Deterministic Sorting and Randomized Median Finding on the BSP model*

Alexandros V. Gerbessiotis and Constantinos J. Siniolakis

Programming Research Group,
Computing Laboratory,
Oxford University,
Parks Road,
Oxford OX1 3QD,
UK

Abstract

We present new BSP algorithms for deterministic sorting and randomized median finding. We sort n general keys by using a partitioning scheme that achieves the requirements of efficiency (one-optimality) and insensitivity against data skew (the accuracy of the splitting keys depends solely on the step distance, which can be adapted to meet the worst-case requirements of our application). Although we employ sampling in order to realize efficiency, we can give a precise worst-case estimation of the maximum imbalance which might occur. We also investigate optimal randomized BSP algorithms for the problem of finding the median of n elements that require, with high-probability, $3n/(2p) + o(n/p)$ number of comparisons, for a wide range of values of n and p. Experimental results for the two algorithms are also presented.

1 Introduction

We present new BSP algorithms for deterministic sorting and randomized median finding. We sort n general keys by using a partitioning scheme that achieves the requirements of efficiency (one-optimality) and insensitivity against data skew (load-balance). An important distinction between the approach that we consider in this paper and previous BSP approaches [8] is that our approach does not sample the unsorted input data, but it samples the sorted initial runs. The accuracy of the splitting keys depends solely on the step distance, which can be adapted to meet the worst-case requirements of our application. Therefore, although we employ sampling [8, 9, 21] in order to realize efficiency, we can give a precise worst-case estimation of the maximum imbalance which might occur. The idea of deterministic sampling is not a new one, see i.e. [21], however our techniques improve upon previous results on (i) the amount of parallel slackness required, (ii) the communication requirements, (iii) the multiplicative constant factors involved. Previously known results on BSP sorting include deterministic [1] and high-probability [8] algorithms. In [1], it is shown that for all values of n and p such that $p = n^{1-\epsilon}$, $0 < \epsilon < 1$, a version of Column-sort [14] provides an upper bound of $\lceil \epsilon^{-3.42} \rceil O(n \lg n/p)$ and $O(gn \lg n/p)$ for computation and communication respectively, where g is the ratio of the total throughput of the whole system in terms of basic computational operations, to the throughput of the router in terms of words of information delivered. The algorithm of [8] requires, with high-probability, computation and communication time $(1 + o(1))(T_{seq}/p)$ and $O(gT_{seq}/(p \lg n))$ respectively, for all values of n and p such that $p = n^{1-\epsilon}$, $0 < \epsilon < 1$, where T_{seq} is the time required by sequential sorting. Our deterministic algorithm complements the high-probability result of [8], and improves the upper bound of [1] by reducing the communication requirements by a factor of $\lg n$; the algorithm also significantly reduces the multiplicative constant factors involved in [1]. In particular, our algorithm requires time $(1 + \lfloor \epsilon^{-1}(1-\epsilon) \rfloor((1-\epsilon)/2) + o(1))(T_{seq}/p)$ and $O(gT_{seq}/(p \lg n))$ for computation and communication respectively.

The problem of computing the t-th largest element from a set of n elements with the minimum number $V_t(n)$ of comparisons has always attracted considerable attention [12]. Of particular interest have been algorithms for computing the median or performing particular selections of elements in time linear in the number of elements [4, 20]. Lower bounds on $V_t(n)$ for various values of t were presented in [4, 10, 11, 17]. The minimum average number of comparisons $\overline{V}_t(n)$ needed to find the t-th largest of n elements has also drawn considerable interest. A lower bound of $n + t - O(1)$, $1 \leq t \leq n/2$, was presented in [5], which matches the performance, within low order terms, of the algorithm by Floyd and Rivest [7]. In this context we focus our attention on randomized parallel algorithms for median finding and present a high-probability algorithm that on a p-processor BSP computer finds the median within $3n/(2p) + o(n/p)$ comparisons. Previous parallel algorithms include [18, 19]. Our algorithms provide tighter probabilistic bounds and low communication requirements.

The BSP model for parallel computation has recently been proposed [22, 23] as a unified framework for the design,

*The work of the first author was supported in part by EPSRC(UK) under grant GR/K16999, and the second author was supported by a Bodossaki Foundation Graduate Scholarship.

This extended abstract is to appear in the 8th Annual ACM Symposium on Parallel Algorithms and Architectures, Padova, Italy, June 1996.

analysis and programming of general purpose parallel computing systems. The bulk-synchronous parallel or BSP computer as described in [15, 16, 23] consists of three parts: a collection of processor/memory components; a communication network with a router that can deliver messages point-to-point among the components; and facilities for globally synchronizing, in barrier style, all or a subset of the components.

A BSP computer can be parametrized by p, the number of processor/memory components, L, the minimal time, measured in terms of basic local computation steps, between successive synchronization operations, and g the ratio of the total throughput of the whole system in terms of basic computational operations, to the throughput of the router in terms of words of information delivered.

Computation proceeds in a succession of supersteps. In each such superstep, a task is assigned to each processor that can be performed using data available locally to that processor before the start of the superstep. The task may include local computation steps, message transmissions and receipts. The synchronizer checks every L time units whether the processors that are currently synchronized have all completed their tasks and only if they have does the machine proceed to the following superstep. Alternatively, as we do here and was the case in [8], a superstep can complete any time after the first L time units.

If x is the maximum number of local computation steps executed by any processor, h_r and h_s are, respectively, the maximum number of packets that any processor sends or receives during a superstep, then each superstep is charged $\max\{L, x + hg\}$ basic time units, where we assume that $h = \max\{h_r, h_s\}$ although $h = h_r + h_s$ is also possible [22].

Along the lines of [8], in order to simplify the description of the performance of our algorithms, we assume that each superstep is either pure computation or pure communication. This way, the two kinds of costs are easily distinguished.

We assume $g \geq 1$. For any particular machine a minimum time necessary for doing synchronization imposes a lower bound on L. Also, the communication throughput specified by the chosen g may be realisable only for large enough L because of startup or latency considerations in routing.

We specify the performance of a BSP algorithm by comparing its performance with a sequential or PRAM algorithm [8]. In order to achieve this, we measure ratios between runtimes on pairs of models that have the same set of local instructions. We specify two ratios π and μ. The former, π, is the ratio between the computation count of the BSP algorithm to the computation count of the comparing sequential or PRAM algorithm. The latter, μ, is the ratio between the parallel time required by the communication supersteps of the BSP algorithm and the parallel computation time on the p processors if π were equal to one for the given sequential algorithm. Conditions on n, p, and L are specified that are sufficient for the algorithm to make sense and the claimed bounds on π and μ to hold.

In this paper we restrict ourselves to *one-optimal* and *c-optimal* algorithms in the spirit of [8]. Thus we shall say that a BSP algorithm is *c-optimal in computation* if $\pi = c + o(1)$ as $n \to \infty$. Insistence on one-optimality leads to algorithms that only work in restricted ranges of n and L. These ranges can be extended in most cases, however, if the requirements for optimality are relaxed to c-optimality.

All asymptotic bounds refer to the problem size as $n \to \infty$. We use the term "with high-probability" to denote that a claim holds with probability at least $1 - 1/n^{\delta}$ for some constant $\delta > 0$, where n is the size of the input set.

One also needs to specify how the local operations in BSP algorithms are charged. Since we are interested in ratios of performance of two algorithms we can define operations in a higher level than machine level instructions. For a median finding algorithm we charge 1 for choosing a random element, 1 for the operations over an associative operator in parallel prefix computations, and 1 for each comparison performed. As this model is the one used in [8], our claims for sorting related operations on the BSP drawn from [8] are also consistent with this model.

