Programming Research Group

CONCURRENT HEAPS ON THE BSP MODEL

Alexandros V. Gerbessiotis and Constantinos J. Siniolakis

PRG-TR-14-96

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD

Concurrent Heaps on the BSP Model*

Alexandros V. Gerbessiotis and Constantinos J. Siniolakis

Computing Laboratory,
Oxford University,
Wolfson Building,
Oxford, OX1 3QD,
UK.

(Revised Version)

October 1996

Abstract

In this paper we present a new randomized selection algorithm on the Bulk-Synchronous Parallel (BSP) model of computation along with an application of this algorithm to dynamic data structures, namely Parallel Priority Queues (PPQs). We show that our algorithms improve previous results upon both the communication requirements and the amount of parallel slack required to achieve optimal performance. We also establish that optimality to within small multiplicative constant factors can be achieved for a wide range of parallel machines. While these algorithms are fairly simple themselves, descriptions of their performance in terms of the BSP parameters is somewhat involved. The main reward of quantifying these complications is that it allows transportable software to be written for parallel machines that fit the model. We also present experimental results for the selection algorithm that reinforce our claims.

Keywords. BSP model - algorithms - selection - heaps - priority queues

*The work of the first author was supported in part by EPSRC(UK) under grant GR/K16999 and the second author was supported in part by a Bodossaki Foundation Graduate Scholarship. An extended abstract of a preliminary version of part of this work was presented in the 1996 International Conference on Parallel and Distributed Processing Techniques and Applications, Sunnyvale, CA.

1 Introduction and the BSP Model

The main technical contribution of this work is the architecture independent study of the computation and communication requirements of a fundamental combinatorial problem namely, selection, along with an application to Parallel Priority Queues (PPQs). A PPQ is an abstract data structure that generalizes the well-known sequential priority-queue data structure [21]. A p-processor PPQ implementation allows the efficient simultaneous insertion of n new items, n/p per processor, and the simultaneous deletion of the n items associated with the smallest values. As noted in [27], PPQs can be employed to efficiently implement several parallel techniques, such as *branch-and-bound* [16, 17, 18], which require the solution of several sub-problems, each associated with a possibly different cost. By utilizing PPQs, the p processors can insert at each phase the newly generated sub-problems and efficiently select the n "global best" sub-problems to be solved in the next phase. In this work we establish time bounds for PPQ operations as a function of the available communication throughput of the system.

The computational model adopted is the Bulk-Synchronous Parallel (BSP) model, which deals explicitly with the notion of communication and synchronization among computational threads. The BSP model was originally proposed by L. G. Valiant [38] as a bridge between software and hardware.

1.1 The BSP model

The main reason for the moderate success of parallel computing has been the lack of a bridging and unifying model for parallel computation the way the von Neumann model has served sequential computing for the last fifty years. The introduction of the *Bulk-Synchronous Parallel* (BSP) model of computation by L.G. Valiant addresses these limitations. A BSP computer as introduced in [36, 37, 38] and also examined in [10, 22, 23, 24, 39] consists of three parts: (i) a *collection of processor-memory components*, (ii) a *communication network* that can deliver messages point-to-point among the components, and (iii) facilities for global synchronization, in barrier style, of all or subsets of the components.

A BSP computer is parametrized by

(1) p - the number of processor-memory components,

(2) L - the minimal time, measured in terms of basic computational operations, between successive synchronization operations, and

(3) 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 on the BSP model proceeds in a succession of *supersteps*. A superstep may be thought of as a segment of a process during which each processor performs a given task using data already available there locally before the start of the superstep [23]. The task may include (i) local computations, (ii) message transmissions, and (iii) message receipts. Each superstep is charged $\max\{L, x + gh\}$ time steps, where x is the maximum number of basic computational operations executed by any processor during a superstep and h is the maximum number of messages transmitted or received by any processor. In the remainder of this paper we use $\log x$ and $\lg x$ to denote the logarithm of x to the base e and 2 respectively.

