized selection algorithms are detailed in Section 2, followed by analysis and experimental results in Section 3. Additional information on Chernoff Bounds is located in Appendix A. More extensive statistics from our experiments are reported in [Bader 1999].

2. PARALLEL SELECTION

The selection algorithm for rank k assumes that input data X of size n is initially distributed evenly across the p processors, such that each processor holds $\frac{n}{p}$ elements. Note that median finding is a special case of the selection problem where k is equal to $\lceil \frac{n}{2} \rceil$. The output, namely the element from X with rank k, is returned on each processor.

The randomized selection algorithm locates the element of rank k by pruning the set of candidate elements using the following iterative procedure. Two *splitter* elements (k_1, k_2) are chosen which partition the input into three groups, G_0, G_1, and G_2, such that each element in G_0 is less than k_1, each element in G_1 lies in $[k_1, k_2]$, and each in G_2 is greater than k_2. The desire is to have the middle group G_1 much smaller than the outer two groups $(|G_1| \ll |G_0|, |G_2|)$ with the *condition* that the selection index lies within this middle group. The process is repeated iteratively on the group holding the selection index until the size of the group is "small enough," whereby the remaining elements are gathered onto a single processor and the problem is solved sequentially.

The key to this approach is choosing splitters k_1 and k_2 which minimize the size of the middle group while maximizing the probability of the *condition* that the selection index lies within this group. Splitters are chosen from a random sample of the input, by finding a pair of elements of certain rank in the sample (see Section 3). The algorithm of Rajasekaran and Reif (see [Rajasekaran and Reif 1993; Rajasekaran 1996]) takes a conservative approach which guarantees the condition with high probability. We have discovered a more aggressive technique for pruning the input space by choosing splitters closer together in the sample while holding the condition with likely probability. In practice, the condition almost always holds, and in the event of a failure, new splitters are chosen from the sample with a greater spread of ranks until the condition is satisfied.

In addition, we improve upon previous algorithms in the following ways.

(1) **Stopping Criterion**. For utmost performance, current parallel machines typically require a coarse granularity, the measure of problem size per node, because communication is typically an order of magnitude slower than local computation. In addition, machine configurations tend to be small to moderate in terms of number of processors (p). Thus, a stopping criterion of problem size $< p^2$ is much too fine grained for current machines, and we suggest, for instance, a stopping size of $\max(p^2, 4096)$. When p is small and $n = \mathrm{O}(p^2)$, a second practical reason for increasing the stopping size is that the sample is very limited and might not yield splitters which further partition the input.

(2) **Aggressive Convergence**. As outlined in Section 3, our algorithm converges roughly twice as fast as the best known previous algorithm.

(3) **Algorithmic Reduction**. At each iteration, we use "selection" to choose the splitters instead of sorting, a computationally harder problem.

(4) **Communication Aggregation**. Similar collective communication steps are merged into a single operation. For instance, instead of calling the **Combine** primitive twice to find the size of groups G_0 and G_1 ($|G_2|$ can be calculated from this information and the problem size), we aggregate these operations into a single step.

(5) **"Min/Max" Selection Algorithm**. When the selection index is relatively close to 1 or n, our approach switches to a faster algorithm for this special case.

Next we outline our new UltraFast Randomized Selection Algorithm, followed by the Fast Randomized algorithm.

2.1 UltraFast Randomized Selection Algorithm

An SPMD algorithm on each processor P_i:

ALGORITHM 1. *UltraFast Randomized Selection Algorithm*

Input:

$\{ n \}$ - Total number of elements
$\{ p \}$ - Total number of processors, labeled from 0 to $p-1$
$\{ L_i \}$ - List of elements on processor P_i, where $|L_i| = \frac{n}{p}$
$\{ C \}$ - A constant $\approx \max(p^2, 4096)$
$\{ \varepsilon \}$ - $\log_n$ of the sample size (e.g. 0.6)
$\{ \Delta^* \}$ - selection coefficient (e.g. 1.0)
$\{ \kappa \}$ - selection coefficient multiplier (e.g. 2.25)
$\{ \eta \}$ - Min/Max constant (e.g. $2p$)
rank - desired rank among the elements

begin

Step 0. Set $n_i = \frac{n}{p}$.
While $(n > C)$ and $(|n - rank| > \eta)$
Step 1. Collect a sample S_i from L_i by picking $n_i \frac{n^\varepsilon}{n}$ elements at random on P_i.
Step 2. $S =$ **Gather**(S_i, p).
Set $z =$ TRUE and $\Delta = \Delta^*$.
While $(z \equiv$ TRUE$)$
On P_0
Step 3. Select k_1, k_2 from S with ranks $\left\lfloor \frac{i|S|}{n} - \Delta\sqrt{|S|} \right\rfloor$ and $\left\lfloor \frac{i|S|}{n} + \Delta\sqrt{|S|} \right\rfloor$.
Step 4. Broadcast k_1 and k_2.
Step 5. Partition L_i into $< k_1$ and $[k_1, k_2]$, and $> k_2$, to give counts *less*, *middle*, (and *high*). Only save the elements which lie in the middle partition.
Step 6. $c_{less} =$ **Combine**$(less, +)$; $c_{mid} =$ **Combine**$(middle, +)$;
Step 7. If $(rank \in (c_{less}\ ,\ c_{less} + c_{mid}]\ )$
$n = c_{mid}$; $n_i = middle$; $rank = rank - c_{less}$; $z =$ FALSE
Else
On P_0: $\Delta = \kappa \cdot \Delta$
Endif
Endwhile
Endwhile
If $(|n - rank| \leq \eta)$ then
If $rank \leq \eta$ then we use the "minimum" approach, otherwise, we use the "maximum" approach in parentheses, as follows.
Step 8. Sequentially sort our n_i elements in nondecreasing (nonincreasing) order using a modified insertion sort with output size $|L_i| = \min(rank, n_i)$ $(|L_i| = \min(n - rank + 1, n_i))$. An element that is greater (less) than the L_i minimum (maximum) elements is discarded.
Step 9. Gather the p sorted subsequences onto P_0.
Step 10. Using a p-way tournament tree of losers [Horowitz and Sahni 1978] constructed from the p sorted subsequences, $rank$ $(n - rank + 1)$ elements are extracted, to find the element q with selection index *rank*.
Else
Step 11. $L =$ **Gather**(L_i).
Step 12. On P_0
Perform sequential selection to find element q of *rank* in L;
Endif
result = **Broadcast**(q).