2 Deterministic BSP Sorting

In [8] it was recently presented a randomized one-optimal BSP algorithm for the problem of sorting n general keys. In this section we present a deterministic BSP sorting algorithm that achieves comparable performance. The algorithm is one-optimal in computation for an ample range of the parameters n, p, and L, and is also one-optimal in communication if g is appropriately bounded. The algorithm is based on an efficient partitioning scheme which splits an arbitrary number of sorted sequences among all the processors almost equally (independently of the input distribution).

The algorithm consists of the following distinct phases:

(1) *Local Sorting.* In this phase, every processor, in parallel with all the other processors, sorts the subsequence of size $\lceil n/p \rceil$ (or $\lfloor n/p \rfloor$) that resides in its local memory, by employing any optimal-time sequential algorithm [13].

(2) *Partitioning.* In this phase, processors cooperate to evenly split the sorted sequences among the processors (independently of the input distribution).

(3) *Merging.* Every processor, in parallel with all the other processors, merges a small number of subsequences (of total size at most $(1 + o(1))\lceil n/p \rceil$), by employing an optimal sequential algorithm.

The iterative version of the above algorithm performs a constant number m of iterations of phases 2 and 3. In each iteration the data set is partitioned into $k = p^{1/m}$ subsets of approximately equal size; the next iteration performs a similar process of sub-partitioning each of these k subsets into k further subsets, and so on. The size of every subset produced in the m iterations is within a factor $(1 + 1/(m\lceil \omega_n \rceil))^m \leq e^{1/\omega_n} = 1 + o(1)$ of n/p, for any slowly growing function ω_n of n such that $\omega_n \to \infty$ as $n \to \infty$, i.e. $\omega_n = \lg \lg n$ (see lemma 1). Then, by unfolding the iteration and employing multi-way merging [13] we obtain the sorted set.

The remainder of this section is devoted to the proof of the following proposition and its corresponding theorem.

Proposition 1 *Let ω_n be any function of n such that $\omega_n \to \infty$ as $n \to \infty$. For any integers p and k such that p is an integral power of k and $m = \lg p / \lg k$, there is a BSP sorting algorithm that for any integer $2 \leq t \leq p$, and using the charging policy of section 1, has,*

$$\pi \leq \frac{p}{n \lg n}\left(\max\left\{L, \left\lceil\frac{n}{p}\right\rceil \lg\left\lceil\frac{n}{p}\right\rceil\right\}+ m\max\{L, m\lceil\omega_n\rceil k\}+ \frac{m\lceil\lg p\rceil(\lceil\lg p\rceil+1)}{2}\max\{L, 2m\lceil\omega_n\rceil k\}+ m\max\left\{L, 2\left(\left\lceil\left(1+\frac{1}{m\lceil\omega_n\rceil}\right)^m\frac{n}{p}\right\rceil+k-1\right)\right\}+ 4m\lceil\log_t p\rceil\max\left\{L, \frac{t(k-1)}{\lceil\log_t p\rceil}\right\}+mL+ \frac{m+1}{2}\left\lceil\left(1+\frac{1}{m\lceil\omega_n\rceil}\right)^m\frac{n}{p}\right\rceil(\lg p+2)\right),$$

and

$$\mu \leq \frac{gp}{n\lg n}\left(m\lceil\lg p\rceil\lceil\lg p+1\rceil\max\left\{\frac{L}{2g}, (m\lceil\omega_n\rceil k)\right\}+ 6m\lceil\log_t p\rceil\max\left\{\frac{L}{g}, \frac{t(k-1)}{\lceil\log_t p\rceil}\right\}+ m\max\left\{\frac{L}{g}, k-1\right\}+ m\max\left\{\frac{L}{g}, \left\lceil\left(1+\frac{1}{m\lceil\omega_n\rceil}\right)^m\frac{n}{p}\right\rceil\right\}\right).$$

The expression for π allows for all arithmetic as well as comparison operations.

Proof: The algorithm is recursive in nature, and each recursive call is divided into a sequence of stages. Initially, the variables l, r have the values $l = 0$, $r = p-1$. In any recursive call of the algorithm, these two variables give the range of the processors that participate. We also set $k = p^{1/m}$ for some constant m to be specified later. The input is assumed to be evenly distributed among the p processors before the beginning of the execution of the algorithm. Moreover, we assume that the keys are distinct since we can always make them so by, for example, appending to them the code for the memory location in which they are stored.

Algorithm SORT (l,r)

begin
SEQUENTIAL_SORT(l, r)
RECURSIVE_SORT (l, r)
end

Procedure SEQUENTIAL_SORT (l, r)

begin
Stage 0 :
Every processor, in parallel with all the other processors, sorts the keys in its possession.
[**Analysis of Stage 0:** When this step is executed, every processor holds at most $\lceil n/p\rceil$ keys. The cost of this stage is, therefore, that of sequential sorting $\lceil n/p\rceil$ keys by some deterministic algorithm such as [13], which takes time $\max\{L, \lceil n/p\rceil \lg\lceil n/p\rceil\}$.]
end

Procedure RECURSIVE_SORT (l,r)

begin
Stage 0 :
Each processor selects locally $m\lceil\omega_n\rceil k-1$ evenly spaced splitters, $k = p^{1/m}$, that partition its input into $m\lceil\omega_n\rceil k$ evenly sized segments, where ω_n is any function of n such that $\omega_n \to \infty$ as $n \to \infty$. Moreover, each processor appends to this sorted list the largest element in its subsequence. Let $s = m\lceil\omega_n\rceil k$ be the size of the so identified list.
[**Analysis of Stage 0:** Since the keys residing in the local memory of any processor are sorted when this stage is executed, this stage requires time $\max\{L, s\}$.]

Stage 1 :
The $q = r-l+1$ sorted lists, each consisting of s splitters, are merged using one of Batcher's methods [3]. Let the sequence $S_{l,r} = \{s_{ls+1}, s_{ls+2}, ..., s_{ls+qs}\}$ be the result of the merge operation. Moreover, assume that $S_{l,r}$ is evenly distributed among the q processors, i.e. the subsequence $\{s_{is+1}, ..., s_{(i+1)s}\}$, $l \leq i \leq r$, resides in the local memory of processor p_i.
[**Analysis of Stage 1:** It can be verified that the computation and communication time required for this stage is, respectively, $\lceil\lg q\rceil(\lceil\lg q\rceil+1)/2\max\{L, 2s\}$ and $\lceil\lg q\rceil(\lceil\lg q\rceil+1)/2\max\{L, 2gs\}$.]

Stage 2 :
A broadcast operation is initiated, where splitter $s_{ls+iqs/k}$, for all $1 \leq i < k$, along with its index in the set of splitters, is sent to all other participating processors.
[**Analysis of Stage 2:** If a (splitter, index) pair can be sent as one message and the $k-1$ splitters are first sent to processor p_l and then broadcast by employing a pipelined t-ary tree with root p_l, then this stage takes time at most $\max\{L, g(k-1)\} + 2\lceil\log_t q\rceil\max\{L, gt(k-1)/\lceil\log_t q\rceil\}$.]

Stage 3 :
(i) Each processor, in a single superstep, decides the position of every element it holds with respect to the $k-1$ splitters that were received in the previous stage, by way of merging, and counts the number of its keys that fall into each of the k so identified buckets.
(ii) k independent parallel prefix computations are performed (one for each bucket), in order to determine how to split the keys of each bucket as evenly as possible among the processors, using the information collected in step (i) of this stage.
[**Analysis of Stage 3:** When this step is executed, by way of lemma 1, every processor possesses a maximum of $\lceil(1+1/(m\lceil\omega_n\rceil))^m(n/p)\rceil$ keys. Thus, step (i) takes at most $\max\{L, 2(\lceil(1+1/(m\lceil\omega_n\rceil))^m(n/p)\rceil+k-1)\}$ time. The k parallel prefix computations can be realized by employing a pipelined t-ary tree with root p_l. Therefore, the prefix computation requires at most $4\lceil\log_t q\rceil\max\{L, t(k-1)/\lceil\log_t q\rceil\}$ and $4\lceil\log_t q\rceil\max\{L, gt(k-1)/\lceil\log_t q\rceil\}$ time for computation and communication respectively.]