1.2 Related Work and New Results

The selection problem that is, the computation of the C-th smallest key from a set of n keys has always been an interesting research topic [4, 19, 20]. Median finding in linear time is examined in [4, 34] and lower bounds for various values of C are also presented in [4, 14, 19, 25]. The minimum average number of comparisons required to find the C-th largest of n keys has also drawn considerable interest. A lower bound of $n+C-O(1)$ comparisons, $1 \leq C \leq \lceil n/2 \rceil$, is presented in [7], which matches the performance, within low order terms, of the algorithm in [9]. A sequential randomized algorithm for median finding that can be further extended to a parallel one is proposed in [11]. The sequential version requires $3n/2+O(n^{2/3}\log^{1/3} n)$ comparisons, with high probability. An improved sequential median finding algorithm based on a bootstrapping technique of [9] is also suggested in [11] that requires with high probability $3n/2 + O(n^{1/2+\beta})$ comparisons, for any constant $\beta > 0$. In this work we generalize these median finding results to the general selection problem

	Computation			
Reference	INSERT	DELETEMIN	FINDMIN	Max Processors
[27]	$O(\frac{n}{p}\lg\frac{m}{n}+\frac{n}{p}\lg n)$	$O(\frac{n}{p}\lg\frac{m}{n}+\lg p)^*$	$O(1)$	n
[30]	$O(\frac{n}{p}\lg m)^*$	$O(\frac{n}{p}\lg m)^*$	$O(\frac{n}{p}\lg m)^*$	$\lg m$
[29]	$O(\frac{n}{p}\lg\frac{m}{p})$	$O(\frac{n}{p}\lg\frac{m}{p}+\frac{n}{p}\lg n)$	$O(\frac{n}{p}\lg\frac{m}{p}+\frac{n}{p}\lg n)^*$	$\frac{n}{\lg n}$
This work and [29]	$O(\frac{n}{p}+\lg\frac{m}{p})$	$O(\frac{n}{p}\lg\frac{m}{p}+\lg p)$	$O(\frac{n}{p}\lg\frac{m}{p}+\lg p)$	$\frac{n}{\lg n}$
This work	$O(\frac{n}{p}\lg\frac{m}{n}+\lg\frac{m}{n}\lg p)$	$O(\frac{n}{p}\lg\frac{m}{n}+\lg\frac{m}{n}\lg p)$	$O(1)$	$\frac{n}{\lg n}$
This work (amortized)	$O(\frac{n}{p}\lg\frac{m}{n}+\lg p)$	$O(\frac{n}{p}\lg\frac{m}{n}+\lg p)$	$O(1)$	$\min\{n,\frac{n}{\lg n}\lg\frac{m}{n}\}$

Table 1: Computation results on PPQs. A * sign indicates that the associated result has been adapted to fit our model.

and we extend them by deriving parallel implementations. The parallel algorithms we present, work on the BSP model of computation and require $(n+C)/p+o(n/p)$ parallel time, with probability $1-n^{-c}$, for any constant $c>0$. Previous parallel selection algorithms include the results in [28, 32]. Our techniques provide tighter probabilistic bounds and optimal number of comparisons, within lower order terms, on a realistic parallel computational model. We also show that these results can be extended to efficiently solve a more interesting problem that of parallel priority queues subject to the operations of insert, find-min and delete-min [3, 8, 15, 18, 27, 29, 30].

Previous approaches based on traditional *concurrent* priority queues [3, 15, 30] provide only limited parallelism. In this abstract, we concern ourselves with *parallel* priority queues that offer the prospect of a high degree of parallelism. The first approach on parallel priority queues is implied in the work of Karp and Zhang [17, 18]. They derive a parallel randomized algorithm to approximate a priority queue. Their algorithm is extended to handle exact priority queues in [29, 33]. In [29], it is shown that each of the insert and delete-min operations can be performed (for a set of n items) in optimal $O(n^{1/d})$ time on an n-processor d-dimensional mesh. It is also shown how to implement deterministic priority queues on n-processor d-dimensional meshes that require $O(n^{1/d}\lg^{1-1/d}n)$ time to perform n insert or delete-min operations.

Pinotti and Pucci [27] derive deterministic algorithms for maintaining parallel priority queues. They introduce two new data structures, the n-Bandwidth-Heap and the n-Bandwidth-Leftist-Heap, that are obtained as extensions of the sequential binary-heap and leftist-heap. Using these structures, they show that the insertion of n items in a parallel priority queue of size $m\geq n$ can be performed in $O(\lg(m/n))$ time on an n-processor CREW machine. Similarly, they show that the deletion of the n smallest items can be performed in $O(\lg(m/n)+\lg\lg n)$ time.

These approaches fail, however, to deliver optimal performance when directly implemented on the Bulk-Synchronous Parallel model of computation. In this work we improve upon the communication requirements of previous results on parallel priority queues [27], and extend the randomized results of [18, 29, 33] by improving upon both the communication requirements and the amount of parallel slack required to achieve optimal performance. We also show that optimality to within small multiplicative constant factors can be achieved for a wide range of the BSP parameters. The randomized algorithms we present are based on a new randomized processor-mapping of a variant of n-Bandwidth-Heaps. In particular, the underlying data structure is a mixture of n-Bandwidth-Heaps [27] and concurrent (pipelined) heaps [30], and is based on an efficient selection algorithm to achieve low contention.

