

An Improved Randomized Selection Algorithm With an Experimental Study

David A. Bader*
dbader@ece.unm.edu

Department of Electrical and Computer Engineering
The University of New Mexico, Albuquerque, NM 87131

September 30, 1999

Abstract

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 authors.

Keywords: Selection Algorithm, Randomized Algorithms, Parallel Algorithms, Experimental Parallel Algorithmics.

*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.

1 Introduction

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 [4, 5, 3]. In this paper, we discuss a new UltraFast Randomized algorithm for the selection problem which, unlike previous research (for example, [10, 11, 14, 7, 13, 12, 15, 18, 17, 16]), 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-Furaih et al. [2] implement both our deterministic algorithm and the randomized algorithms due to Rajasekaran et al. [14, 12] 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 known 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 Appendix B.

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 [14, 12] 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 $n = O(p^2)$, 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 and detailed in the tables in Appendix B, 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 more computationally hard 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.

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 $\approx \max(p^2, 4096)$
- { ϵ } - \log_n of the sample size (e.g. 0.6)
- { Δ^* } - selection coefficient (e.g. 1.0)
- { κ } - selection coefficient multiplier (e.g. 2.25)
- $rank$ - desired rank among the elements

begin

Step 0. Set $n_i = \frac{n}{p}$.

While ($n > C$)

Step 1. Collect a sample S_i from L_i by picking $n_i \frac{n^\epsilon}{n}$ elements at random on P_i .

Step 2. $S = \mathbf{Gather}(S_i, p)$.

Set $z = \text{TRUE}$ and $\Delta = \Delta^*$.

While ($z \equiv \text{TRUE}$)

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. $c_{less} = \mathbf{Combine}(less, +)$; $c_{mid} = \mathbf{Combine}(middle, +)$;

Step 7. If ($rank \in (c_{less}, c_{less} + c_{mid}]$)

$n = c_{mid}$; $n_i = middle$; $rank = rank - c_{less}$; $z = \text{FALSE}$

Else

On P_0 : $\Delta = \kappa \cdot \Delta$

Endif

Endwhile

Endwhile

Step 8. $L = \mathbf{Gather}(L_i)$.

Step 9. On P_0

Perform sequential selection to find element q of $rank$ in L ;

$result = \mathbf{Broadcast}(q)$.

end

2.2 Fast Randomized Selection Algorithm

This algorithm is due to Rajasekaran and Reif [14, 12], and implemented by Al-furaih et al. [2].

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 $|L_i| = \frac{n}{p}$
- { ϵ } - \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 $n_i \frac{n^\epsilon}{n}$ elements at random on P_i between l and r .

Step 2. $S = \mathbf{ParallelSort}(S_i, p)$.

On P_0

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

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. $c_{mid} = \mathbf{Combine}(middle, +)$.

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

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. $L = \mathbf{Gather}(L_i[l, r])$.

Step 10. On P_0

Perform sequential selection to find element q of $rank$ in L ,
 $result = \mathbf{Broadcast}(q)$.

end

3 Analysis

The following sampling lemma from Rajasekaran [14] will be used in the analysis.

Let $S = \{v_1, v_2, \dots, v_s\}$ be a random sample from a set X of cardinality n . Also, let v'_1, v'_2, \dots, 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 , $Pr\left(|r_i - i\frac{n}{s}| > \sqrt{3\alpha}\frac{n}{\sqrt{s}}\sqrt{\log_e n}\right) < n^{-\alpha}$.*

Thus, if k_1 and k_2 are chosen as the splitters from sample set S by selecting the elements with rank $\frac{is}{n} - d\sqrt{s\log_e n}$ and $\frac{is}{n} + d\sqrt{s\log_e n}$, 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 $r \leq 8\frac{n}{\sqrt{s}}\sqrt{\log_e n}$. Since $s \approx n^\epsilon$, this can be approximated by $r \leq 8n^{1-\frac{\epsilon}{2}}\sqrt{\log_e n}$.

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 $\Delta = 2\sqrt{-\log_e(1-\rho)}$ (see Table I). Then the size of the middle partition can be bounded similarly by $r \leq 16\frac{n}{\sqrt{s}}\sqrt{-\log_e(1-\rho)}$. This can be approximated by $r \leq 16n^{1-\frac{\epsilon}{2}}\sqrt{-\log_e(1-\rho)}$. 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 [2].

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 $O(n^{(j)} + (\tau + \sigma) \log p)$ 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 ,

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

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

Each iteration of the UltraFast algorithm is similar to Fast, except sorting is replaced by sequential selection, which takes linear time [8]. Also, the problem size during iteration i is bounded with the following recurrence,

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

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 three inputs. Given a problem of size n and a p processors,

- **[I]** - Identical elements $\{0, 1, \dots, \frac{n}{p} - 1\}$ on each processor,
- **[S]** - Sorted elements $\{0, 1, \dots, n - 1\}$ distributed in p blocks across the processors, and
- **[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 [6]. 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:

$$x_{k+1} = ax_k \pmod{2^{46}}$$

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.

3.3 Empirical Results

Results for a previous implementation of the Fast randomized selection algorithm on the TMC CM-5 parallel machine appear in [2]. 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	SP-2	CM-5	SP-2
512K	4	174	68.0	194	104
	8	105	62.7	119	79.6
	16	69.5	39.5	86.7	61.9
2M	4	591	153	601	229
	8	318	108	359	182
	16	193	74.4	237	136

Table II: Comparison of the execution time of the Fast Randomized Selection Algorithm on TMC CM-5 [1, 2] and IBM SP-2-TN (in milliseconds).

Table II compares the execution time of the Fast Randomized algorithm on both the CM-5 [1, 2] and the 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 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 communication latency and bandwidth are improved roughly by a factor of three.