Stage 4 :
Every processor knows where to route every key in its possession, and performs this routing. Processors route keys in "column-major" form, that is, keys destined to some bucket fill the processors assigned to this bucket in linear order, the first processor of that order being determined by the information gathered in part (ii) of stage 3.
[**Analysis of Stage 4:** This stage takes time at most $\max\{L, g\lceil(1+1/(m\lceil\omega_n\rceil))^m(n/p)\rceil\}$.]

Stage 5 :
Each processor, in parallel with all other processors, merges the at most q sorted sequences that were received in the previous stage. Note that, since by lemma 1, each processor will receive at most $\lceil (1+1/(m\lceil\omega_n\rceil))^m (n/p)\rceil$ keys, no processor will merge more than $\min\{q, \lceil (1+1/(m\lceil\omega_n\rceil))^m (n/p)\rceil\}$ sorted sequences. In order to simplify the exposition, in the remainder of the analysis, we assume that each processor receives at most q sorted sequences. Obviously, this assumption can be removed without increasing π or μ.
[**Analysis of Stage 5:** When this step is executed, every processor holds at most q sorted sequences for a total of at most $\lceil (1+1/(m\lceil\omega_n\rceil))^m (n/p)\rceil$ keys. The cost of this stage is that of sequential q-way merging $\lceil (1+1/(m\lceil\omega_n\rceil))^m (n/p)\rceil$ keys by some algorithm such as [13], which is bounded above by $\max\{\lceil (1+1/(m\lceil\omega_n\rceil))^m (n/p)\rceil \lceil \lg q\rceil\}$. Thus, the cost of this stage over the m iterations of this procedure is bounded above as follows, for $q = p^{1-(h-1)/m}$,

$$\sum_{h=1}^{m} \max\left\{L, \left(1+\frac{1}{m\lceil\omega_n\rceil}\right)^m \left\lceil\frac{n}{p}\right\rceil \lceil\lg q\rceil\right\} \leq$$
$$mL + \frac{m+1}{2}\left(1+\frac{1}{m\lceil\omega_n\rceil}\right)^m \left\lceil\frac{n}{p}\right\rceil (\lg p + 2).$$

]

Stage 6 :
After the merging has been completed, a recursive call has been completed as well.
begin
for $i = 1$ to k do in parallel
    RECURSIVE_SORT $((i-1)\frac{q}{k}+1,\ i\frac{q}{k})$
end
[**Analysis of Stage 6:** The depth of the recursion is $m = \lg p/\lg k$.]
end

By combining all computation and communication time costs and assuming that we charge $n \lg n$ for sorting n keys sequentially we obtain the pre-mentioned result. The addition of a constant so that $cn \log n$ operations are required for sorting does not increase the values of π and μ. ■

The correctness of the algorithm is easy to see. It now remains to prove that procedure RECURSIVE_SORT partitions the input into evenly sized buckets. In order to simplify the exposition, in the remainder of the analysis, we assume that $n/(ps)$ is an integer. The main result is summarized in the following lemma.

Lemma 1 *The size of every bucket produced in the m iterations of procedure* RECURSIVE_SORT *is within a factor of at most $(1+1/(m\lceil\omega_n\rceil))^m$ of n/p.*

Proof: Refer to the first recursive call of stage 2 of procedure RECURSIVE_SORT and consider an arbitrary splitter $s_{ls+iqs/k} = s_{ips/k}$, where $1 \leq i < k$. There are at least $(nips)/(kps) = ni/k$ keys which are not larger than $s_{ips/k}$, since there are ips/k segments each of size $n/(ps)$ whose keys are not larger than $s_{ips/k}$. Likewise, there are at least $n(ps - ips/k - p + 1)/(ps) = n(1 - i/k - 1/s + 1/(ps))$ keys which are not smaller than $s_{ips/k}$, since there are $ps - ips/k - p + 1$ segments each of size $n/(ps)$ whose keys are not smaller than $s_{ips/k}$. Thus, the number of keys $n_{(i)}$ that are smaller than $s_{ips/k}$ is bounded by,

$$\frac{ni}{k} \leq n_{(i)} \leq \frac{ni}{k} + \frac{n}{s} - \frac{n}{ps}.$$

We conclude the following.

$$n_{(i+1)} - n_{(i)} \leq \left(1 + \frac{1}{m\lceil\omega_n\rceil}\right)\frac{n}{k}$$

since $s = m\lceil\omega_n\rceil k$. Hence, the size of any bucket after m iterations of procedure RECURSIVE_SORT is bounded above by

$$\left(1+\frac{1}{m\lceil\omega_n\rceil}\right)^m \frac{n}{k^m} = \left(1+\frac{1}{m\lceil\omega_n\rceil}\right)^m \frac{n}{p}$$

since, by definition, $p = k^{1/m}$. The lemma follows. ■

By substituting appropriate values for the parameters in proposition 1 a multiplicity of bounds on the runtime of the algorithm can be obtained. As noted in [8], the best time bounds are obtained by selecting m, t and ω_n with regard to the actual values of n, p and L. By making however more generic selections for m, t and ω_n and worst-case bounds for p and L, more general results in terms of n can be obtained. The following is one such bound.

Theorem 1 *For any constant ϵ, $1/2 < \epsilon < 1$, there exists a BSP algorithm for sorting n data items that for all integers n, p, and L satisfying $p = n^{1-\epsilon}$ and $L = o(n^\epsilon/\lg n)$ has $\pi = 1 + o(1)$, and if $g = o(\lg n)$ also, $\mu = o(1)$.*

Proof: Let in proposition 1 $m = 1$, $t = 2$, $k = p$ and $\omega_n = \lg\lg n$. ■

By allowing more than one iteration the algorithm achieves constant factor optimality for values of n much closer to p than these of theorem 1.

Theorem 2 *For any constant ϵ, $0 < \epsilon < 1$, there exists a BSP algorithm for sorting n data items that for all integers n, p, and L such that p is an integral power of k, $p = n^{1-\epsilon}$ and $L = o(n^\epsilon/\lg n)$ has $\pi = c + o(1)$, for some constant c that depends on ϵ, and if $g = o(\lg n)$ also, $\mu = o(1)$.*

Proof: Let in proposition 1 $m = \lfloor (1-\epsilon)/\epsilon\rfloor + 1$, $t = 2$, $k = p^{1/m}$ and $\omega_n = \lg\lg n$. Then, the above result holds for $c = \epsilon + (1-\epsilon)(\lfloor (1-\epsilon)/\epsilon\rfloor + 2)/2$. ■

We note that not only can the bounds given above be expressed in a variety of ways, but also the algorithm itself leaves room for several improvements. As stated, the algorithm sorts the splitters using some of Batcher's sorting networks. In certain circumstances its performance can be improved by any of several alternative methods. For example, the sorting of the splitters could be performed by using either our algorithm recursively or some completely different algorithm [8].