These results are summarized in tables 1 and 2. With reference to these tables, we assume an exclusive read exclusive write parallel model of computation. All algorithms operate on a data structure of size $m\geq n$ for sets of n items (we assume $m\leq 2^{n^c}$, for any constant $c>0$). Finally, the reference "this work and [29]" relates to the algorithm in [29] implemented using the techniques introduced in this work.

Independently with this work similar results on an extension of the BSP model called BSP* have been obtained in [2]. The BSP* model takes into explicit account the size of communicated messages. The original BSP model can also take message size into consideration by absorbing message size, for small messages, into one of its parameters, L, and for larger messages, into another parameter, g. Our results

	Communication			
Reference	INSERT	DELETEMIN	FINDMIN	Max Processors
[27]	$O(\frac{n}{p}\lg\frac{m}{n}+\frac{n}{p}\lg n)$	$O(\frac{n}{p}\lg\frac{m}{n}+\lg p)^*$	$O(1)$	n
[30]	$O(\frac{n}{p}\lg m)^*$	$O(\frac{n}{p}\lg m)^*$	$O(\frac{n}{p}\lg m)^*$	$\lg m$
[29]	$O(\frac{n}{p})$	$O(\frac{n}{p}\lg n)$	$O(\frac{n}{p}\lg n)^*$	$\frac{n}{\lg n}$
This work and [29]	$O(\frac{n}{p})$	$O(\lg p)+o(\frac{n}{p})$	$O(\lg p)+o(\frac{n}{p})$	$\frac{n}{\lg n}$
This work	$O(\frac{n}{p}+\lg\frac{m}{n}\lg p)+$ $o(\frac{n}{p}\lg\frac{m}{n})$	$O(\lg\frac{m}{n}\lg p)+$ $o(\frac{n}{p}\lg\frac{m}{n})$	$O(1)$	$\frac{n}{\lg n}$
This work (amortized)	$O(\frac{n}{p}+\lg\frac{m}{n}+\lg p)+$ $o(\frac{n}{p}\lg\frac{m}{n})$	$O(\frac{n}{p}+\lg\frac{m}{n}+\lg p)+$ $o(\frac{n}{p}\lg\frac{m}{n})$	$O(1)$	$\min\{n, \frac{n}{\lg n}\lg\frac{m}{n}\}$

Table 2: Communication results on PPQs. A * sign indicates that the associated result has been adapted to fit our model.

compare favorably to those in [2] in various ways. The selection algorithm presented in [2] requires $O(n/p)$ computation time as opposed to the $(n+C)/p+o(n/p)$ in this work. The communication time of the algorithm in [2] is also higher than the one in this work. As a consequence, the priority queue algorithms presented in [2] are slower in both computation and communication. In addition, our algorithms work for values of p much closer to n than those of [2] even when differences in the two models are taken into consideration.

2 Primitive Operations

In this section we state results on BSP algorithms for the fundamental operations of *broadcasting* and *parallel-prefix* [12]. The proofs of these results can be found in [12]. These primitive operations are auxiliary routines for the randomized selection and priority-queue algorithms presented in the following sections.

Lemma 1 *There exists a BSP algorithm for broadcasting an n-word message that requires time at most $M^n_{brd}(p) = (\lceil n/\lceil n/h\rceil\rceil + h - 1)\max\{L, gt\lceil n/h\rceil\}$, for any integer $2 \le t \le p$, where $h = \lceil \log_t((t-1)p+1)\rceil - 1$, for a total time of $T^n_{brd}(p) = M^n_{brd}(p)$.*

Lemma 2 *There exists a BSP algorithm for computing n independent parallel-prefix operations that requires time at most $C^n_{ppf}(p) = 2(\lceil n/\lceil n/h\rceil\rceil + h - 1)\max\{L, t\lceil n/h\rceil\}$ and $M^n_{ppf}(p) = 2(\lceil n/\lceil n/h\rceil\rceil + h - 1)\max\{L, 2gt\lceil n/h\rceil\}$, for computation and communication respectively, for any integer $2 \le t \le p$, where $h = \lceil \log_t p\rceil$, for a total time of $T^n_{ppf}(p) = C^n_{ppf}(p) + M^n_{ppf}(p)$.*

If $n = 1$, then we write $T_{ppf}(p)$ (respectively $C_{ppf}(p)$, $M_{ppf}(p)$) for $T^1_{ppf}(p)$ (respectively $C^1_{ppf}(p)$, $M^1_{ppf}(p)$). The same notational convention applies to all pipelined operations.