We conducted experiments with our UltraFast and the known Fast randomized selection algorithms on an IBM SP-2 with four, eight, and sixteen processors, by finding the median of each input in the previous section for various problem sizes (ranging between $16K$ to $16M$ elements)¹. 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, 2, and 3, respectively. 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.

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 cases when it iterates hundreds or even thousands of times. However, the UltraFast algorithm never iterates more than three times. This is due to two reasons. First, as evidenced in the following tables, 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 $n = O(p^2)$, 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 iterate from tens to thousands of times before pruning any additional elements from the solution space.

Tables of statistics for various inputs and using $p = 8$ processors are provided for both algorithms in Appendix B which show the statistics of each iteration for each input, including the sample size (s), partitioning information, and a term, k^* , defined as half of the difference between the ranks of the two splitters selection from the sample. All results from the UltraFast algorithm appear in Table IV. Results from the Fast algorithm (for the **[I]**, **[S]**, and **[R]** inputs) for $n = 512K, 1M, 2M, 4M,$ and $8M$, appear in Tables V, VI, VII, VIII, and IX, respectively. In addition, the statistics from the **[N]** input using the Fast algorithm are provided in Tables X and XI. Note that to condense the size of these tables, when no progress occurred during successive iterations of the Fast algorithm (and thus, similar lines would appear successively in the table), these identical multiple lines are replaced by a single informative line.

¹Throughout 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
1M	I	24	2
	S	17	2
	R	22	2
	N	32	2
2M	I	26	2
	S	22	3
	R	21	2
	N	38	3
4M	I	37	3
	S	23	3
	R	21	3
	N	4095	3
8M	I	28	3
	S	24	3
	R	21	3
	N	866	3

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

Execution Time of Fast and UltraFast Randomized Selection Algorithms on a 4-node IBM SP-2-TN

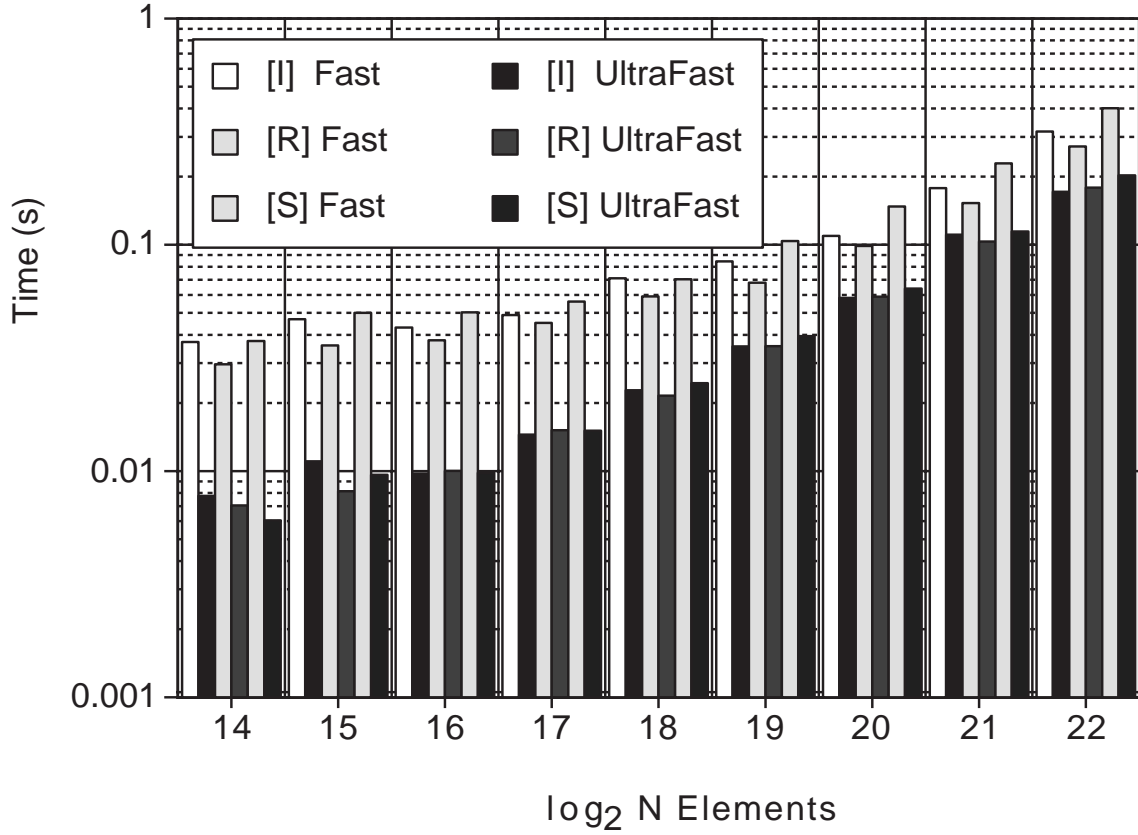


Figure 1: Empirical Performance of Fast versus UltraFast Randomized Selection Algorithms with $p = 4$ nodes of an IBM SP-2-TN.