end

2.2 Fast Randomized Selection Algorithm

This algorithm is due to Rajasekaran and Reif (see [Rajasekaran and Reif 1993; Rajasekaran 1996]), and implemented in [Al-furiah et al. 1997].
An SPMD algorithm on each processor P_i:

ALGORITHM 2. *Fast Randomized Selection Algorithm*

Input:

$\{ n \}$ - Total number of elements
$\{ p \}$ - Total number of processors, labeled from 0 to $p-1$
$\{ L_i \}$ - List of elements on processor P_i, where $|L_i| = \frac{n}{p}$
$\{ \varepsilon \}$ - $\log_n$ of the sample size (e.g. 0.6)
rank - desired rank among the elements
$l = 0$; $r = \frac{n}{p} - 1$

begin

while $(n > p^2)$

Step 0. Set $n_i = r - l + 1$

Step 1. Collect a sample S_i from $L_i[l,r]$ by picking $n_i \frac{n^\varepsilon}{n}$ elements at random on P_i between l and r.

Step 2. $S =$ **ParallelSort**(S_i, p).

On P_0

Step 3. Pick k_1, k_2 from S with ranks $\left\lceil \frac{i|S|}{n} - \sqrt{|S| \ln n} \right\rceil$ and $\left\lceil \frac{i|S|}{n} + \sqrt{|S| \ln n} \right\rceil$.

Step 4. Broadcast k_1 and k_2. The *rank* to be found will be in $[k_1, k_2]$ with high probability.

Step 5. Partition L_i between l and r into $< k_1$, $[k_1, k_2]$, and $> k_2$ to give counts *less*, *middle*, and *high*, and splitters s_0 and s_1.

Step 6. $c_{mid} =$ **Combine**$(middle, +)$.

Step 7. $c_{less} =$ **Combine**$(less, +)$.

Step 8. If $(rank \in (c_{less}, c_{mid}])$

$n = c_{mid}$; $l = s_1$; $r = s_2$; $rank = rank - c_{less}$

Else

If $(rank \leq c_{less})$

$r = s_1$; $n = c_{less}$

Else

$n = n - (c_{less} + c_{mid})$; $l = s_2$; $rank = rank - (c_{less} + c_{mid})$

Endif

Endif

Endwhile

Step 9. $L =$ **Gather**$(L_i[l,r])$.

Step 10. On P_0

Perform sequential selection to find element q of *rank* in L,
$result =$ **Broadcast**(q).

end

3. ANALYSIS

The following sampling lemma from Rajasekaran (see [Rajasekaran and Reif 1993]) will be used in the analysis.

Let $S = \{v_1, v_2, \ldots, v_s\}$ be a random sample from a set X of cardinality n. Also, let $v'_1, v'_2, \ldots, v'_s$ be the sorted order of this sample. If r_i is the rank of k'_i in X, the following lemma provides a high probability confidence interval for r_i.

LEMMA 1. *For every* α, $Pr\left(|r_i - i\frac{n}{s}| > \sqrt{3\alpha}\frac{n}{\sqrt{s}}\sqrt{\ln n}\right) < n^{-\alpha}$.

Thus, if k_1 and k_2 are chosen as the splitters from sample set S by selecting the elements with rank $\frac{is}{n} - d\sqrt{s \ln n}$ and $\frac{is}{n} + d\sqrt{s \ln n}$, respectively, and $d = \sqrt{4\alpha}$, then the element of desired rank will lie in the middle partition $(c_{less}\ ,\ c_{less} + c_{mid}]$ with high probability $(1 - n^{-\alpha})$.

A tradeoff occurs between the size of the middle partition (r) and the confidence that the desired element lies within this partition. Note that in the Fast Randomized algorithm, with $d = 1$, this probability is $1 - n^{-\frac{1}{4}}$, and $r \leq 8\frac{n}{\sqrt{s}}\sqrt{\ln n}$. Since $s \approx n^{\varepsilon}$, this can be approximated by $r \leq 8n^{1-\frac{\varepsilon}{2}}\sqrt{\ln n}$.

Suppose now the bound is relaxed with probability no less than $1 - n^{-\alpha} = \rho$. Then $\alpha = -\frac{\log(1-\rho)}{\log n}$, and the splitters k_1, k_2 can be chosen with ranks $\frac{is}{n} - \Delta\sqrt{s}$ and $\frac{is}{n} + \Delta\sqrt{s}$, for $\Delta = 2\sqrt{-\ln(1-\rho)}$ (see Table I). Then the size of the middle partition can be bounded similarly by $r \leq 16\frac{n}{\sqrt{s}}\sqrt{-\ln(1-\rho)}$. This can be approximated by $r \leq 16n^{1-\frac{\varepsilon}{2}}\sqrt{-\ln(1-\rho)}$. Thus, the middle partition size of the UltraFast algorithm is typically smaller than that of the Fast algorithm, whenever the condition $n > (1-\rho)^{-4}$.

Δ	Lower bound of capture (ρ, in %)
6.07	99.99
5.26	99.9
4.29	99.0
3.03	90.0
2.54	80.0
2.19	70.0
1.91	60.0
1.50	43.0
1.00	22.1
0.50	6.05

Table I. Lower bound of the capture probability (ρ) that the selection index is in the middle partition, where $\rho = 1 - e^{-\frac{\Delta^2}{4}}$.

A large value for ε increases running time since the sample (of size n^{ε}) must be either sorted (in Fast) or have elements selected from it (in UltraFast). A small value of ε increases the probability that both of the splitters lie on one side of the desired element, thus causing an unsuccessful iteration. In practice, 0.6 is an appropriate value for ε [Al-furiah et al. 1997].

3.1 Complexity

We use a simple model of parallel computation to analyze the performance of these two selection algorithms. Current hardware platforms can be viewed as a collection of powerful

processors connected by a communication network that can be modeled as a complete graph on which communication is subject to the restrictions imposed by the latency and the bandwidth properties of the network. We view a parallel algorithm as a sequence of local computations interleaved with communication steps, and we allow computation and communication to overlap. We account for communication costs as follows.

The transfer of a block consisting of m contiguous words, assuming no congestion, takes $O(\tau+\sigma m)$ time, where τ is an bound on the latency of the network and σ is the time per word at which a processor can inject or receive data from the network.