3 Random Sampling

We present in this section results related to random sampling. In the discussion and proofs to follow we use the notation from [8]. Let $X = \{x_1, \ldots, x_N\}$ be a set of input elements ordered such that $x_i < x_{i+1}$, for $1 \leq i \leq N-1$. We are assuming that the elements are distinct because by appending to each element its index we can make them so. For integers $\lambda \geq 1$ and $s \geq 1$, let $Y = \{y_1, \ldots, y_{2\lambda s-1}\}$ be a randomly chosen subset of $2\lambda s - 1 \leq N/2$ elements of X, also ordered so that $y_i < y_{i+1}$. We then form the following three subsets subsets X_0, X_1, X_2, for some $\mu, t \geq 1$ such that $\lambda s > \mu t/2$ where

$$\begin{aligned} X_0 &= \{x : x < y_{\lambda s - \mu t/2}\}, \\ X_1 &= \{x : y_{\lambda s - \mu t/2} < x < y_{\lambda s + \mu t/2}\}, \\ X_2 &= \{x : y_{\lambda s + \mu t/2} < x\}. \end{aligned}$$

For $1 \leq i \leq 2\lambda s - 1$ and $1 \leq j \leq N$, let $q_i(j)$ be the probability that $y_i = x_j$. It is then [8],

$$q_i(j) = \frac{\binom{j-1}{i-1}\binom{N-j}{2\lambda s-1-i}}{\binom{N}{2\lambda s-1}}.$$

For the binomial coefficients, we assume that $\binom{i}{j} = 0$ if $j > i$. For $0 \leq i \leq 2$, let $n_i = |X_i|$ and let $p_i(j)$ be the probability that $n_i = j$. The following follows from [8].

Claim 1 *For $i = 0, 2$*

$$p_i(j) = p(j) = \frac{\binom{j}{\lambda s - \mu t/2 - 1}\binom{N-j-1}{\lambda s + \mu t/2 - 1}}{\binom{N}{2\lambda s-1}},$$

and for $i = 1$

$$p_i(j) = \frac{\binom{j}{\mu t - 1}\binom{N-j-1}{2\lambda s - \mu t - 1}}{\binom{N}{2\lambda s-1}}.$$

We would like to find lower bounds on the size of sets X_0 and X_2 by proving that the probability of having such a set of size less than $N/2 - f(N)$, for some function $f(N)$ of N, is very small. We investigate how small $f(N)$ can be for this to hold. Let us call $B = N/2 - f(N) - 1$. Particularly, we would like to show that the term $\sum_{j \leq B} p(j)$ is sufficiently small for appropriate choices of $f(N)$. We deduce the following relationship used later for $n \geq N$.

Proposition 2 *Let $\lambda, \mu \geq 1$, $s, t \geq 1$, $2\lambda s - 1 < N/2$, $n \geq N$, $\lambda s > \mu t/2$, and $\rho > 0$. If $f(N) = (N\ \mu t)/(2\ \lambda s)$ and $\mu t = 2\sqrt{3\rho\ \lambda s\ \log n}$, the probability that any one of the X_0, X_2 is of size less than $N/2 - f(N)$ is at most $\sqrt{2\pi(2\lambda s - 1)}\ e^{1/(6(2\lambda s-1))}\ /n^{\rho}$.*

Proof: The proof utilizes a similar line of arguments as [8]. We first observe that the combinatorial identity [12],

$$\sum_{n-u \leq v \leq r-m} \binom{u+v}{n}\binom{r-v}{m} = \binom{r+u+1}{m+n+1},$$

implies that

$$\sum_{j=\lambda s-\mu t/2-1}^{N-\lambda s-\mu t/} \binom{j}{\lambda s - \mu t/2 - 1}\binom{N-j-1}{\lambda s + \mu t/2 - 1} = \binom{N}{2\lambda s - 1}.$$

We use a combinatorial interpretation argument, as in [8], to argue that the sum $\sum_{j \leq B} p(j)$ for $j \leq B$ is the number of ways we can choose $2\lambda s - 1$ objects so that among them, the object with the $(\lambda s - \mu t/2)$-th highest label is labeled at most B.

In order to establish an upper bound for $\sum_{j \leq B} p(j)$ we approximate $p(j)$ by a binomial term and use bounds on the right tail of the binomial distribution. Let $q = (2\lambda s - 1)/N$. The probability that in N Bernoulli trials, each with a probability q of success we get $\lambda s - \mu t/2 - 1$ successes among the first j, a success at the $(j+1)$-st trial, and $\lambda s + \mu t/2 - 1$ successes in the remainder is

$$\begin{aligned} r(j) &= \binom{j}{\lambda s - \mu t/2 - 1} q^{\lambda s - \frac{\mu t}{2} - 1}(1-q)^{j - \lambda s + \frac{\mu t}{2} + 1} \\ &\quad q\binom{N-j-1}{\lambda s + \mu t/2 - 1} q^{\lambda s + \frac{\mu t}{2} - 1}(1-q)^{N - j - \lambda s - \frac{\mu t}{2}}. \end{aligned}$$

Let A be the ratio of $r(j)$ over $p(j)$. Then

$$A = \frac{r(j)}{p(j)} = \binom{N}{2\lambda s - 1} q^{2\lambda s - 1}(1-q)^{N - 2\lambda s + 1}.$$

The following result ([6], page 4) for $a = c/d$, $b = 1 - a$, $c < d/2$, gives that

$$e^{-\frac{1}{6c}}\ \frac{1}{\sqrt{2\pi abd}}\ \left(a^a b^b\right)^{-d} \leq \binom{d}{c}.$$

If we substitute for $d = N$, $c = 2\lambda s - 1$, $a = q = (2\lambda s - 1)/N$, and $b = 1 - a$ we get the following inequality.

$$\frac{e^{-\frac{1}{6(2\lambda s-1)}}\ q^{-2\lambda s+1}\ (1-q)^{-N+2\lambda s-1}}{\sqrt{2\pi(2\lambda s-1)(1-(2\lambda s-1)/N)}} \leq \binom{N}{2\lambda s-1}.$$

It is then immediate that the following holds.

$$A \geq \frac{e^{-\frac{1}{6(2\lambda s-1)}}}{\sqrt{2\pi(2\lambda s-1)(1-(2\lambda s-1)/N)}}. \qquad (1)$$

The following inequality follows.

$$\begin{aligned} \sum_{j \leq B} p(j) &\leq \frac{q}{A}\sum_{j \leq B}\binom{j}{\lambda s - \mu t/2 - 1} q^{\lambda s - \mu t/2 - 1} \\ &\quad (1-q)^{j - \lambda s + \mu t/2 + 1}. \end{aligned} \qquad (2)$$

An upper bound for the term below is to be established.

$$\sum_{j \leq B}\binom{j}{\lambda s - \mu t/2 - 1} q^{\lambda s - \mu t/2 - 1}(1-q)^{j - \lambda s + \mu t/2 + 1}.$$

We employ generating functions. The term $\binom{j}{\lambda s-\mu t/2-1} q^{\lambda s-\mu t/2-1}(1-q)^{j-\lambda s+\mu t/2+1}$ is the coefficient of $x^{\lambda s-\mu t/2-1}$ in the polynomial $((1-q)+qx)^j$. The sum over all those terms with $j \leq B$ is the coefficient of $x^{\lambda s-\mu t/2-1}$ in the following sum.

$$\sum_{j\leq B}((1-q)+qx)^j \quad = \quad \frac{((1-q)+qx)^{B+1}}{-q(1-x)} + \frac{1}{q(1-x)}.$$

The coefficient of $x^{\lambda s-\mu t/2-1}$ in this expression is the following one.

$$\frac{1}{q} - \frac{1}{q}\sum_{i=0}^{\lambda s-\mu t/2-1}\binom{B+1}{i}q^i(1-q)^{B+1-i} = \frac{1}{q}\sum_{i=\lambda s-\mu t/2}^{B+1}\binom{B+1}{i}q^i(1-q)^{B+1-i}.$$

The sum in the term above gives the probability of having at least $\lambda s-\mu t/2$ successes in $B+1$ independent Bernoulli trials with probability of success q. Chernoff's upper bound [2] on the probability of having at least $(1+\beta)(B+1)q$ successes in such an experiment gives an upper bound of $e^{-(\beta^2(B+1)q)/3}$. We set $\lambda s-\mu t/2 = (1+\beta)(B+1)q$ in this bound and get $e^{-(\lambda s-\mu t/2-(B+1)q)^2/(3(B+1)q)}$. Let us choose function $f(N)$ so that $f(N) = (N\ \mu t)/(2\ \lambda s)$. A lower bound for $\lambda s-\mu t/2-(B+1)q$ is then derived.