Execution Time of Fast and UltraFast Randomized Selection Algorithms on an 8-node IBM SP-2-TN

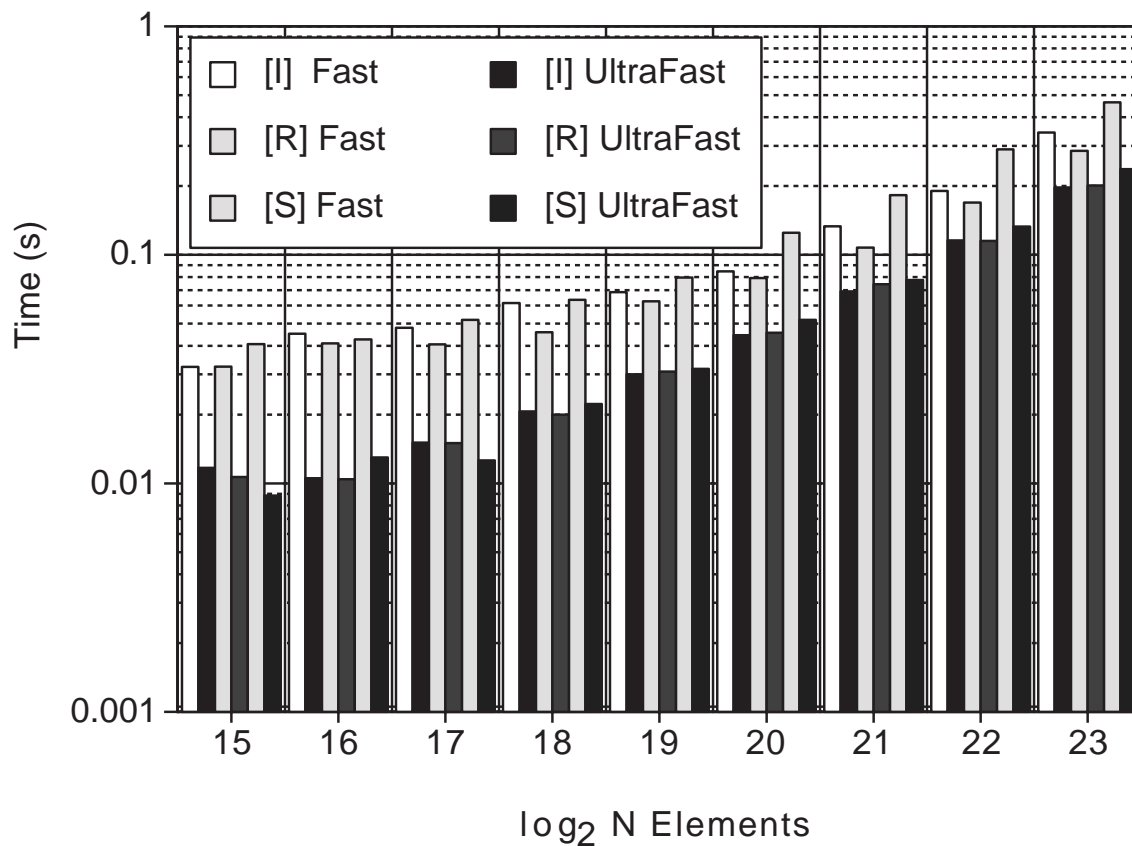


Figure 2: Empirical Performance of Fast versus UltraFast Randomized Selection Algorithms with $p = 8$ nodes of an IBM SP-2-TN.

Execution Time of Fast and UltraFast Randomized Selection Algorithms on a 16-node IBM SP-2-TN

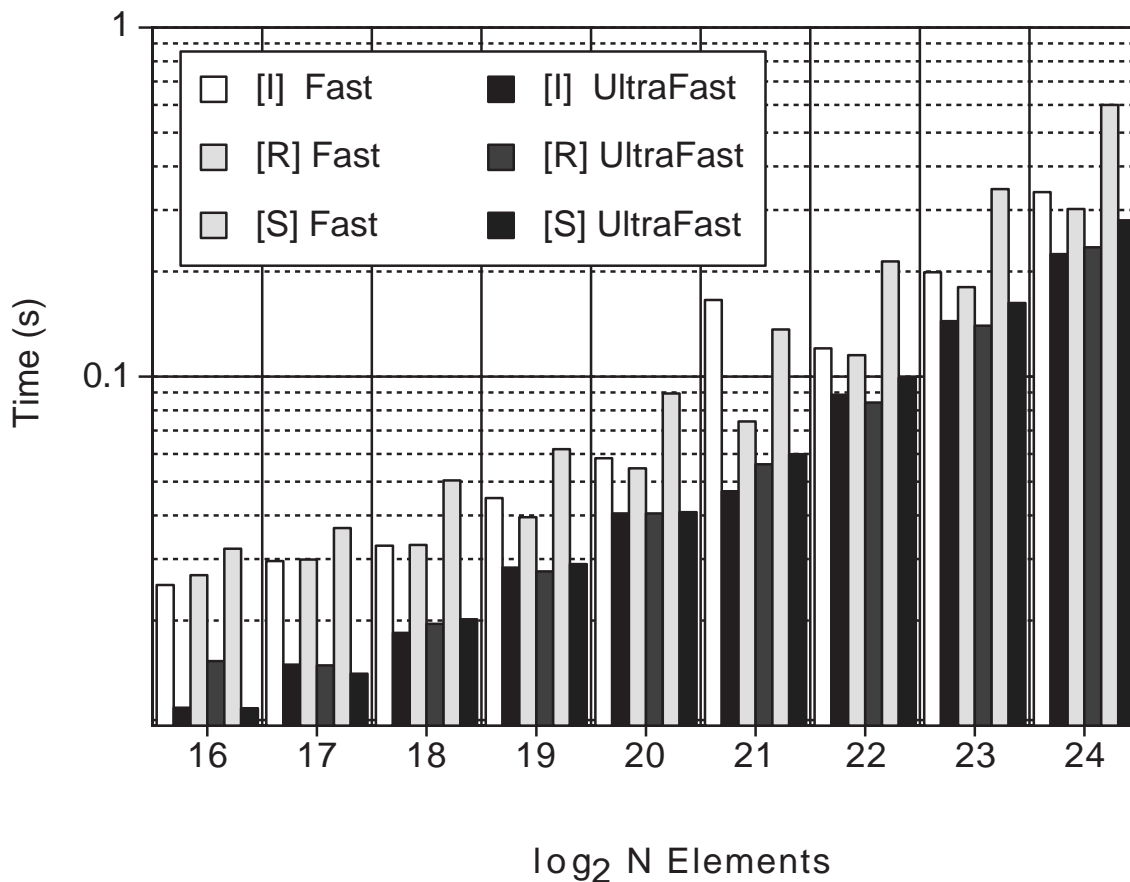


Figure 3: Empirical Performance of Fast versus UltraFast Randomized Selection Algorithms with $p = 16$ nodes of an IBM SP-2-TN.