One iteration of the Fast randomized selection algorithm takes $O\left(n^{(j)}+(\tau+\sigma)\log p\right)$ time, where $n^{(j)}$ is the maximum number of elements held by any processor during iteration j. ¿From the bound on the size of the middle partition, we find a recurrence on the problem size during iteration i,

$$\begin{aligned} n_0 &= n \\ n_{i+1} &\leq 8n_i^{0.7}\sqrt{\ln n_i}\ , \end{aligned} \tag{1}$$

which shows a geometric decrease in problem size per iteration, and thus, $O(\log\log n)$ iterations are required. Since $n^{(j)}=O\left(\frac{n}{p}\right)$, Fast selection requires

$$O\left(\frac{n}{p}\log\log n+(\tau+\sigma)\log p\log\log n\right) \tag{2}$$

time. (Assuming random data distribution, the running time reduces to $O\left(\frac{n}{p}+(\tau+\sigma)\log p\log\log n\right)$.) [Al-furiah et al. 1997]

Each iteration of the UltraFast algorithm is similar to Fast, except sorting is replaced by sequential selection, which takes linear time [Blum et al. 1973]. Also, the problem size during iteration i is bounded with the following recurrence,

$$\begin{aligned} n_0 &= n \\ n_{i+1} &\leq 16n_i^{0.7}\sqrt{-\ln(1-\rho)}\ , \end{aligned} \tag{3}$$

and similar to the Fast algorithm, UltraFast as well requires $O(\log\log n)$ iterations. Thus, UltraFast randomized selection has a similar complexity, with a worst case running time given in Eq. (2). As we will show later by empirical results in Table III, though, the constant associated with the number of iterations is significantly smaller for the UltraFast algorithm.

3.2 Experimental Data Sets

Empirical results for the selection algorithm use the following five input classes. Given a problem of size n and p processors,

—**[I]** - Identical elements $\{0,1,\ldots,\frac{n}{p}-1\}$ on each processor.

—**[S]** - Sorted elements $\{0,1,\ldots,n-1\}$ distributed in p blocks across the processors.

—**[R]** - Random, uniformly-distributed, elements, with $\frac{n}{p}$ elements per processor.

—**[N]** - This input is taken from the NAS Parallel Benchmark for Integer Sorting [Bailey et al. 1994]. Keys are integers in the range $[0,2^{19})$, and each key is the average of four consecutive uniformly-distributed pseudo-random numbers generated by the following recurrence:

$$x_{k+1}=ax_k\ (\mathrm{mod}\ 2^{46})$$

where $a = 5^{13}$ and the seed $x_0 = 314159265$. Thus, the distribution of the key values is a Gaussian approximation. On a p-processor machine, the first $\frac{n}{p}$ generated keys are assigned to P_0, the next $\frac{n}{p}$ to P_1, and so forth, until each processor has $\frac{n}{p}$ keys.

—**[K]** - This input contains $\frac{n}{p}$ randomly generated elements per processor, sampled from the skewed log-normal distribution[1], in the range of positive integers $[1, \text{INTMAX}]$ (where INTMAX, for example, is $2^{31}-1$ on a 32-bit machine). We generate each pseudo-random integer $\left(\left\lfloor \exp\left(\frac{1}{12}\ln \text{INTMAX} \cdot \text{normRand}(0,1) + \frac{1}{2}\ln \text{INTMAX}\right)\right\rfloor\right)$ by taking the largest integer less than or equal to the exponential of a mean 0, standard deviation 1 Gaussian random number (found by adding together twelve uniformly-distributed random numbers from the range $[-0.5, 0.5)$) that is first scaled by $\frac{1}{12}\ln \text{INTMAX}$ and then displaced to the right by $\frac{1}{2}\ln \text{INTMAX}$. For a given INTMAX, the mean and standard deviation of this skewed distribution are computable[2].

3.3 Empirical Results

Results for a previous implementation of the Fast randomized selection algorithm on the TMC CM-5 parallel machine appear in [Al-furiah et al. 1997]. However, this machine is no longer available and does not support the current message passing standard **MPI**. Therefore, we have recoded this algorithm into MPI.

n	p	[R]andom Input			[S]orted Input		
		CM-5 33	SP-2 66 P2	SP-2 160 P2SC	CM-5 33	SP-2 66 P2	SP-2 160 P2SC
512K	4	174	68.0	23.5	194	104	25.6
	8	105	62.7	17.2	119	79.6	21.7
	16	69.5	39.5	10.8	86.7	61.9	15.6
2M	4	591	153	56.6	601	229	67.3
	8	318	108	37.6	359	182	48.0
	16	193	74.4	23.7	237	136	34.6

Table II. Comparison of the execution time of the Fast Randomized Selection Algorithm on TMC CM-5 [Al-Furaih 1996; Al-furiah et al. 1997] and IBM SP-2-TN (in milliseconds).

Table II compares the execution time of the Fast Randomized algorithm on both the CM-5 [Al-Furaih 1996; Al-furiah et al. 1997] and the IBM SP-2. Since selection is computation-bound, we would expect the performance to be closely related to the node performance of these two machines. The SP-2-TN 66MHz Power2 (66-P2) processor is roughly twice as fast as the CM-5 33 MHz RISC processor. As expected, this factor of two performance improvement is apparent in the execution time comparison for equivalent machine and problem sizes. In actuality, the SP-2 is more than twice as powerful, since

[1]The log-normal is a distribution whose natural logarithm is a normal distribution. Given a normal distribution with mean μ and standard deviation σ, the log-normal distribution $\exp(\text{norm}(\mu,\sigma))$ has mean $e^{\mu+\sigma^2/2}$ and variance $e^{2\mu+\sigma^2}\left(e^{\sigma^2}-1\right)$.

[2]For our generator, a log-normal distribution with mean μ and standard deviation σ, the scale ($\frac{1}{12}\ln \text{INTMAX}$) of the mean 0, s.d. 1, Gaussian random number equals $\sqrt{\ln\left(\frac{\mu^2+\sigma^2}{\mu^2}\right)}$, and the displacement ($\frac{1}{2}\ln \text{INTMAX}$) equals $\frac{1}{2}\ln\left(\frac{\mu^4}{\mu^2+\sigma^2}\right)$.

communication latency and bandwidth are improved roughly by a factor of three. The newer SP-2-TN 160MHz Power2 SuperChip (160-P2SC) nodes are roughly three times faster than the 66-P2 nodes, and we see a similar performance improvement.