2.1 Randomized BSP Sorting ($p > n$)

We present a new randomized BSP sorting algorithm that works for the case where the number of available processors p is greater than the number of keys n to be sorted and derives from ideas of [26]. This algorithm is an auxiliary routine of the randomized selection algorithm presented later in this work. The fundamental difference between our algorithm and the deterministic ones in [26] is that it is based on random-sampling techniques [10] and utilizes exclusively broadcast, parallel-prefix and regular-routing operations.

The general formulation of our algorithm is as follows. It selects a sample of size at most $\sqrt{p}$ and sorts the sample by performing all possible $\sqrt{p}(\sqrt{p}-1)/2$ comparisons among the keys in constant time utilizing all p processors. A parallel-prefix operation on the results of the comparisons determines the rank of each key in the final sorted sequence [26]. The time of this sorting step is thus dominated by that of performing parallel-prefix operations. A set of $k-1=o(\sqrt{p})$ splitters is then chosen. The input keys are divided into k buckets by a search procedure that requires local computations, parallel-prefix and regular-routing operations. The k buckets are then sorted recursively by allocating the same number of processors to each bucket. The recursion terminates when for each bucket its size is at most the square root of the number of processors allocated to it. The algorithm previously employed for sample sorting is used to sort the keys of each bucket.

2.1.1 A BSP Searching Algorithm

The aforementioned randomized BSP sorting algorithm employs a search procedure that allows the position of each input key with respect to the $k-1$ splitters to be found efficiently by using local computations, parallel-prefix and regular-routing operations. The search problem we solve in this section can be defined as follows.

Problem: *Let N distinct input keys $X=\{x_1,\ldots,x_N\}$ be given, so that each x_i is held by a single processor. A sorted sequence of $k=o(N)$ distinct keys $Y=\langle y_1,\ldots,y_k\rangle$ in ascending order is also given, where $y_k=\max\{x_1,\ldots,x_N\}$, so that each y_i is held by a single processor. Determine the rank of x_i, for all i, $1\le i\le N$, in Y, where the total number of available processors is $p=Nk^{1/r}$, for some positive integer $r>0$.*

Without loss of generality we assume that $k^{1/r}$ is an integer, otherwise $\lceil k^{1/r}\rceil$ is used instead. We may view the search as drawn on an $k^{1/r}$-ary balanced search tree. The *depth* of this tree is r. Its root is assumed to be at *level* 0 and its leaves at level r. The number of nodes at level I is $k^{I/r}$ and each internal node has $k^{1/r}$ children. Associated with each such node is a key value and an index, where the j-th node (in an left to right indexing) of level I of the tree has key value $y_{jk^{1-I/r}}$ and index $jk^{1-I/r}$, $1\le j\le k^{I/r}$. If it is an internal node, then it also has $k^{1/r}$ children whose key values are $y_{jlk^{1-(I+1)/r}}$, $1\le l\le k^{1/r}$. The root of the tree has key value y_k. If the index of a node is thus given, the index of any of its children can be computed in constant time using simple arithmetic operations.

For the sake of exposition we number processors in a mesh-like fashion, so that we can refer to processor at row i and column j, as $p_{i,j}$, where $1\le i\le N$ and $1\le j\le k^{1/r}$. We assume that processor $p_{i,1}$ holds, in some register say z_i, key x_i, $1\le i\le N$, and processor $q_l=p_{l,1}$ holds key y_l, $1\le l\le k$.

Sequence Y induces a partitioning of the set of input keys into k "buckets". Our goal is to determine for each input key its corresponding bucket. The search proceeds from the root to the leaves of this $k^{1/r}$-ary tree. We associate with each input key the index of its associated bucket. Initially, the bucket index of every key is set to zero, which is to mean that all keys are currently in a single bucket and smaller than the key value of the root of the tree. At each iteration of the search procedure a refined value of each such bucket index is computed and the N keys are rearranged accordingly among the processors so that consecutively indexed processors hold keys having the same bucket index. After r iterations the bucket index of each key points to the bucket it belongs to.