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). 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, the second step, a **Gather** communication, is eliminated. After a single processor determines the splitters k_1 and k_2 from the sample, the **Broadcast** communication in step four simplifies into a memory read by each processor. The **Combine** in step six may be replaced by the corresponding shared-memory primitive. The **Gather** in step eight can be replaced with a shared-memory gather. We are currently investigating the performance of this SMP approach.

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 [9], are as follows.

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

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

for all $0 < \epsilon < 1$.

B Additional Statistics from Selection Experiments

n	Input	k^*	s	mid	
512K	I	52	2704	24320	
		20	432	2648	
	S	52	2704	16095	
		18	335	1590	
	R	52	2704	18457	
		19	365	1636	
	N	52	2704	21054	
		19	396	2325	
	1M	I	64	4096	35176
			23	536	3352
S		64	4096	36978	
		23	552	3473	
R		64	4096	29897	
		22	489	2655	
N		64	4096	35230	
		23	538	1936	
2M		I	78	6216	46120
			25	632	3864
	S	78	6216	51177	
		25	670	4429	
	R	12	155	848	
		78	6216	47711	
	N	25	645	3167	
		78	6216	57714	
	4M	I	26	725	4385
			12	158	706
S		97	9416	87016	
		30	920	5600	
R		13	184	744	
		97	9416	87354	
N		30	923	5517	
		13	176	1044	
8M		I	97	9416	86187
			30	918	4702
	S	12	164	926	
		97	9416	89703	
	R	30	940	5489	
		13	180	968	
	N	119	14264		
		268	14264	309272	
	S	44	1976	14936	
		17	320	1584	
R	119	14264	139838		
	34	1224	7592		
N	14	214	1139		
	119	14264	126940		
S	34	1158	8484		
	15	231	1360		
R	119	14264	139783		
	35	1227	7622		
N	14	218	811		

Table IV: Characteristics of the UltraFast Randomized Selection Algorithm. For each input size (n) and distribution, a row in the table corresponds to a selection of splitters. In this case, only in one case, $n = 8M$ and [I], multiple rows for the same input correspond to widening the center partition. k^* is defined as half of the difference between the ranks of the two splitters selected from the sample of size s . For this table, the number of processors used $p = 8$.

n	Input	k^*	s	less	mid	high	sel-part	
512K	I	189	2704	233424	69912	220952	MID	
		95	808	20176	16688	33048	MID	
		58	344	5992	4944	5752	MID	
		38	168	1360	1936	1648	MID	
		27	96	776	928	232	MID	
		21	64	96	696	136	MID	
		20	56	96	456	144	MID	
		16	40	56	384	16	MID	
		16	40	16	296	72	MID	
		14	32	32	264	0	MID	
		14	32	24	208	32	MID	
		14	32	8	136	64	MID	
		11	24	32	104	0	MID	
		11	24	8	96	0	MID	
		9	16	8	80	8	MID	
		9	16	0	80	0	MID	
		9	16	0	80	0	MID	
		9	16	0	72	8	MID	
		9	16	8	64	0	MID	
		S	189	2704	227089	71404	225795	MID
			96	818	26271	17924	27209	MID
			60	357	6007	5921	5996	MID
			41	185	1609	2876	1436	MID
			31	120	380	1537	959	MID
	25		82	298	1022	217	MID	
	22		65	168	645	209	MID	
	18		50	125	481	39	MID	
	17		42	20	379	82	MID	
	15		36	52	302	25	MID	
	14		32	48	220	34	MID	
	12		26	19	179	22	MID	
	12		24	10	151	18	MID	
	11		21	14	123	14	MID	
	10		18	11	103	9	MID	
	9		17	3	66	34	MID	
	8		13	0	61	5	MID	
	R		189	2704	226841	70460	226987	MID
			96	814	25981	17219	27260	MID
			59	351	7229	5635	4355	MID
			40	183	777	2572	2286	MID
			31	115	483	1568	521	MID
			26	87	509	784	275	MID
			20	59	79	593	112	MID
		18	50	35	446	112	MID	
		17	44	46	366	34	MID	
		15	38	32	306	28	MID	
		15	35	27	234	45	MID	
		13	30	3	202	29	MID	
		13	29	27	174	1	MID	
		12	26	2	148	24	MID	
		11	23	15	132	1	MID	
		11	21	2	130	0	MID	
		10	20	3	125	2	MID	
		11	21	1	124	0	MID	
		11	21	3	120	1	MID	
		10	20	2	117	1	MID	
		10	20	0	115	2	MID	
		10	20	1	114	0	MID	
		10	20	9	105	0	MID	
		10	20	1	80	24	MID	
		9	18	1	78	1	MID	
		9	17	0	77	1	MID	
		9	17	2	75	0	MID	
		9	17	2	71	2	MID	
		8	15	9	61	1	MID	

Table V: Characteristics of the Fast Randomized Selection Algorithm for $n = 512K$ elements and $p = 8$ processors. For each distribution, a row in the table corresponds to an iteration of the algorithm. k^* is defined as half of the difference between the ranks of the two splitters selected from the sample of size s . The sizes of the *less*, *middle*, and *high* partitions are in the similarly named columns. *sel-part* is the selection partition for the following iteration.