We conducted experiments with our UltraFast and the known Fast randomized selection algorithms on an IBM SP-2 (with 160-P2SC nodes) with four, eight, and sixteen processors, by finding the median of each input in the previous section for various problem sizes (ranging between 16*K* to 16*M* elements)[3]. A comparison of the empirical execution times for machine configurations of $p = 4, 8,$ and 16 processors are graphed using log-log plots in Figures 1-15. In all cases, the UltraFast algorithm is substantially faster than the Fast randomized selection algorithm, typically by a factor of two. Running time can be characterized mainly by $\frac{n}{p} \log p$ and is only slightly dependent on input distribution. In addition, we have included the performance of several variations as follows:

—**FR** - the Fast Randomized algorithm (Alg. (2));

—**FT** - the modified (and improved) Fast Randomized with the *while* loop stopping criterion of $n \leq \max(p^2, 4096)$ instead of $n \leq p^2$;

—**R2** - the modified UltraFast Randomized algorithm without the "Min/Max" selection improvement when $(|n - rank| \leq \eta)$; and

—**R3** - the UltraFast Randomized algorithm (Alg. (1)).

For $p = 8$, Table III provides a summary of the number of times each algorithm iterates. While the Fast algorithm typically iterates in the neighborhood of about 25 times, there are some cases when it iterates hundreds or even thousands of times. For some other problem instances, the Fast algorithm may encounter an infinite loop when the number of elements in a step is larger than p^2, and no choice of splitters further partitions the elements. On the other hand, the UltraFast algorithm never iterates more then three times. This is due to two reasons. First, UltraFast converges roughly twice as fast as the Fast algorithm. Second, the algorithm stops iterating by using a more realistic stopping criterion matched to the coarse granularity of current parallel machines. In addition, when p is small and $n = \mathrm{O}(p^2)$, the Fast algorithm's sample is very limited and sometimes does not yield splitters which further partition the input. Thus, in this situation, the Fast algorithm might iterative from tens to thousands of times before pruning any additional elements from the solution space.

Detailed results from the UltraFast and Fast algorithms (for the **[I]**, **[S]**, and **[R]** inputs) for $n = 512$K, 1M, 2M, 4M, and 8M, and further statistics from the **[N]** input, are available in [Bader 1999].

[3]Throughout this paper, *K* and *M* refer to 2^{10} and 2^{20}, respectively.

n	Input	Fast Algorithm	UltraFast Algorithm
512K	I	19	2
	S	17	2
	R	29	2
	N	19	2
	K	18	2
1M	I	24	2
	S	17	2
	R	22	2
	N	32	2
	K	22	2
2M	I	26	2
	S	22	3
	R	21	2
	N	38	3
	K	20	3
4M	I	37	3
	S	23	3
	R	21	3
	N	4095	3
	K	90	3
8M	I	28	3
	S	24	3
	R	21	3
	N	866	3
	K	∞	3

Table III. Total number of iterations of the Fast and UltraFast Randomized Selection Algorithms. For this table, the number of processors used $p = 8$.

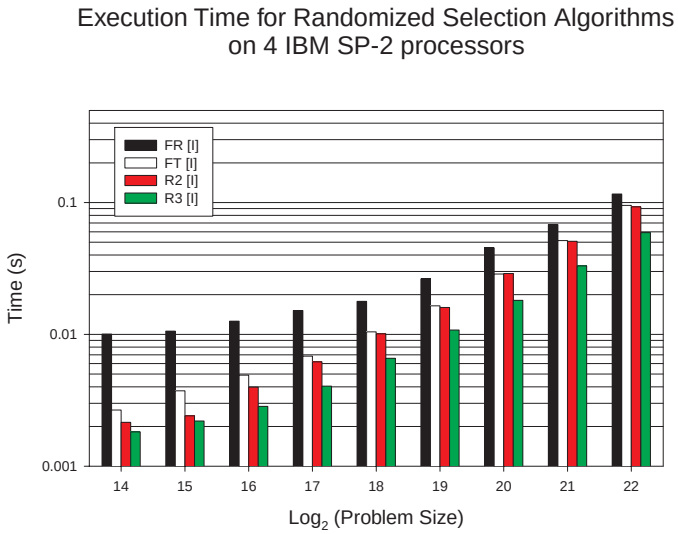

Fig. 1. Empirical Performance of Fast versus UltraFast Randomized Selection Algorithms on the [I] input class, with $p = 4$ nodes of an IBM SP-2-TN. The x-axis represents increasing problem sizes.

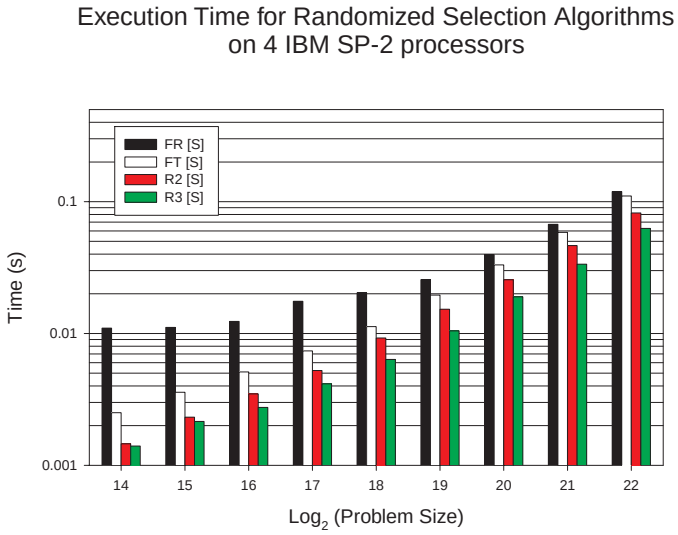

Fig. 2. Empirical Performance of Fast versus UltraFast Randomized Selection Algorithms on the [S] input class, with $p = 4$ nodes of an IBM SP-2-TN. The x-axis represents increasing problem sizes.