The procedure to carry out the aforementioned process is called SEARCH and is described below. The parameters of procedure SEARCH are as follows. The first two parameters identify the leftmost and rightmost index of the input keys participating in the current iterative call of the procedure. The procedure is initially called with *Left* $=0$ and *Right* $=N-1$, where N is the number of keys to be sorted. The total number of processors involved in any particular call of the procedure is $(Right-Left+1)k^{1/r}$. The third parameter v identifies the key value of a tree node at level I whose index is d. Initially, register z_i of processor $p_{i,1}$, $1\le i\le N$, holds key value x_i, and $v=y_k$, $d=k$ and $I=0$. In any subsequent iterative call of the procedure, the keys held by processors $\langle p_{Left+1,1},\ldots,p_{Right+1,1}\rangle$ are smaller than or equal to v. The procedure is iterative and the number of iterations is r.

Procedure SEARCH (*Left*, *Right*, v, d, I)

begin
Stage 1 :
Let $N_m=Right-Left+1$ be the number of input keys participating in this iterative call. The node at level I with key value $v=y_d$ and index d has $k^{1/r}$ children with key values and indices $s_j=y_{djk^{-1/r}}$ and

$djk^{-1/r}$, $1 \leq j \leq k^{1/r}$, respectively. The $k^{1/r}$ processors holding these keys, among the ones identified as q_l, $1 \leq l \leq k$, become active.
[**Analysis of Stage 1:** This stage requires time $\max\{L, O(1)\}$, since, given I, each processor among the k processors identified as q_l, $1 \leq l \leq k$, can identify in constant time whether it holds a particular key with key value s_j and index $djk^{-1/r}$, $1 \leq j \leq k^{1/r}$.]

Stage 2 :
Let $p_{Left+i,j}$ be the $N_m k^{1/r}$ processors allocated to this iterative call of procedure SEARCH, $1 \leq i \leq N_m$, $1 \leq j \leq k^{1/r}$. Key with value s_j is broadcast to processors $p_{Left+i,j}$, for all $1 \leq i \leq N_m$, and subsequently, the contents of register z_{Left+i} held by processor $p_{Left+i,1}$ are broadcast to processors $p_{Left+i,j}$, $1 \leq j \leq k^{1/r}$.
[**Analysis of Stage 2:** The number of processors allocated to this call is $N_m k^{1/r} \leq p$. Thus, the cost of this stage is $2T_{brd}(p) \leq T_{ppf}(p)$.]

Stage 3 :
Processor $p_{Left+i,j}$ compares z_{Left+i} to s_j. If $z_{Left+i} \leq s_j$ a prespecified register of $p_{Left+i,j}$ is set to 1, otherwise, the register is set to 0.
[**Analysis of Stage 3:** This stage requires time $\max\{L, O(1)\}$.]

Stage 4 :
The rank of z_{Left+i} in the sequence $\langle s_1, \ldots, s_{k^{1/r}} \rangle$ is evaluated at processor $p_{Left+i,k^{1/r}}$, by performing simultaneously N_m parallel-prefix operations in a row-major direction. If the so evaluated rank is t, processor $p_{Left+i,t}$, for all $1 \leq i \leq N_m$, stores a 1 in a prespecified register, whereas all remaining processors store a 0.
[**Analysis of Stage 4:** The number of processors allocated to this call is $N_m k^{1/r} \leq p$. Thus, the time complexity of this stage is $T_{ppf}(p) + \max\{L, O(1)\}$.]

Stage 5 :
The size l_j of each bucket is computed in processors $p_{Left+N_m,j}$, $1 \leq j \leq k^{1/r}$, by performing simultaneously $k^{1/r}$ disjoint parallel-prefix operations on the register values of stage 4 in a column-major direction. In addition, processor $p_{Left+N_m,j}$ broadcasts l_j to processors $p_{Left+i,j}$, $1 \leq i \leq N_m$. This step is concluded by performing simultaneously N_m prefix operations on the the bucket sizes l_j in a row-major direction.
[**Analysis of Stage 5:** The time complexity of this stage is thus bounded above by $2T_{ppf}(p) + T_{brd}(p) \leq 3T_{ppf}(p)$.]

Stage 6 :
Processor $p_{Left+i,1}$ reads value l_t from processor $p_{Left+i,t}$ and $\sum_{j=1}^{t-1} l_j$ from processor $p_{Left+i,t-1}$. It also reads the information computed in the first parallel-prefix operation of stage 5 from processor $p_{Left+i,t}$. Using these values, processor $p_{Left+i,1}$ computes the index of the processor that will receive the contents of register z_{Left+i} in the following iterative call.
[**Analysis of Stage 6:** This stage requires the routing of an $O(1)$ relation and a constant number of local operations. Thus, the time required is $\max\{L, O(1)\} + \max\{L, O(g)\}$.]