n	Input	k^*	s	less	mid	high	sel-part	
1M	I	239	4096	467152	109080	472344	MID	
		111	1056	42672	22192	44216	MID	
		64	408	11256	6624	4312	MID	
		42	200	2112	2624	1888	MID	
		31	120	368	1440	816	MID	
		25	80	288	848	304	MID	
		21	64	160	592	96	MID	
		18	48	80	480	32	MID	
		18	48	64	312	104	MID	
		14	32	32	248	32	MID	
		14	32	8	192	48	MID	
		12	24	24	160	8	MID	
		12	24	0	152	8	MID	
		11	24	8	136	8	MID	
		11	24	0	128	8	MID	
		11	24	8	112	8	MID	
		11	24	8	104	0	MID	
		11	24	8	96	0	MID	
		9	16	0	96	0	MID	
		9	16	16	80	0	MID	
		9	16	0	72	8	MID	
		9	16	0	72	0	MID	
		9	16	0	72	0	MID	
		9	16	8	64	0	MID	
		S	239	4096	468313	115293	464970	MID
			113	1090	45942	21792	47559	MID
			64	402	7661	6394	7737	MID
			42	193	1208	2514	2672	MID
			30	110	500	1195	819	MID
			23	72	296	714	185	MID
			19	52	162	459	93	MID
			16	40	68	297	94	MID
			14	32	17	256	24	MID
			13	29	69	161	26	MID
			11	22	8	126	27	MID
	10		19	6	101	19	MID	
	9		16	1	97	3	MID	
	9		16	10	87	0	MID	
	9		16	1	76	10	MID	
	8		14	0	67	9	MID	
	8		13	3	57	7	MID	
	R		239	4096	450794	131755	466027	MID
			119	1184	61089	26645	44021	MID
			69	456	8444	7285	10916	MID
			44	212	2881	2756	1648	MID
			31	120	471	1357	928	MID
			25	80	178	860	319	MID
			21	62	163	585	112	MID
			18	49	42	453	90	MID
			17	43	56	326	71	MID
			15	37	41	272	13	MID
			14	33	55	210	7	MID
			13	30	6	176	28	MID
			12	27	4	138	34	MID
			11	22	10	124	4	MID
			11	22	7	115	2	MID
			10	21	1	97	17	MID
			10	20	8	84	5	MID
			10	19	6	78	0	MID
		9	17	4	72	2	MID	
		9	17	2	70	0	MID	
		9	17	2	68	0	MID	
		9	17	1	64	3	MID	

Table VI: Characteristics of the Fast Randomized Selection Algorithm for $n = 1M$ elements and $p = 8$ processors. For each distribution, a row in the table corresponds to an iteration of the algorithm. k^* is defined as half of the difference between the ranks of the two splitters selected from the sample of size s . The sizes of the *less*, *middle*, and *high* partitions are in the similarly named columns. *sel-part* is the selection partition for the following iteration.

n	Input	k^*	s	less	mid	high	sel-part	
2M	I	301	6216	953968	213840	929344	MID	
		140	1584	76912	36176	100752	MID	
		76	544	12392	10720	13064	MID	
		50	264	3792	3736	3192	MID	
		35	144	736	1792	1208	MID	
		27	96	312	1064	416	MID	
		23	72	184	616	264	MID	
		18	48	112	432	72	MID	
		16	40	16	328	88	MID	
		16	40	16	288	24	MID	
		14	32	24	216	48	MID	
		14	32	24	192	0	MID	
		12	24	0	192	0	MID	
		12	24	8	184	0	MID	
		12	24	0	160	24	MID	
		12	24	8	152	0	MID	
		11	24	8	128	16	MID	
		11	24	8	120	0	MID	
		11	24	0	120	0	MID	
		11	24	8	112	0	MID	
		11	24	8	96	8	MID	
		9	16	0	96	0	MID	
		9	16	0	96	0	MID	
		9	16	0	72	24	MID	
		9	16	0	72	0	MID	
		9	16	16	56	0	MID	
		S	301	6216	942777	202452	951923	MID
			137	1527	87938	35804	78710	MID
			76	541	12438	10905	12461	MID
			50	266	3806	3792	3307	MID
			35	141	540	1833	1419	MID
			27	92	591	1086	156	MID
			22	67	114	708	264	MID
			19	52	108	507	93	MID
			17	43	122	330	55	MID
	14		33	37	279	14	MID	
	14		31	18	194	67	MID	
	12		25	5	178	11	MID	
	12		24	10	166	2	MID	
	11		23	15	146	5	MID	
	11		21	9	122	15	MID	
	10		19	13	108	1	MID	
	10		18	0	89	19	MID	
	9		15	0	88	1	MID	
	9		15	1	86	1	MID	
	9		15	14	71	1	MID	
	8	14	4	66	1	MID		
	8	13	5	48	13	MID		
	R	301	6216	951691	195168	950293	MID	
		136	1498	82144	33571	79453	MID	
		74	524	10667	9171	13733	MID	
		47	242	2453	3307	3411	MID	
		33	133	965	1657	685	MID	
		26	89	227	845	585	MID	
		21	62	202	580	63	MID	
		18	49	63	387	130	MID	
		16	40	10	289	88	MID	
		14	34	16	266	7	MID	
		14	32	26	227	13	MID	
		13	30	46	163	18	MID	
		12	25	6	126	31	MID	
		11	21	7	119	0	MID	
		11	21	3	114	2	MID	
		10	21	6	108	0	MID	
		10	21	1	90	17	MID	
		9	18	0	89	1	MID	
		9	18	3	84	2	MID	
		10	20	12	70	2	MID	
	9	17	7	58	5	MID		