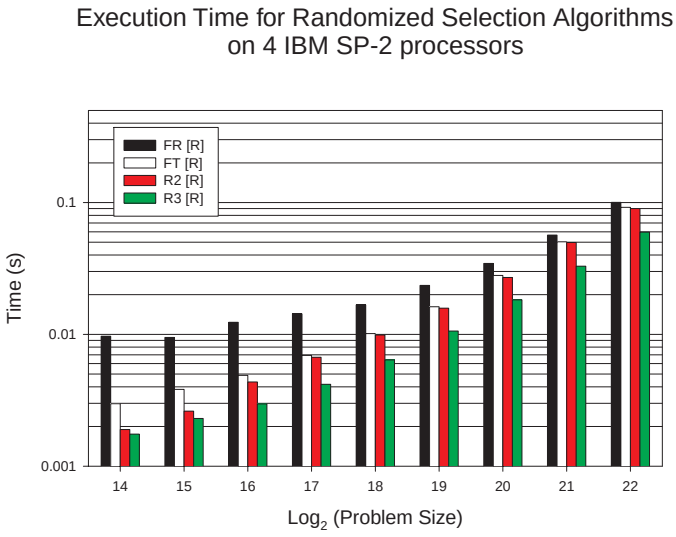

Fig. 3. Empirical Performance of Fast versus Ultra-Fast Randomized Selection Algorithms on the [R] input class, with $p = 4$ nodes of an IBM SP-2-TN. The x-axis represents increasing problem sizes.

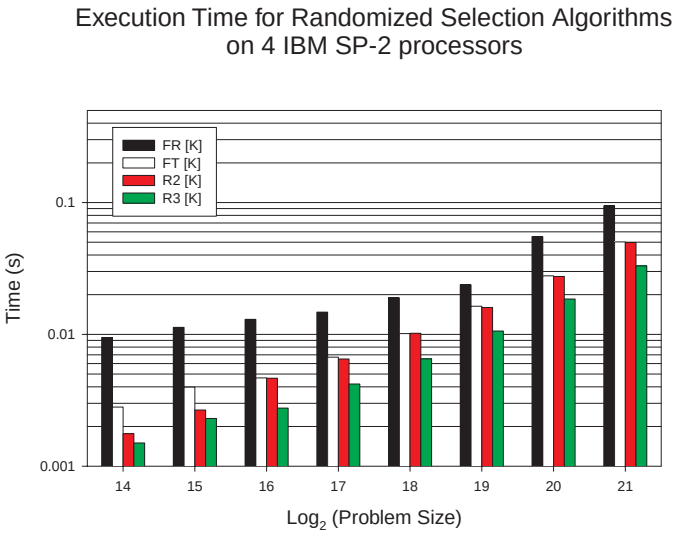

Fig. 5. Empirical Performance of Fast versus Ultra-Fast Randomized Selection Algorithms on the [K] input class, with $p = 4$ nodes of an IBM SP-2-TN. The x-axis represents increasing problem sizes.

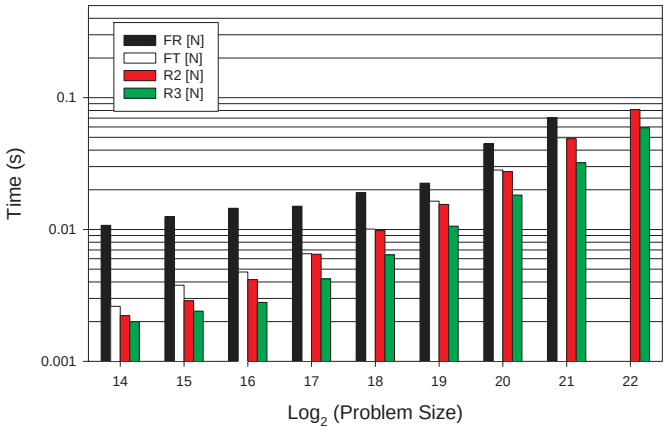

Fig. 4. Empirical Performance of Fast versus Ultra-Fast Randomized Selection Algorithms on the [N] input class, with $p = 4$ nodes of an IBM SP-2-TN. The x-axis represents increasing problem sizes.

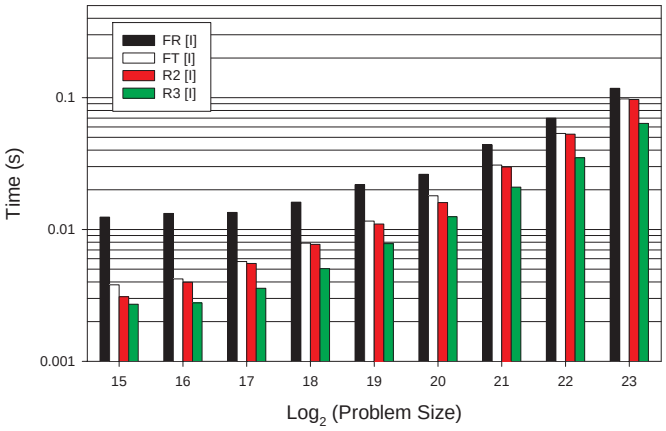

Fig. 6. Empirical Performance of Fast versus Ultra-Fast Randomized Selection Algorithms on the [I] input class, with $p = 8$ nodes of an IBM SP-2-TN. The x-axis represents increasing problem sizes.

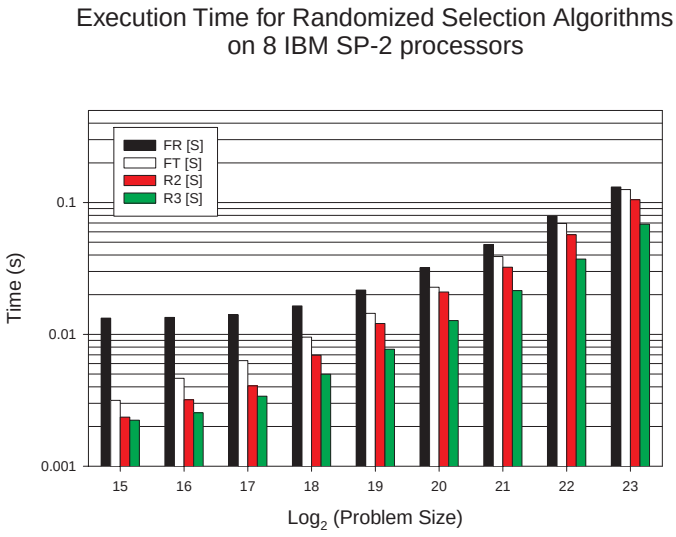

Fig. 7. Empirical Performance of Fast versus Ultra-Fast Randomized Selection Algorithms on the [S] input class, with $p = 8$ nodes of an IBM SP-2-TN. The x-axis represents increasing problem sizes.