Stage 7 :
Each processor routes its key to the appropriate destination.
[**Analysis of Stage 7:** Communication time $\max\{L, O(g)\}$ is required for this routing operation.]

Stage 8 :
for $j = 1$ **to** $k^{1/r}$ **do**
begin
  SEARCH ($Left + \sum_{u=1}^{j-1} l_u$, $Left - 1 + \sum_{u=1}^{j} l_u$, $y_{djk^{-1/r}}$, $djk^{-1/r}$,$I + 1$)
end
[**Analysis of Stage 8:** Procedure SEARCH is called recursively $k^{1/r}$ times, once for each of the $k^{1/r}$ newly formed buckets induced by the sequence $\langle s_1, \ldots, s_{k^{1/r}} \rangle$. After r iterations not only is the bucket index of each input key fully determined but also each key has been routed to its destination processor.]

end
Analysis of the algorithm. The total running time of the algorithm over all r iterations derived from the analysis of each individual stage is at most $5rT_{ppf}(p) + O(r \max\{L, O(1)\}) + O(r \max\{L, O(g)\})$.

2.1.2 The Randomized BSP Sorting Algorithm

The randomized BSP sorting algorithm that uses procedure Search is presented in detail below. The algorithm sorts N keys with p processors, where $p = N^{1+\epsilon}$, for any positive constant $0 < \epsilon < 1$, with high probability. Since the number of processors is greater than the number of keys available, we assume that initially each one of the first N of the p processors holds exactly one of the input keys.

We first present the following probabilistic argument. Let $X = \langle x_1, x_2, \ldots, x_N \rangle$ be an ordered set of keys indexed such that $x_i < x_{i+1}$, for all i, $1 \leq i \leq N-1$, and $Y = \langle y_1, y_2, \ldots, y_{ks-1} \rangle$ be a randomly chosen subset of $ks - 1 \leq N$ keys of X also indexed such that $y_i < y_{i+1}$, for all $1 \leq i \leq ks-2$, for some positive integers k and s. We cite the following claim of [10] related to the size of the k buckets $X_0, \ldots, X_{k-1}$ induced on sequence X by sequence Y.

Claim 1 *Let $k \geq 2$, $s \geq 1$, $ks < N/2$, $n \geq N$, $0 < \varepsilon < 1$, $\rho > 0$, and*

$$s \geq \frac{1+\varepsilon}{\varepsilon^2}(2\rho \log n + \log(2\pi k^2 (ks-1) e^{1/(3(ks-1))})).$$

Then the probability that any one of the X_i, for all i, $0 \leq i \leq k-1$, is of size more than $\lceil (1+\varepsilon)\frac{N-k+1}{k} \rceil$ is at most $n^{-\rho}$.

We note that the parameter n above refers to the size of the selection problem which the sorting algorithm is part of. For the selection algorithm to be described, we assume $n > N \geq \sqrt{n}$. The randomized BSP sorting algorithm is described below.

Procedure Sort (N, p)

begin

Stage 1 :
A sample set $Y = \{y_1, y_2, \ldots, y_{ks-1}\}$ of size $ks - 1 < N/2$ is selected at random without replacement among the N input keys. The oversampling parameter s is set to $s = 4\log^2 n$, and k is chosen so that $\sqrt{N} - 1 \leq ks - 1 \leq \sqrt{p}$, where $n \geq N$.

Let $x = \Theta(1)$. Every x-th processor of the N processors that hold an input key is involved in the following process. Call each such processor "active" and consider it to be associated with a set of $x-1$ "non-active" processors. Among the N/x "active" processors, a possibly smaller subset of size $ks-1$ selects processor identifiers. We call the latter processors "selectors".

(1) Each of $ks-1$ "selector" processors chooses uniformly at random and without replacement one processor identifier that identifies one of the N/x "active" processors involved in this process.

(2) The processor identifiers selected are routed to the so identified processors.

(3) Each processor p_i receives t_i identifiers. If $t_i > x$ a failure of the sample selection process is declared. Otherwise, processor p_i selects uniformly at random and without replacement t_i keys among the its keys and the ones of its $x-1$ associated processors.

(4) A parallel-prefix on the number t_i of keys selected per processor is performed.

(5) The result of the parallel-prefix is used to evenly distribute the selected key sample among a predetermined set of $ks-1$ processors.