$$\lambda s-\mu t/2-(B+1)q \quad \geq \quad \mu t/2-(\mu t)/(2\lambda s).$$

An upper bound for $(B+1)q$ is derived similarly.

$$\begin{aligned}(B+1)q &\leq (\lambda-1/2)-(2f(N)/N)(\lambda s-1/2)\\ &\leq (\lambda s-\mu t)-1/2+(\mu t)/(2\ \lambda s).\end{aligned}$$

We thus get for the sum of interest.

$$\sum_{j\leq B}\binom{j}{\lambda s-\mu t/2-1}q^{\lambda s-\frac{\mu t}{2}-1}(1-q)^{j-\lambda s+\frac{\mu t}{2}+1} = \frac{1}{q}\sum_{i=\lambda s-\mu t/2}^{B+1}\binom{B+1}{i}q^i(1-q)^{B+1-i} \leq e^{-(\lambda s-\mu t/2-(B+1)q)^2/(3(B+1)q)}. \quad (3)$$

The desired bound follows from equations 1, 2, and 3 and is given next.

$$\sum_{j\leq B}p(j) \quad \leq \quad \sqrt{2\pi(2\lambda s-1)}\ e^{\frac{1}{6(2\lambda s-1)}}\ e^{-\frac{(\lambda s-\mu t/2-(B+1)q)^2}{3(B+1)q}}.$$

We note that if in the third term of the above equation we substitute μt for $\lambda s-\mu t/2$ we also get the probability that set X_1 is of size at most B. Thus, by way of the above result, the probability that X_0 or X_2 is less than $N/2-f(N)$ is at most

$$\sqrt{2\pi(2\lambda s-1)}\ e^{1/(6(2\lambda s-1))}\ e^{-(\lambda s-\mu t/2-(B+1)q)^2/(3(B+1)q)}.$$

From the lower bound of $\lambda s-\mu t/2-(B+1)q$ derived earlier and the corresponding upper bound for $(B+1)q$ we get that

$$e^{-(\lambda s-\mu t/2-(B+1)q)^2/(3(B+1)q)} \leq e^{-(1/3)(\mu t/2)^2/(\lambda s)}.$$

Substituting for $\mu t = 2\sqrt{3\rho\ \lambda s\ \log n}$ gives the desired result. ■

We can similarly establish upper bounds for the sizes of the three sets X_0, X_1, X_2. The corresponding formulae are just generalizations of the results in [8]. For the size of set X_0, and by symmetry, X_2, we get that, for $B = N/2-f(N)$

$$\sum_{j\geq B}p(j) \leq \sqrt{2\pi(2\lambda s-1)}\ e^{\frac{1}{6(2\lambda s-1)}}\ e^{-\frac{(B\,q-\lambda s+\mu t/2+1)^2}{2B\,q}}.$$

Proposition 3 *Let $\lambda,\mu \geq 1$, $s,t \geq 1$, $2\lambda s-1 < N/2$, $n \geq N$, $\lambda s > \mu t/2$, and $\rho > 0$. If $\mu t = 2\sqrt{2\rho\ \lambda s\ \log n}$ then the probability that any one of the X_0, X_2 is of size more than $N/2-1$ is at most $\sqrt{2\pi(2\lambda s-1)}\ e^{1/(6(2\lambda s-1))}\ /n^{\rho}$.*

Proof: The proof follows that of proposition 2 with $f(N) = 0$. ■

4 Preliminary Results

The median of set X of n distinct elements needs to be found. Let a subset of $(2\lambda s-1) < n/2$ elements of X is chosen at random without replacement, for some integers $\lambda \geq 1$ and $s \geq 1$, and let this set in sorted order be $Y = \{y_1,\ldots,y_{2\lambda s-1}\}$, in other words, $y_i < y_{i+1}, \forall i < 2\lambda s-1$. The algorithm we are going to present, first for the sequential case, needs to compute the $(\lambda s-\mu t/2)$-th and $(\lambda s+\mu t/2)$-th elements of Y which are $y_{\lambda s-\mu t/2}$ and $y_{\lambda s+\mu t/2}$ respectively. This operation requires $O(\lambda s)$ comparisons only, by employing a deterministic selection algorithm [4]. For practical purposes, a sorting algorithm may be employed. We use sample Y to select the two splitters that split X into the three sets X_1, X_2, X_3 following the terminology of the previous section.

The objective of the splitting is to locate the median in the "middle" bucket X_1. For this to happen we need to show that the middle bucket will contain with high-probability the median. We prove the following two claims. The first one shows that with high-probability the median is in the middle bucket. The second lemma shows that the middle bucket X_1 is small by providing lower bounds for X_0, X_2.

Claim 2 *Let $\lambda, s \geq 1$, $2\lambda s-1 < N/2$, $n \geq N$, and $\rho > 1$. If $\mu, t \geq 1$ so that $\mu t = 2\sqrt{2\rho\ \lambda s\ \log n}$, and $\lambda s > \mu t/2$, then with probability at least $1-1/n^{\rho-1}$ both X_0 and X_2 are of size at most $N/2-1$ each.*

Proof: The proof follows directly from proposition 3.. We only note that the failure probability bound $1/n^{\rho-1}$ is an overestimate of a tighter one $1/n^{\rho-\eta}$, where η is any constant greater than 0.5. ■

Claim 3 *Let $\lambda, s \geq 1$, $2\lambda s - 1 < N/2$, $n \geq N$, and $\rho > 1$. Let $\mu, t \geq 1$ so that $\lambda s > \mu t/2$ and $\mu t = 2\sqrt{3\rho\, \lambda s\, \log n}$. Then, with probability at least $1 - 1/n^{\rho-1}$, both X_0 and X_2 are of sizes at least $N/2 - N\sqrt{3\rho\, \log n}/\sqrt{\lambda s}$, each.*

Proof: The claim is a direct consequence of proposition 2 after substitution of the value of $\mu t = 2\sqrt{3\rho\, \lambda s\, \log n}$ in the expression for $f(N)$. ■

The sequential algorithm below is not the best possible randomized algorithm but it will help the presentation of the BSP one.

Theorem 3 *There exists an algorithm which with probability at least $1 - 1/n^{\delta}$, for any $\delta > 0$, finds the median of n elements within $3n/2 + o(n)$ comparisons.*

Proof: Algorithm FIND_MEDIAN that computes the median of a set X of size n is given below. Initially, the variables l and r are set to zero, and S is set to the empty set.

Algorithm FIND_MEDIAN (X)

begin
Stage 1 :
Values $\lambda s = n^{1-a}$, $\rho > 1 + \delta$, $\mu t = 2\sqrt{3\rho \log n}\, n^{1/2-a/2}$, for any $a > 0$, are chosen, so that $\lambda s > \mu t/2$ and $2\lambda s - 1 < n/2$. A random sample of size $2\lambda s - 1$ is taken and the $(\lambda s - \mu t/2)$-th and $(\lambda s + \mu t/2)$-th elements of the sample are computed using the algorithm in [4]. Let $y_{\lambda s - \mu t/2}$ and $y_{\lambda s + \mu t/2}$ be the so identified elements.

Stage 2 :
For each element x of X, x is compared to $y_{\lambda s - \mu t/2}$ first. If x is smaller than or equal to $y_{\lambda s - \mu t/2}$, then $l = l + 1$. If x is larger than $y_{\lambda s - \mu t/2}$, x is compared to $y_{\lambda s + \mu t/2}$. If it is found larger than $y_{\lambda s + \mu t/2}$, then $r = r + 1$. Otherwise $S = S \cup \{x\}$.