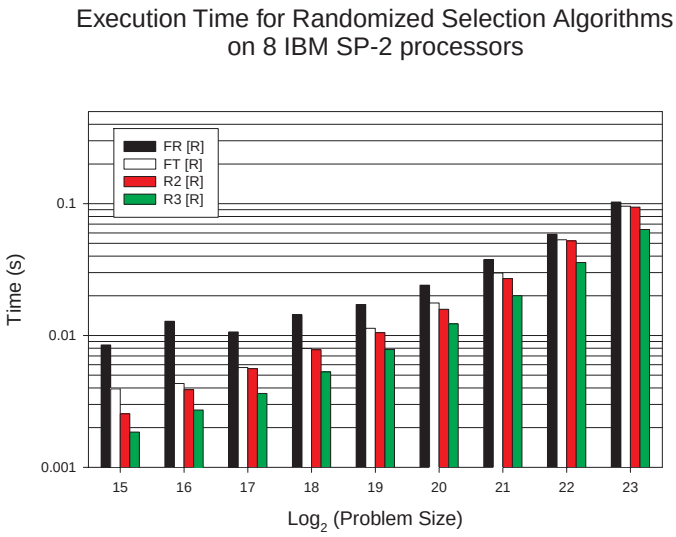

Fig. 8. Empirical Performance of Fast versus Ultra-Fast Randomized Selection Algorithms on the [R] input class, with $p = 8$ nodes of an IBM SP-2-TN. The x-axis represents increasing problem sizes.

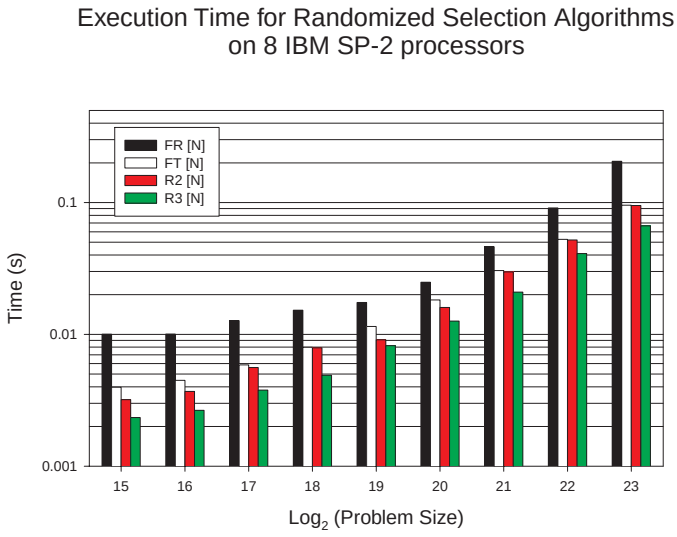

Fig. 9. Empirical Performance of Fast versus Ultra-Fast Randomized Selection Algorithms on the [N] input class, with $p = 8$ nodes of an IBM SP-2-TN. The x-axis represents increasing problem sizes.

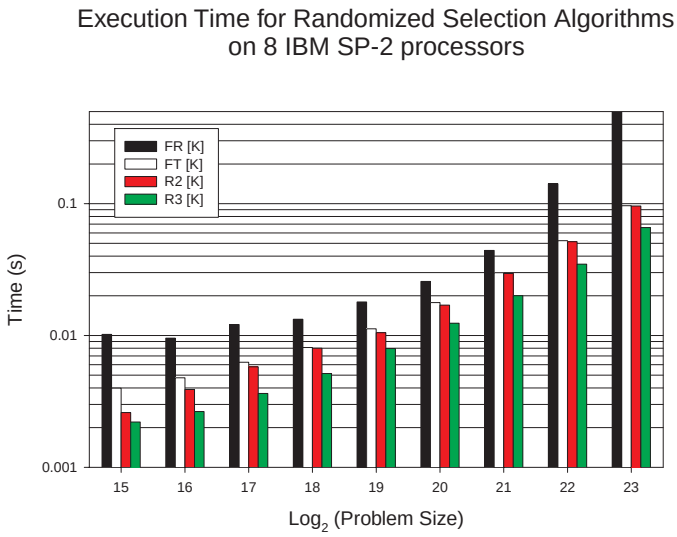

Fig. 10. Empirical Performance of Fast versus Ultra-Fast Randomized Selection Algorithms on the [K] input class, with $p = 8$ nodes of an IBM SP-2-TN. The x-axis represents increasing problem sizes.

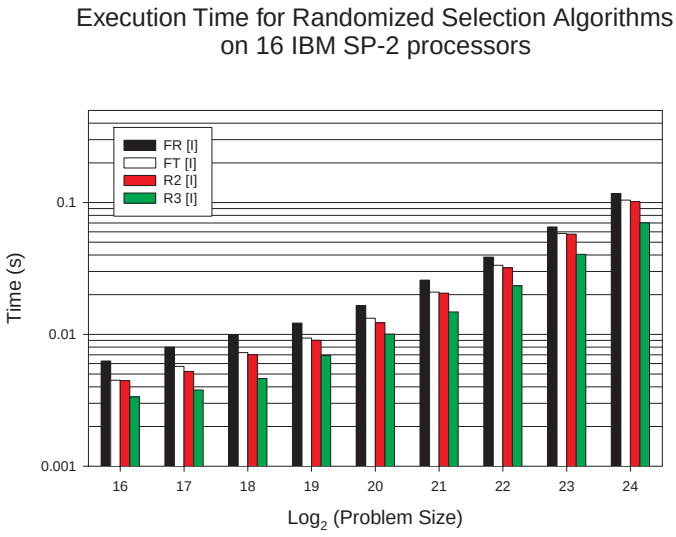

Fig. 11. Empirical Performance of Fast versus Ultra-Fast Randomized Selection Algorithms on the [I] input class, with $p = 16$ nodes of an IBM SP-2-TN. The x-axis represents increasing problem sizes.