[**Analysis of Stage 1:** In step 1, $\max\{L, O(1)\}$ time is required for a single random processor identifier generation. In step 2 we use the generalized bound due to Chernoff [5] $\text{Prob}[S_{n,P} \geq eunP/\log u] \leq (1/e)^{unP}$ for $u \geq e$ and $u^2 nP \geq \log u$, where $S_{n,P}$ is the number of successes in n independent Bernoulli trials with individual probability of success P. By substituting for $n = ks-1$, $P = x/N$ and $u/\log u = N/(e(ks-1))$, we obtain that no processor gets more than x processor identifiers with probability at least $1 - e^{\log N - (x/e)\log(N/(e(ks-1)))}$. By the lower and upper bounds on ks, the fact that $N \geq \sqrt{n}$ and $x = \Theta(1)$ we obtain that the failure probability of this stage is at most $n^{1-\rho}$, for any constant $\rho > 1$. Larger values of x yield lower failure probabilities.

Step 3 requires at most $O(x)$ time for computation and the routing of two $O(x)$-relations. Step 4 takes time $T_{ppf}(p)$. Step 5 requires the routing of an $O(x)$-relation. Thus, the overall complexity of this stage is bounded above by $T_{ppf}(p) + O(\max\{L, O(1)\}) + O(\max\{L, O(g)\})$ with probability at least $1 - n^{1-\rho}$, for any constant $\rho > 1$.]

Stage 2 :
The sample of size $ks - 1$ is sorted.
[**Analysis of Stage 2:** The sorting algorithm used is a BSP adaptation of an algorithm in [26]. The sorting method follows. Consider input $Y = \{y_1, y_2, \ldots, y_{ks-1}\}$ of size $ks - 1$ and $(ks - 1)^2$ processors. We note that, since $ks - 1 \leq \sqrt{p}$, there are at least $(ks - 1)^2$ processors available. We number processors in a mesh-like fashion, so that we can refer to processor at row i and column j, as $p_{i,j}$, $1 \leq i, j \leq ks - 1$. Input elements are assumed to be distributed among the processors in the first row, i.e., element y_j resides in the local memory of processor $p_{1,j}$. By performing $ks - 1$ disjoint broadcast operations, processor $p_{1,j}$, for all $1 \leq j \leq ks - 1$, broadcasts element y_j to all processors of column j. Similarly, processor $p_{i,i}$, for all $1 \leq i \leq ks - 1$, broadcasts element y_i to all processors of row i. In a single superstep, processor $p_{i,j}$, $1 \leq i, j \leq ks - 1$, compares y_i and y_j, and subsequently, by executing simultaneously $ks - 1$ disjoint parallel-prefix operations determines the rank of each y_i in Y. The sorting operation is completed by routing each element to its final destination. Therefore, the overall complexity of this stage is bounded above by $2T_{brd}(p) + T_{ppf}(p) + \max\{L, O(1)\} + \max\{L, O(g)\} \leq 2T_{ppf}(p) + \max\{L, O(1)\} + \max\{L, O(g)\}$.]

Stage 3 :
Every s-th key of the sorted sample is selected for a total of $k - 1$ splitters. The maximum of the N input keys is also determined and appended to the list of the $k - 1$ splitters.
[**Analysis of Stage 3:** Each processor in $\max\{L, O(1)\}$ time decides whether it holds a splitter. The maximum of N keys can be determined in time $T_{ppf}(p)$, for a total of $T_{ppf}(p) + \max\{L, O(1)\}$.]

Stage 4 :
The $k - 1$ splitters are employed to partition the input set into k buckets.
[**Analysis of Stage 4:** Procedure SEARCH is used to determine the destination bucket of each input key. The $k - 1$ splitters split the sample into k buckets, where the maximum key of the input set is used to identify the right boundary of the k-th bucket. As a result of the operations performed in stage 6 of SEARCH, the input keys destined to the same bucket are routed to the consecutively indexed processors allocated to that bucket. The time required for this stage is that of procedure SEARCH.]

Stage 5 :
The keys falling into each bucket are sorted.
[**Analysis of Stage 5:** By Claim 1, the number of keys in each bucket is at most $\lceil (1 + \delta)(N - k + 1)/k \rceil$, where $\delta = O(\sqrt{\log n})$, with probability at least $1 - n^{-\rho}$, for any constant $\rho > 0$. We assign to each such bucket an equal number of processors p/k. If the size of a bucket is at most $\sqrt{p/k}$, then its keys are sorted in time $2T_{ppf}(p) + \max\{L, O(1)\}$ by using the sorting algorithm implied in stage 2. Otherwise, SORT is called recursively.]

end
Analysis of the algorithm. Let $s = 4\log^2 n$, $ks = N^t$ and $p = N^{1+\epsilon}$, for any constants t and ϵ such that $1/2 \leq t < (1 + \epsilon)/2$ and $0 < \epsilon < 1$.