Table VII: Characteristics of the Fast Randomized Selection Algorithm for $n = 2M$ elements and $p = 8$ processors. For each distribution, a row in the table corresponds to an iteration of the algorithm. k^* is defined as half of the difference between the ranks of the two splitters selected from the sample of size s . The sizes of the *less*, *middle*, and *high* partitions are in the similarly named columns. *sel-part* is the selection partition for the following iteration.

n	Input	k^*	s	less	mid	high	sel-part		
4M	I	379	9416	1951136	317088	1926080	MID		
		160	2000	124968	46064	146056	MID		
		83	632	14712	11152	20200	MID		
		51	272	3848	4488	2816	MID		
		37	160	1480	2192	816	MID		
		29	104	400	1264	528	MID		
		24	80	232	784	248	MID		
		20	56	128	528	128	MID		
		18	48	96	384	48	MID		
		16	40	0	312	72	MID		
		14	32	24	280	8	MID		
		14	32	32	232	16	MID		
		14	32	0	200	32	MID		
		14	32	16	152	32	MID		
		11	24	8	144	0	MID		
		11	24	16	128	0	MID		
		11	24	0	120	8	MID		
		11	24	0	120	0	MID		
		11	24	0	120	0	MID		
		11	24	8	112	0	MID		
		11	24	0	112	0	MID		
		11	24	0	96	16	MID		
		9	16	0	96	0	MID		
		9	16	0	96	0	MID		
		9	16	0	96	0	MID		
		9	16	8	80	8	MID		
		9	16	0	80	0	MID		
		REPEAT LAST LINE 7 TIMES							
		9	16	0	72	8	MID		
		9	16	0	72	0	MID		
		9	16	24	48	0	MID		
		S	379	9416	1920715	336973	1936616	MID	
			163	2074	152436	48458	136079	MID	
			84	648	17526	14143	16789	MID	
			55	310	4053	5644	4446	MID	
			40	179	1167	2590	1887	MID	
	30		113	544	1458	588	MID		
	25		80	150	1079	229	MID		
	22		67	253	719	107	MID		
	19		53	70	502	147	MID		
	17		42	88	377	37	MID		
	15		36	3	332	42	MID		
	15		34	16	279	37	MID		
	13		30	30	215	34	MID		
	12		26	28	180	7	MID		
	12		24	3	150	27	MID		
	11		21	7	139	4	MID		
	10		20	6	132	1	MID		
	10		20	3	118	11	MID		
	10		19	14	103	1	MID		
	9		17	0	92	11	MID		
	9		16	15	73	4	MID		
	8		14	1	70	2	MID		
	8		14	7	62	1	MID		
	R		379	9416	1889677	346471	1958156	MID	
		165	2111	180378	49487	116606	MID		
		85	660	20098	12069	17320	MID		
		52	286	5539	3924	2606	MID		
		35	148	411	1781	1732	MID		
		27	93	340	1088	353	MID		
		23	70	284	767	37	MID		
		20	59	184	557	26	MID		
		18	48	66	386	105	MID		
		16	39	37	274	75	MID		
		14	33	63	203	8	MID		
		13	29	8	150	45	MID		
		11	24	9	131	10	MID		
		11	23	7	119	5	MID		
		11	23	11	96	12	MID		
		10	18	3	88	5	MID		
		10	19	0	79	9	MID		
		9	18	1	77	1	MID		
		9	18	2	75	0	MID		
		9	18	2	72	1	MID		
		9	18	12	59	1	MID		

Table VIII: Characteristics of the Fast Randomized Selection Algorithm for $n = 4M$ elements and $p = 8$ processors. For each distribution, a row in the table corresponds to an iteration of the algorithm. k^* is defined as half of the difference between the ranks of the two splitters selected from the sample of size s . The sizes of the *less*, *middle*, and *high* partitions are in the similarly named columns. *sel-part* is the selection partition for the following iteration.

n	Input	k^*	s	less	mid	high	sel-part	
8M	I	477	14264	3855392	566568	3966648	MID	
		194	2832	306544	74296	185728	MID	
		98	840	23856	17680	32760	MID	
		60	360	5592	5416	6672	MID	
		39	176	1256	2976	1184	MID	
		32	128	728	1576	672	MID	
		26	88	592	832	152	MID	
		21	64	80	528	224	MID	
		18	48	48	448	32	MID	
		16	40	48	376	24	MID	
		16	40	8	304	64	MID	
		14	32	56	248	0	MID	
		14	32	8	208	32	MID	
		14	32	32	152	24	MID	
		11	24	0	152	0	MID	
		11	24	0	136	16	MID	
		11	24	0	136	0	MID	
		11	24	0	136	0	MID	
		11	24	0	136	0	MID	
		11	24	8	120	8	MID	
		11	24	0	112	8	MID	
		11	24	8	96	8	MID	
		9	16	0	96	0	MID	
		9	16	0	96	0	MID	
		9	16	0	88	8	MID	
		9	16	8	72	8	MID	
		9	16	0	72	0	MID	
		9	16	16	56	0	MID	
		S	477	14264	3909178	566332	3913098	MID
			194	2831	241034	84680	240618	MID
			102	906	35441	19360	29879	MID
			61	374	5543	6378	7439	MID
			42	193	1949	2622	1807	MID
			30	113	591	1285	746	MID
			24	74	232	668	385	MID
			19	50	123	474	71	MID
			17	42	50	316	108	MID
			14	33	69	242	5	MID
	13		28	7	211	24	MID	
	12		26	2	185	24	MID	
	12		24	7	178	0	MID	
	11		23	9	161	8	MID	
	11		23	3	151	7	MID	
	11		21	11	132	8	MID	
	10		19	0	125	7	MID	
	10		19	15	110	0	MID	
	10		18	3	89	18	MID	
	9		16	0	89	0	MID	
	9		16	3	86	0	MID	
	9		15	2	82	2	MID	
	9		15	0	69	13	MID	
	8		13	7	60	2	MID	
	R		477	14264	3913253	549785	3925570	MID
			192	2785	242682	81672	225431	MID
			101	890	29303	19408	32961	MID
			62	377	5684	6580	7144	MID
			42	199	2149	3054	1377	MID
			32	127	815	1281	958	MID
			24	77	86	737	458	MID
			20	57	99	440	198	MID
			16	41	129	290	21	MID
			14	34	14	172	104	MID
			12	24	24	145	3	MID
			11	24	4	133	8	MID
			11	23	4	127	2	MID
			11	23	14	113	0	MID
		11	22	2	98	13	MID	
		10	20	3	94	1	MID	
		10	18	0	93	1	MID	
		9	17	5	85	3	MID	
		9	18	2	77	6	MID	
		9	16	4	68	5	MID	
		8	15	9	52	7	MID	