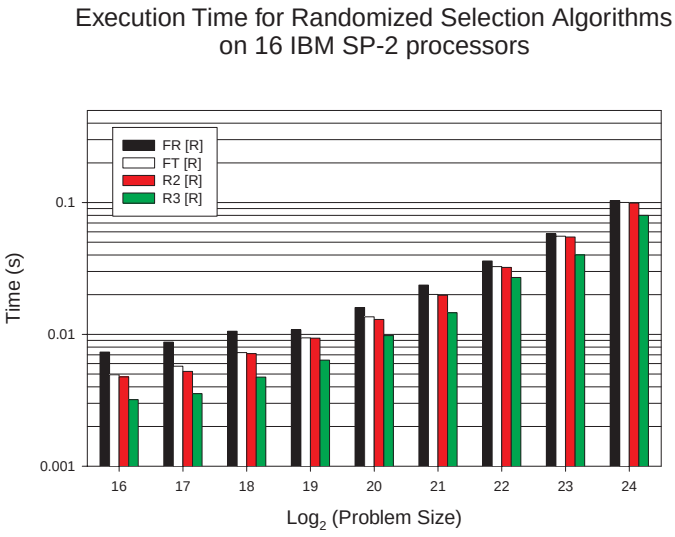

Fig. 13. Empirical Performance of Fast versus Ultra-Fast Randomized Selection Algorithms on the [R] input class, with $p = 16$ nodes of an IBM SP-2-TN. The x-axis represents increasing problem sizes.

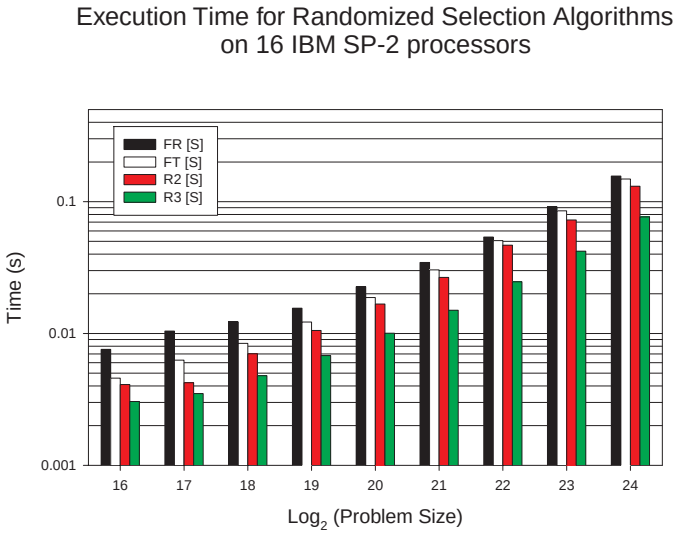

Fig. 12. Empirical Performance of Fast versus Ultra-Fast Randomized Selection Algorithms on the [S] input class, with $p = 16$ nodes of an IBM SP-2-TN. The x-axis represents increasing problem sizes.

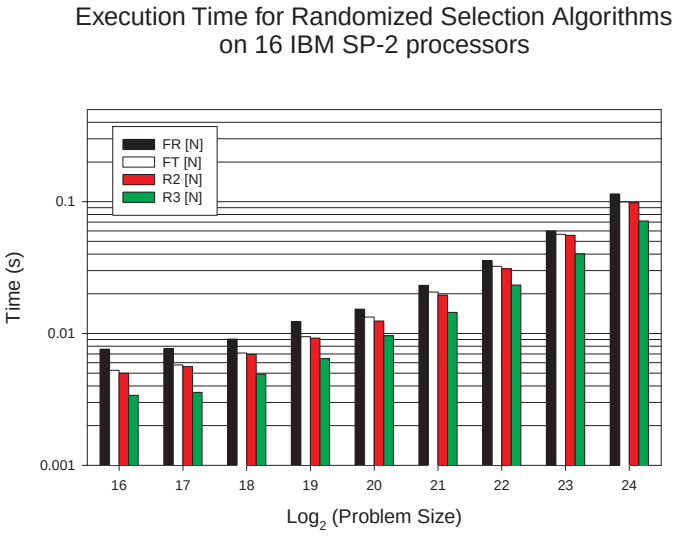

Fig. 14. Empirical Performance of Fast versus Ultra-Fast Randomized Selection Algorithms on the [N] input class, with $p = 16$ nodes of an IBM SP-2-TN. The x-axis represents increasing problem sizes.

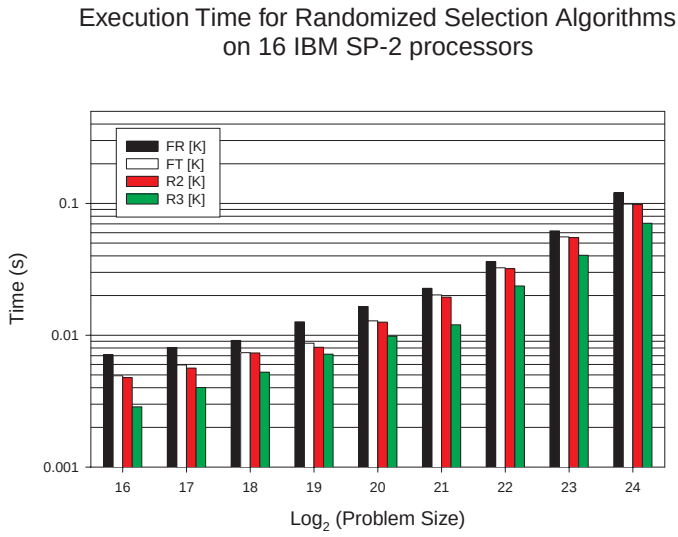

Fig. 15. Empirical Performance of Fast versus UltraFast Randomized Selection Algorithms on the [K] input class, with $p = 16$ nodes of an IBM SP-2-TN. The x-axis represents increasing problem sizes.

4. FUTURE DIRECTIONS

We are investigating other combinatorial algorithms that may have significant practical improvement by relaxing the probabilistic bounds, as demonstrated by our UltraFast randomized selection.