Stage 3 :
Let $s = |S|$. If $l < n/2$, $r < n/2$ the $(n/2 - l)$-th smallest element of set S is computed using a deterministic selection algorithm [4]. Otherwise failure of the algorithm is declared and the algorithm is re-executed.
end

Analysis of the algorithm: Stage 1 of the algorithm, by the choice of λs requires $O(\lambda s) = O(n^{1-a})$ comparisons. By claim 3, for $N = n$, at least $n/2 - n^{1/2+a/2}\sqrt{3\rho \log n}$ elements have value at most $y_{\lambda s - \mu t/2}$, with probability at least $1 - 1/n^{\rho-1}$. With the stated probability the number of comparisons performed at stage 2 is at most $3n/2 + n^{1/2+a/2}\sqrt{3\rho \log n}$. The size of set S is also bounded above by $2n^{1/2+a/2}\sqrt{3\rho \log n}$. Stage 3 then, by way of [4], requires $O(n^{1/2+a/2}\sqrt{3\rho \log n})$ comparisons. Altogether, an upper bound of $3n/2 + O(n^{1-a}) + O(n^{1/2+a/2}\sqrt{3\rho \log n})$ is proved with probability at least $1 - 1/n^{\rho-1}$. ■

We note that the bound established for the sequential algorithm is minimized when $n^{1-a} = n^{1/2+a/2}\sqrt{3\rho \log n}$, which occurs when $\lambda s = n^{1-a} = n^{2/3}(3\rho \log n)^{1/3}$.

A better scheme would be the following one in the lines of [7]. In the description below we assume we can sort r elements in $r \log r$ comparisons. The addition of a constant so that $cr \log r$ comparisons are required for sorting will only affect low order terms by a constant in the description to follow.

Initially a sample Y_1 of size m is chosen. Two element a_1^1 and a_2^1 are chosen which are respectively the $((m+1)/2 \pm \sqrt{(m+1)/2}\sqrt{3\rho \log n})$-th elements of Y_1. We can sort the m elements with $m \log m$ comparisons. The elements of the sample are partitioned by these two elements into three sets X_0^1, X_1^1, X_2^1.

In the following step, a sample of $(c-1)m$ additional elements is taken which along with the elements of Y_1 constitute a sample set Y_2 of size cm. The parameter c as well as m will be fixed later. We view set Y_1 as a sample of Y_2. Then, with high-probability (for sub-polynomial choices of m) the number of elements of Y_2 between a_1^1 and a_2^1 is by claim 3, for $\lambda s = (m+1)/2$, $\mu t/2 = \sqrt{3\rho \log n}\sqrt{(m+1)/2}$, and $N = mc$, at most $2mc\sqrt{3\rho \log n}/\sqrt{(m+1)/2}$. We find these elements by performing $3(c-1)m/2 + mc\sqrt{3\rho \log n}/\sqrt{(m+1)/2}$ extra comparisons (we do not compare elements of Y_1 as the position of its elements relative to a_1^1, a_2^1 is already known).

Within this set we find two elements a_1^2, a_2^2 that are the $((mc+1)/2 \pm \sqrt{(mc+1)/2}\sqrt{3\rho \log n})$-th elements of this set. Sorting this set requires $(2mc\sqrt{3\rho \log n}/\sqrt{(m+1)/2}) \log n$ comparisons. These elements split Y_2 into three sets X_0^2, X_1^2, X_2^2.

We proceed similarly by selecting a sample Y_3 of size mc^2 by sampling $mc(c-1)$ extra elements, and in the i-th iteration we need a total of

$$\frac{3c^{i-1}(c-1)m}{2} + \frac{\sqrt{3\rho \log n}\, mc^i}{\sqrt{(c^{i-1}m+1)/2}} + \frac{2 \log n \sqrt{3\rho \log n}\, mc^i}{\sqrt{(c^{i-1}m+1)/2}}$$

comparisons for the various steps to split Y_i into the three sets X_0^i, X_1^i, X_2^i.

Let us fix $m = n^a$, $a < 1/2$. Summing over all iterations up to say iteration t so that $mc^t = n$, $c > 1$, we get a total number of comparisons which is bounded above by $3n/2 + O(\sqrt{3\rho} n^{1/2+(1-a)/(2t)} \log^{3/2} n)$. Set X_1^t contains with high-probability the median and it can be found in time linear to its size within $O(\sqrt{3\rho} n^{1/2+(1-a)/(2t)} \log^{1/2} n)$ comparisons.

We can make the total comparison count $3n/2 + O(n^{1/2+\delta})$ for any chosen constant δ by selecting the constant values a, t so that $\delta > (1-a)/(2t)$. Smaller values of m, c require a t logarithmic in n and the smallest value of m to be limited by the choices in claim 3. The probability of failure per step is at most $1/n^{\rho-1}$ and since we have a constant number of steps (or at most logarithmic in n) we can make the failure probability arbitrarily small by choosing an appropriate constant $\rho > 1$. The complete proof will appear in the full paper.

5 A BSP Algorithm

In this section we show how algorithm FIND_MEDIAN presented in the previous section can be modified to give a BSP algorithm for the median finding problem. The number of processors of the BSP machine is denoted by p.

In the BSP implementation the problem of distribution of the input elements to the processors becomes important. If the input elements are randomly and evenly distributed among the processors, then the BSP algorithm to be presented will, with high-probability, not only be work-optimal

in the sense that the processors will perform $3n/2 + o(n)$ comparisons altogether to find the median, but also optimal, provided $n/p = \omega_n \log n$, where ω_n is any slowly growing function of n such that $\omega_n \to \infty$ as $n \to \infty$. By optimality in the latter case it is implied that with high-probability each processor will perform a maximum of $(3n/(2p)) + o(n/p)$ comparisons.

If the elements are not initially randomly distributed among the processors we get a work-optimal algorithm that requires however, a maximum of $2n/p + o(n/p)$ comparisons per processor leading to a 4/3-optimal algorithm. In this latter case, and for logarithmic slack, we can still achieve optimality at the expense of a randomization stage in the beginning of the algorithm that distributes elements randomly among the processors, each processor receiving with high-probability at most $(1+\delta)n/p$ elements, for some $\delta > 0$, that depends on the slack n/p (the proof is by using Chernoff's bounds).

We describe the algorithm in detail below. We call a number of supersteps that perform a certain task a stage. The notion of a stage is unrelated otherwise to the definition of the BSP model. The algorithm presented below works only for a limited range of p relative to n. This facilitates the easier exposition of the algorithm. Subsequently, we show how we can modify two of its stages to get an algorithm that works for greater ranges of p. The size of the sample we select is set to $2\lambda s - 1 = 2n^{2/3}(3\rho \log n)^{1/3} - 1$ for an algorithm that will fail with probability at most $1/n^{\rho-1}$. Such a choice of the sample limits the median within a bucket of size $2n^{2/3}(3\rho \log n)^{1/3}$.

Algorithm FIND_MEDIAN_BSP (X,n)

begin
Stage 1 :
If the number of processors p exceeds $k = 2n^{2/3}(3\rho \log n)^{1/3}$, terminate. Otherwise, select a sample of overall size $2\lambda s - 1 = 2n^{2/3}(3\rho \log n)^{1/3} - 1$.

The sample selection method is as follows.

a) Each processor selects approximately the same number of processor identifiers uniformly at random compensating for the fact that one processor may have one more element than another one. Altogether a sample of size $2\lambda s - 1$ is selected.

b) The processor identifiers selected are routed to the identified processors.

c) Each processor i that receives t_i identifiers selects among its elements uniformly at random without replacement t_i elements.

d) A parallel prefix on the number t_i of elements selected per processor is performed.

e) The result of the parallel prefix is used to evenly distribute the selected sample among the processors.

[**Analysis of Stage 1:** One comparison is required for the program to decide whether to terminate or not. This is the only comparison performed in this stage. In step (a) at most $\lceil 2\lambda s/p \rceil$ time is spent on each processor for random processor identifier generation.

In step (b) we distinguish two cases in the analysis.