Table IX: Characteristics of the Fast Randomized Selection Algorithm for $n = 8M$ elements and $p = 8$ processors. For each distribution, a row in the table corresponds to an iteration of the algorithm. k^* is defined as half of the difference between the ranks of the two splitters selected from the sample of size s . The sizes of the *less*, *middle*, and *high* partitions are in the similarly named columns. *sel-part* is the selection partition for the following iteration.

n	Input	k^*	s	less	mid	high	sel-part
512K	N	189	2704	226151	74047	224090	MID
		97	839	25308	16715	32024	MID
		58	345	8103	4910	3702	MID
		38	169	1359	2137	1414	MID
		29	104	507	1231	399	MID
		24	76	298	802	131	MID
		20	59	147	527	128	MID
		18	47	117	373	37	MID
		16	39	37	293	43	MID
		15	35	14	246	33	MID
		14	32	8	197	41	MID
		13	29	7	179	11	MID
		12	27	28	151	0	MID
		11	24	0	122	29	MID
		11	22	14	108	0	MID
		10	20	13	95	0	MID
		10	19	3	72	20	MID
		9	18	0	67	5	MID
		9	18	3	64	0	MID
		1M	N	239	4096	461522	126606
117	1157			52970	24875	48761	MID
67	438			5825	8152	10898	MID
46	227			2197	2942	3013	MID
32	125			911	1715	316	MID
27	91			324	1029	362	MID
22	66			214	677	138	MID
19	54			135	484	58	MID
17	46			20	349	115	MID
16	39			42	295	12	MID
15	36			80	182	33	MID
12	26			0	139	43	MID
11	24			7	108	24	MID
10	20			0	98	10	MID
10	20			11	85	2	MID
9	18			4	81	0	MID
9	18			0	70	11	MID
9	17			0	70	0	MID
9	17			0	66	4	MID
9	16			0	66	0	MID
REPEAT LAST LINE 11 TIMES							
9	16	0	59	7	MID		
2M	N	301	6216	959464	207955	929733	MID
		138	1555	65916	38684	103355	MID
		78	569	18790	10146	9748	MID
		49	257	2210	3667	4269	MID
		35	141	1247	1928	492	MID
		28	98	477	1063	388	MID
		23	70	130	755	178	MID
		20	58	48	554	153	MID
		18	48	125	404	25	MID
		16	40	0	365	39	MID
		16	40	47	286	32	MID
		14	33	13	249	24	MID
		14	31	0	231	18	MID
		13	30	23	191	17	MID
		12	27	0	191	0	MID
		12	27	18	163	10	MID
		12	25	0	142	21	MID
		11	23	12	121	9	MID
		11	21	0	121	0	MID
		11	21	12	109	0	MID
10	21	0	109	0	MID		
10	21	8	101	0	MID		
10	21	0	90	11	MID		
9	18	0	90	0	MID		
9	18	0	90	0	MID		
9	18	8	82	0	MID		
10	19	0	82	0	MID		
10	19	0	82	0	MID		
10	19	0	72	10	MID		
9	16	0	72	0	MID		
REPEAT LAST LINE 7 TIMES							
9	16	8	64	0	MID		

Table X: Characteristics of the Fast Randomized Selection Algorithm for input distribution [N] and $p = 8$ processors. Part 1 of 2. For each distribution, a row in the table corresponds to an iteration of the algorithm. k^* is defined as half of the difference between the ranks of the two splitters selected from the sample of size s . The sizes of the *less*, *middle*, and *high* partitions are in the similarly named columns. *sel-part* is the selection partition for the following iteration.