In addition, our UltraFast parallel, randomized selection algorithm, here designed and analyzed for a message-passing platform, would also be suitable for shared-memory multiprocessors (SMP's) and SMP Clusters [Bader and JáJá 1999]. In the SMP UltraFast selection algorithm, each communication step can be eliminated, simplified, or replaced with a shared-memory primitive. For instance, the SMP algorithm would be as follows. Each processor collects its portion of the sample from the corresponding block of the input and writes the sample to a shared-memory array. Thus, **Step 2**, the **Gather** communication, is eliminated. After a single processor determines the splitters k_1 and k_2 from the sample, the **Broadcast** communication in **Step 4** simplifies into a memory read by each processor. The **Combine** in **Step 6** may be replaced by the corresponding shared-memory primitive. The **Gather** in **Step 11** can be replaced with a shared-memory gather. We are currently investigating the performance of this SMP approach.

APPENDIX

A. CHERNOFF BOUNDS

The following inequalities are useful for bounding the tail ends of a binomial distribution with parameters (n, p). If X is a binomial with parameters (n, p), then the tail distributions, known as Chernoff bounds [Chernoff 1952], are as follows.

$$\Pr(X \leq (1-\varepsilon)np) \leq e^{-\frac{\varepsilon^2 np}{2}} \tag{4}$$

$$\Pr(X \geq (1+\varepsilon)np) \leq e^{-\frac{\varepsilon^2 np}{3}} \tag{5}$$

for all $0 < \varepsilon < 1$.

REFERENCES

AL-FURAIH, I. S. 1996. Timings of Selection Algorithm. Personal communication.

AL-FURIAH, I., ALURU, S., GOIL, S., AND RANKA, S. 1997. Practical Algorithms for Selection on Coarse-Grained Parallel Computers. *IEEE Transactions on Parallel and Distributed Systems 8*, 8, 813–824.

BADER, D. A. 1996. *On the Design and Analysis of Practical Parallel Algorithms for Combinatorial Problems with Applications to Image Processing*. Ph. D. thesis, University of Maryland, College Park, Department of Electrical Engineering.

BADER, D. A. 1999. An Improved Randomized Selection Algorithm With an Experimental Study. Technical report (September), Electrical and Computer Engineering Department, The University of New Mexico, Albuquerque, NM.

BADER, D. A. AND JÁJÁ, J. 1995. Practical Parallel Algorithms for Dynamic Data Redistribution, Median Finding, and Selection. Technical Report CS-TR-3494 and UMIACS-TR-95-74 (July), UMIACS and Electrical Engineering, University of Maryland, College Park, MD.

BADER, D. A. AND JÁJÁ, J. 1996. Practical Parallel Algorithms for Dynamic Data Redistribution, Median Finding, and Selection. In *Proceedings of the 10th International Parallel Processing Symposium* (Honolulu, HI, April 1996), pp. 292–301.

BADER, D. A. AND JÁJÁ, J. 1999. SIMPLE: A Methodology for Programming High Performance Algorithms on Clusters of Symmetric Multiprocessors (SMPs). *Journal of Parallel and Distributed Computing 58*, 1, 92–108.

BAILEY, D., BARSZCZ, E., BARTON, J., BROWNING, D., CARTER, R., DAGUM, L., FATOOHI, R., FINEBERG, S., FREDERICKSON, P., LASINSKI, T., SCHREIBER, R., SIMON, H., VENKATAKRISHNAN, V., AND WEERATUNGA, S. 1994. The NAS Parallel Benchmarks. Technical Report RNR-94-007 (March), Numerical Aerodynamic Simulation Facility, NASA Ames Research Center, Moffett Field, CA.

BERTHOMÉ, P., FERREIRA, A., MAGGS, B. M., PERENNES, S., AND PLAXTON, C. G. 1993. Sorting-Based Selection Algorithms for Hypercubic Networks. In *Proceedings of the 7th International Parallel Processing Symposium* (Newport Beach, CA, April 1993), pp. 89–95. IEEE Computer Society Press.

BLUM, M., FLOYD, R. W., PRATT, V. R., RIVEST, R. L., AND TARJAN, R. E. 1973. Time Bounds for Selection. *Journal of Computer and System Sciences 7*, 4, 448–461.

CHERNOFF, H. 1952. A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the Sum of Observations. *Annals of Math. Stat. 23*, 493–509.

HAO, E., MACKENZIE, P. D., AND STOUT, Q. F. 1992. Selection on the Reconfigurable Mesh. In *Proceedings of the 4th Symposium on the Frontiers of Massively Parallel Computation* (McLean, VA, October 1992), pp. 38–45. IEEE Computer Society Press.

HOROWITZ, E. AND SAHNI, S. 1978. *Fundamentals of Computer Algorithms*. Computer Science Press, Inc., Potomac, MD.

KRIZANC, D. AND NARAYANAN, L. 1992. Optimal Algorithms for Selection on a Mesh-Connected Processor Array. In *Proceedings of the Fourth IEEE Symposium on Parallel and Distributed Processing* (Arlington, TX, December 1992), pp. 70–76.

RAJASEKARAN, S. 1996. Randomized Selection on the Hypercube. *Journal of Parallel and Distributed Computing 37*, 2, 187–193.

RAJASEKARAN, S., CHEN, W., AND YOOSEPH, S. 1994. Unifying Themes for Network Selection. In *Proceedings of the 5th International Symposium on Algorithms and Computation (ISAAC'94)* (Beijing, China, August 1994), pp. 92–100. Springer-Verlag.

RAJASEKARAN, S. AND REIF, J. H. 1993. Derivation of Randomized Sorting and Selection Algorithms. In R. PAIGE, J. REIF, AND R. WACHTER Eds., *Parallel Algorithms Derivation and Program Transformation*, Chapter 6, pp. 187–205. Boston, MA: Kluwer Academic Publishers.

RAJASEKARAN, S. AND SAHNI, S. 1997. Sorting, Selection and Routing on the Array with Reconfigurable Optical Buses. *IEEE Transactions on Parallel and Distributed Systems 8*, 11, 1123–1132.

RAJASEKARAN, S. AND SAHNI, S. 1998. Randomized Routing, Selection, and Sorting on the OTIS-Mesh. *IEEE Transactions on Parallel and Distributed Systems 9*, 9, 833–840.

RAJASEKARAN, S. AND WEI, D. S. 1997. Selection, Routing, and Sorting on the Star Graph. *Journal of Parallel and Distributed Computing 41*, 225–233.

SARNATH, R. AND HE, X. 1997. On Parallel Selection and Searching in Partial Orders: Sorted Matrices. *Journal of Parallel and Distributed Computing 40*, 242–247.