If $p \leq \lambda s/(1.5\rho \log n)$, and since processor identifiers are selected uniformly at random, the bound in [2] on the right tail of the Binomial distribution shows that with probability $1 - O(1/n^{\rho-1})$ no processor gets more than $2\lceil 2\lambda s/p \rceil$ processor identifiers.

Otherwise, for each value of a processor identifier, the average number of processor identifiers drawn is at most $1.5\rho \log n$. We use the generalized bound due to Chernoff [6] $\text{Prob}[S_{n,p} \geq upn] \leq (e/u)^{upn}$, $pn \geq 1, u(1-p) > 2$, where $S_{n,p}$ is the number of successes in n independent Bernoulli trials with individual probability of success p. We can then prove that each processor gets no more that $3e\rho \log n$ processor identifiers with probability at least $1 - n^{3\rho}$. Over all, a max $\{2\lceil 2\lambda s/p \rceil, 3e\rho \log n\}$-relation needs to be routed.

Step (c) requires at most max $\{2\lceil 2\lambda s/p \rceil, 3e\rho \log n\}$ time. Step (d) requires $2l$ rounds, each round consisting of two supersteps, one for routing a $\lceil p^{1/l} \rceil$-relation in g max$\{\frac{L}{g}, \lceil p^{1/l} \rceil\}$ time, and one for computation of time max$\{L, \lceil p^{1/l} \rceil\}$. We use a $\lceil p^{1/l} \rceil$-ary tree for parallel prefix. Step (e) requires the routing of a max $\{2\lceil 2\lambda s/p \rceil, 3e\rho \log n\}$-relation.]

Stage 2 :
The sample is sorted by employing the BSP sorting algorithm of [8] resulting in a sorted sequence $y_1 < y_2 < \ldots < y_{2\lambda s-1}$.

[**Analysis of Stage 2:** If $p = (\lambda s)^{1-t}$ for some constant $0 < t < 1$, the algorithm summarized in theorem 5 of [8] can be used, and for $L = o(n/p)$ the sample sorting of stage 2 requires computation time $o(n/p)$, and communication time $O(g(\lambda s/p))$. Otherwise we can sort using a fraction of the processors, say p/n^{δ}, for any constant $0 < \delta < 1/3$. The same BSP sorting algorithm is employed that gives with $L = o(n/p)$ an algorithm that requires computation time $o(n/p)$ and communication time $O(g(n^{\delta}\lambda s/p))$.

We comment that if in the latter case we employ the algorithm summarized in theorem 6 of [8], we can allow for smaller values of L but larger values of g. We also note that one does not need to sort but only select two elements from the sample.]

Stage 3 :
Let $\mu t/2 = n^{1/3}(3\rho \log n)^{2/3}$. The elements $y_{\lambda s - \mu t/2}$ and $y_{\lambda s + \mu t/2}$ of the sorted sample are broadcast to all other processors.

[**Analysis of Stage 3:** A broadcast in l supersteps, each such superstep requiring the routing of a $2\lceil p^{1/l} \rceil$-relation is performed.]

Stage 4 :
The elements stored in each processor, say processor i, are compared with $y_{\lambda s - \mu t/2}$ and $y_{\lambda s + \mu t/2}$ using the comparison sequence outlined in Step (b) of algorithm FIND_MEDIAN of the previous section. Local copies of two variables l_i, r_i which count the number of elements local to processor i that are less than $y_{\lambda s - \mu t/2}$ and greater than $y_{\lambda s + \mu t/2}$ are also updated.

[**Analysis of Stage 4:** This stage requires only one superstep. We estimate the number of comparisons required by taking into consideration the discussion in the analysis of the performance of the sequential algorithm related to the number of elements less than $y_{\lambda s - \mu t/2}$ and greater than $y_{\lambda s + \mu t/2}$. Because of the choice of n and p, the number of elements of the input between these two splitters is just $2\lambda s$ with probability at least $1 - 1/n^{\rho-1}$. Sine the elements of the input are randomly distributed among the processors we can employ the arguments used to argue about the processor identifiers in stage 1, to show that with probability at least $1 - 1/n^{\rho-1}$ each processor has in its local memory at most max $\{2\lceil 2\lambda s/p \rceil, 3e\rho \log n\}$ such elements. As each processor has at least $\lfloor n/p \rfloor$ elements altogether, the same

argument used in the analysis of the sequential argument shows that each processor with the stated probability performs $3n/(2p) + o(n/p)$ comparisons.]

Stage 5 :
A parallel prefix on the l_i's and on the r_i's is performed.
[**Analysis of Stage 5:** The time complexity of this step is similar to step (d) of stage 1.]

Stage 6 :
A broadcast of the two sums $\sum_i l_i$ and $\sum_i r_i$ is initiated.
[**Analysis of Stage 6:** The two sums were computed in the parallel prefix operation of stage 5. The time complexity of this step is that of stage 3.]

Stage 7 :
Let the elements greater than $y_{\lambda s - \mu t/2}$ and less than $y_{\lambda s + \mu t/2}$ form set X'. The information collected from the prefix operations in stages 5 and 6 is sufficient for the processors to proceed with the computation of the $(\lceil n/2 \rceil - \sum_i l_i)$-th smallest element of X', the median of X.
[**Analysis of Stage 7:** The size of X' is at most $2\lambda s = 2n^{2/3}(3\rho \log n)^{1/3}$. The elements to be sorted are first randomly distributed among the processors so that each one of them gets the same number of elements. All previous operations preserved the relative positions of the initially randomly distributed elements, i.e. elements of X' are randomly distributed. The information of stages 5 and 6 can be used to evenly distribute these elements to the processors. From the analysis of stage 4 a $\max\{2\lceil 2\lambda s/p \rceil, 3e\rho \log n\}$-relation is to be routed. The number of elements to be sorted is $2\lambda s$, the same as in stage 2, and the analysis of stage 2 applies here as well.]
end

Thus, the BSP algorithm requires, with high-probability, $3n/2 + o(n)$ comparisons, or equivalently the algorithm is one-optimal as compared to the best expected time sequential of [7]. The following theorem is then derived.

Theorem 4 *There is a BSP median finding algorithm that for any n, p, L, and $\rho > 1$ satisfying $p \leq n^{2/3}(3\rho \log n)^{1/3}$, and $L = o(n/p)$ has with probability $1 - O(1/n^{\rho - 1})$, $\pi = 1 + o(1)$. If $g = o(n^{1/3-\delta}/\log^{1/3} n)$, for any constant $\delta < 1/3$, then also $\mu = o(1)$.*

For larger values of p with respect to n as long as $p = n^{1-t}$, for some constant $0 < t < 1$, we can still use the same algorithm with one modification related to the number of supersteps l per broadcast/parallel prefix operation. Comparing the maximum possible values of g required for broadcasting and for the communication in stages 2 and 7, if for given t, we choose constants δ, l such that $(1-t)(1+1/l) < 2/3 + \delta$, then $p^{1+1/l} < \lambda s n^{\delta}$ and the value of g given in the theorem is still valid. We also note that in this case, in the cost analysis, we use the following bound related to the right tail of the binomial distribution $\text{Prob}[S_{n,p} > eupn/\log u] < e^{-upn}$, $u \geq e$ and $u^2 pn \geq \log u$ as found in [6].

For polylogarithmic slack we use techniques previously employed in the analysis of the algorithm implied in theorem 6 of [8]. We describe this variant in the full paper. We also describe general selection BSP algorithms which involve reduced low-order terms by way of using smaller sample sizes.

6 Experimental Results

Below we present indicative experimental results for the two algorithms. We used the Oxford BSP Library Version 1.2. For each test problem instance, we performed a series of 8 experiments, and the running times indicated are the average of those 8 experiments. We tested our algorithms on two different machines with a varying number of nodes and input sizes. Figures 1 and 2 compare the optimal and actual speedup of deterministic sorting for 32,000 to 1,024,000 elements. Similarly Figures 3 and 4 compare the optimal and actual speedup of randomized median finding for 512,000 to 2,048,000 elements.