n	Input	k^*	s	less	mid	high	sel-part		
4M	N	379	9416	1923313	323202	1947789	MID		
		161	2024	143887	52105	127210	MID		
		87	681	23511	13027	15567	MID		
		54	300	4889	3983	4155	MID		
		36	149	501	1794	1688	MID		
		27	95	654	1007	133	MID		
		22	67	67	722	218	MID		
		20	56	117	484	121	MID		
		17	46	44	378	62	MID		
		16	41	0	320	58	MID		
		14	33	50	270	0	MID		
		14	33	30	224	16	MID		
		14	32	0	208	16	MID		
		13	28	0	189	19	MID		
		12	26	0	173	16	MID		
		12	25	17	156	0	MID		
		12	24	0	156	0	MID		
		12	24	17	139	0	MID		
		11	23	0	139	0	MID		
		11	23	0	139	0	MID		
		11	23	0	121	18	MID		
		11	22	0	121	0	MID		
		REPEAT LAST LINE 407 TIMES							
		11	22	0	98	23	MID		
		10	19	0	98	0	MID		
		10	19	0	98	0	MID		
		10	19	0	98	0	MID		
		10	19	0	98	0	MID		
		10	19	26	72	0	MID		
		9	18	0	72	0	MID		
		REPEAT LAST LINE 3,658 TIMES							
		9	18	25	47	0	MID		
		8M	N	477	14264	3871502	569292	3947814	MID
				195	2843	278692	84448	206152	MID
				102	908	33990	17465	32993	MID
				60	357	7691	5715	4059	MID
				40	183	842	2649	2224	MID
				31	118	1055	1340	254	MID
				24	79	37	869	434	MID
				21	63	192	591	86	MID
19	51			125	385	81	MID		
16	39			36	310	39	MID		
15	35			40	270	0	MID		
14	33			0	270	0	MID		
14	33			0	215	55	MID		
13	28			0	215	0	MID		
REPEAT LAST LINE 19 TIMES									
13	28			44	171	0	MID		
12	26			0	171	0	MID		
REPEAT LAST LINE 5 TIMES									
12	26			0	122	49	MID		
11	23			0	122	0	MID		
REPEAT LAST LINE 802 TIMES									
11	23			37	85	0	MID		
10	19			0	85	0	MID		
REPEAT LAST LINE 19 TIMES									
10	19			0	49	36	MID		

Table XI: Characteristics of the Fast Randomized Selection Algorithm for input distribution [N] and $p = 8$ processors. Part 2 of 2. For each distribution, a row in the table corresponds to an iteration of the algorithm. k^* is defined as half of the difference between the ranks of the two splitters selected from the sample of size s . The sizes of the *less*, *middle*, and *high* partitions are in the similarly named columns. *sel-part* is the selection partition for the following iteration.

References

- [1] I.S. Al-Furaih. Timings of Selection Algorithm. Personal communication, April 1996.
- [2] I. Al-furiah, S. Aluru, S. Goil, and S. Ranka. Practical Algorithms for Selection on Coarse-Grained Parallel Computers. *IEEE Transactions on Parallel and Distributed Systems*, 8(8):813–824, 1997.
- [3] D.A. Bader. *On the Design and Analysis of Practical Parallel Algorithms for Combinatorial Problems with Applications to Image Processing*. PhD thesis, University of Maryland, College Park, Department of Electrical Engineering, April 1996.
- [4] D.A. Bader and J. JáJá. Practical Parallel Algorithms for Dynamic Data Redistribution, Median Finding, and Selection. Technical Report CS-TR-3494 and UMIACS-TR-95-74, UMIACS and Electrical Engineering, University of Maryland, College Park, MD, July 1995.
- [5] D.A. Bader and J. JáJá. Practical Parallel Algorithms for Dynamic Data Redistribution, Median Finding, and Selection. In *Proceedings of the 10th International Parallel Processing Symposium*, pages 292–301, Honolulu, HI, April 1996.
- [6] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrisnan, and S. Weeratunga. The NAS Parallel Benchmarks. Technical Report RNR-94-007, Numerical Aerodynamic Simulation Facility, NASA Ames Research Center, Moffett Field, CA, March 1994.
- [7] P. Berthomé, A. Ferreira, B.M. Maggs, S. Perennes, and C.G. Plaxton. Sorting-Based Selection Algorithms for Hypercubic Networks. In *Proceedings of the 7th International Parallel Processing Symposium*, pages 89–95, Newport Beach, CA, April 1993. IEEE Computer Society Press.
- [8] M. Blum, R.W. Floyd, V.R. Pratt, R.L. Rivest, and R.E. Tarjan. Time Bounds for Selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1973.
- [9] H. Chernoff. A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the Sum of Observations. *Annals of Math. Stat.*, 23:493–509, 1952.
- [10] E. Hao, P.D. MacKenzie, and Q.F. Stout. Selection on the Reconfigurable Mesh. In *Proceedings of the 4th Symposium on the Frontiers of Massively Parallel Computation*, pages 38–45, McLean, VA, October 1992. IEEE Computer Society Press.
- [11] D. Krizanc and L. Narayanan. Optimal Algorithms for Selection on a Mesh-Connected Processor Array. In *Proceedings of the Fourth IEEE Symposium on Parallel and Distributed Processing*, pages 70–76, Arlington, TX, December 1992.
- [12] S. Rajasekaran. Randomized Selection on the Hypercube. *Journal of Parallel and Distributed Computing*, 37(2):187–193, 1996.
- [13] S. Rajasekaran, W. Chen, and S. Yooseph. Unifying Themes for Network Selection. In *Proceedings of the 5th International Symposium on Algorithms and Computation (ISAAC'94)*, pages 92–100, Beijing, China, August 1994. Springer-Verlag.

- [14] S. Rajasekaran and J.H. Reif. Derivation of Randomized Sorting and Selection Algorithms. In R. Paige, J. Reif, and R. Wachter, editors, *Parallel Algorithms Derivation and Program Transformation*, chapter 6, pages 187–205. Kluwer Academic Publishers, Boston, MA, 1993.
- [15] S. Rajasekaran and S. Sahni. Sorting, Selection and Routing on the Array with Reconfigurable Optical Buses. *IEEE Transactions on Parallel and Distributed Systems*, 8(11):1123–1132, 1997.
- [16] S. Rajasekaran and S. Sahni. Randomized Routing, Selection, and Sorting on the OTIS-Mesh. *IEEE Transactions on Parallel and Distributed Systems*, 9(9):833–840, 1998.
- [17] S. Rajasekaran and D.S.L. Wei. Selection, Routing, and Sorting on the Star Graph. *Journal of Parallel and Distributed Computing*, 41:225–233, 1997.
- [18] R. Sarnath and X. He. On Parallel Selection and Searching in Partial Orders: Sorted Matrices. *Journal of Parallel and Distributed Computing*, 40:242–247, 1997.