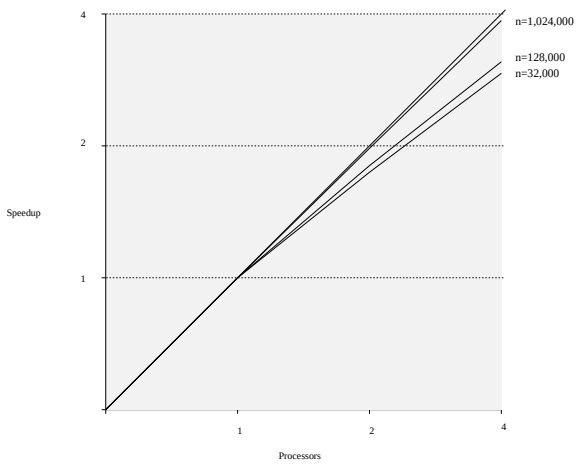

Figure 1: Speedup achieved by deterministic sorting on a 4 node Silicon Graphics Power Challenge shared memory system.

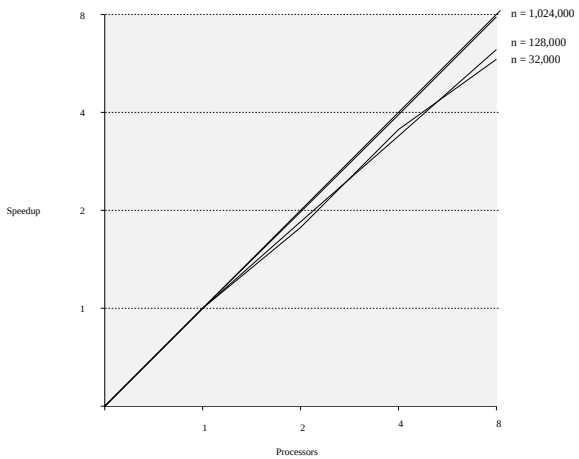

Figure 2: Speedup achieved by deterministic sorting on an 8 node IBM 9076 SP1 close-coupled distributed memory system.

7 Acknowledgements

We would like to thank the anonymous referees for their comments and suggestions and Bill McColl for his encouragement and support during this work.

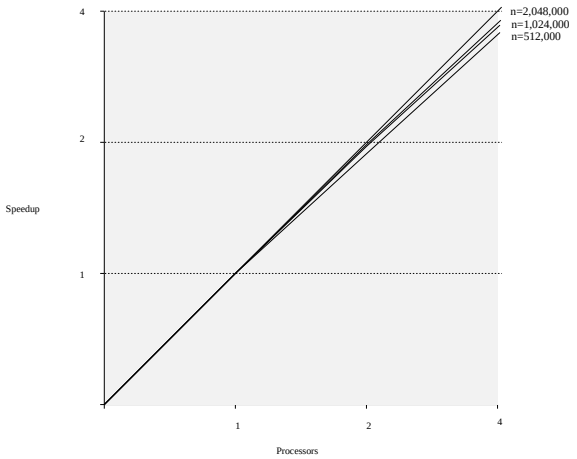

Figure 3: Speedup achieved by randomized median finding on a 4 node Silicon Graphics Power Challenge shared memory system.

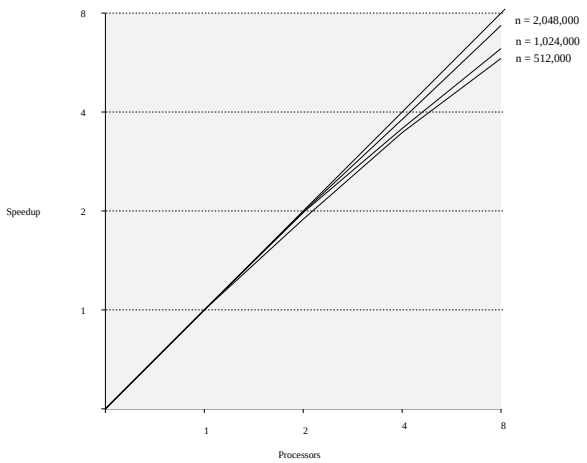

Figure 4: Speedup achieved by randomized median finding on an 8 node IBM 9076 SP1 close-coupled distributed memory system.

References

[1] M. Adler, J. W. Byers, and R. M. Karp. Parallel Sorting with Limited Bandwidth. In *Proceedings of the 7-th ACM Symposium on Parallel Algorithms and Architectures,* pp. 129-136, 1995.

[2] D. Angluin and L. G. Valiant. Fast probabilistic algorithms for Hamiltonian circuits and matchings. *J. Computer and System Sciences*, 18:155-193, 1979.

[3] K. Batcher. Sorting Networks and their applications. In *Proceedings of the AFIPS Spring Joint Computing Conference,* pp. 307-314, 1968.

[4] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest and R. E. Tarjan. Time bounds for selection. *J. Computer and System Sciences*, 7:448-461, 1973.

[5] W. Cunto and J. Ian Munro. Average case selection. *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing* pp. 369-375, 1984.

[6] Béla Bollobás. *Random Graphs.* Academic Press, New York, 1985.

[7] R. W. Floyd and R. L. Rivest. Expected time bounds for selection. *Communications of the ACM*, 18:165-172, 1975.

[8] A. V. Gerbessiotis and L. G. Valiant. Direct Bulk-Synchronous Algorithms. *Journal of Parallel and Distributed Computing,* 22:251-267, 1994.

[9] J. S. Huang and Y. C. Chow, Parallel Sorting and Data Partitioning by Sampling. *Proceedings of the IEEE Computer Society's 7-th International Computer Software and Applications Conference,* pp. 627-631, 1983.

[10] L. Hyafil. Bounds for selection. *SIAM Journal of Computing* 5:109-114, 1976.

[11] D. G. Kirkpatrick. A unified lower bound for selection and set partitioning problems. *Journal of the ACM* 28:150-165, 1981.

[12] D. E. Knuth. *The Art of Computer Programming. Volume I: Fundamental Algorithms.* Addison-Wesley, Reading, 1969.

[13] D. E. Knuth. *The Art of Computer Programming. Volume III: Sorting and Searching.* Addison-Wesley, Reading, 1973.

[14] F. T. Leighton. Tight Bounds on the complexity of parallel sorting. *IEEE Transactions on Computers,* C-34(4):344-354, 1985.

[15] W. F. McColl. General purpose parallel computing. In *Lectures on parallel computation,* (Gibbons, A. and Spyrakis, P., eds.), Cambridge University Press, 1993.

[16] W. F. McColl. Scalable parallel computing: A grand unified theory and its practical development. In *Proceedings of IFIP World Congress*, 1:539-546, Hamburg, August 1994.

[17] J. I. Munro and P. V. Poblete. A lower bound for determining the median. University of Waterloo Research Report CS-82-21, 1982.

[18] S. Rajasekaran. Randomized parallel selection. In *10-th Conference on Foundations of Software Technology and Theoretical Computer Science*, pp. 215-224, LNCS 472, Springer-Verlag 1990.

[19] R. Reischuk. Probabilistic parallel algorithms for sorting and selection. *SIAM Journal on Computing*, 14(2):396-409, 1985.

[20] A. Schonhage, M. Paterson, and N. Pippenger. Finding the median. *J. Computer and System Sciences*, 13:184-199, 1976.

[21] H. Shi and J. Schaeffer. Parallel sorting by regular sampling. *Journal of Parallel and Distributed Computing,* 14:362-372, 1992.

[22] L. G. Valiant. General purpose parallel architectures. In *Handbook of Theoretical Computer Science* (J. van Leeuwen, ed.), North Holland, pp. 945-971, 1990.

[23] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33:103-111, 1990.