We examine the running time $S(N, p)$ of the algorithm. Stages 1 through 3 require $O(\max\{L, O(1)\}) + O(\max\{L, O(g)\}) + 4T_{ppf}(p)$ time. Stage 4 requires $5rT_{ppf}(p) + O(r \max\{L, O(1)\}) + O(r \max\{L, O(g)\})$ time, for any constant $r > 1/\epsilon$. By claim 1 and the choice of s, parameter δ in Stage 5 is $O(1/\sqrt{\log n})$. Therefore, the execution time of stage 5 is either that of stage 3, if the algorithm of stage 2 is applicable, which is $2T_{ppf}(p) + \max\{L, O(1)\} + \max\{L, O(g)\}$, or $S((1 + O(1/\sqrt{\log n}))N/k + 2, p/k)$, otherwise. The overall time complexity $S(N, p)$ of the algorithm can thus be summarized as follows.

$$S(N,p) = r\ O(T_{ppf}(p) + \max\{L, O(1)\} + \max\{L, O(g)\}) + S((1 + O(\frac{1}{\sqrt{\log n}}))N/k, p/k).$$

We unfold the recurrence relation l times so that in the i-th recursive call, $i \leq l$, we choose $k_i = (1+\delta)^{tR} N^{t(1-t)^i} s^{tR-1}$, where $R = \sum_{j=0}^{i-1} (1-t)^j$ and r_i is the value of r in the i-th recursive call. We then get that the time complexity $S(n,p)$ of the algorithm is bounded above by the following expression.

$$\sum_i r_i \, O(T_{ppf}(p) + \max\{L, O(g)\} + \max\{L, O(1)\}) + S((1+O(\frac{1}{\sqrt{\log n}}))^{1+R} N^{(1-t)^l} s^{1/t}, N^{\epsilon+(1-t)^l} s) \quad \leq$$

$$\sum_i r_i \, O(T_{ppf}(p) + \max\{L, O(g)\} + \max\{L, O(1)\}) + S(O(N^{(1-t)^l} s^{1/t}), O(N^{\epsilon+(1-t)^l} s)).$$

Let $k_i s = N_i^t$, $N_{i+1} \leq (1+\delta) N_i / k_i$ and $p_{i+1} = p_i / k_i$, where $N_0 = N$ and $p_0 = p$. The sorting operation in stage 2 requires that $k_i s \leq \sqrt{p_i}$. Stage 4 applies provided that $p > N_i k_i^{1/r_i}$, where r_i is as defined in procedure SEARCH. The sorting algorithm used in stage 5 requires that $N_{i+1} \leq (1+\delta) N_i / k_i \leq \sqrt{p_i / k_i}$. The following conditions must then hold.

(1) $t < \frac{\epsilon}{2(1-t)^i} + \frac{1}{2}$, for all i, $0 \leq i \leq l$,

(2) $\epsilon > \frac{t(1-t)^i}{r_i}$, for all i, $0 \leq i \leq l$, and

(3) $(1-t)^l < \epsilon$.

By way of inequality 2 we choose $r_i = 1 + t(1-t)^i/\epsilon$ we get that $\sum_{i \leq l} r_i = O(l + 1/\epsilon)$. By way of inequality 3 the depth of the recursion l is the smallest l such that $l > \lg(1/\epsilon)/\lg(1/(1-t))$. For such an l, the algorithm of stage 2 is applicable to stage 5 and thus

$$S(O(N^{(1-t)^l} s^{1/t}), O(N^{\epsilon+(1-t)^l} s)) = 2T_{ppf}(p)) + \max\{L, O(1)\} + \max\{L, O(g)\}.$$

We then get the following expression for $S(N,p)$, where $l = 1 + \lfloor \lg(1/\epsilon)/\lg(1/(1-t)) \rfloor$.

$$S(N,p) = O((1/\epsilon + l) T_{ppf}(p)) + O((1/\epsilon + l) \max\{L, O(1)\}) + O((1/\epsilon + l) \max\{L, O(g)\}).$$

The following theorem is then proved.

Theorem 1 *There exists a randomized BSP sorting algorithm that with probability $1 - O(n^{1-\rho})$, for any constant $\rho > 0$ and $n > N \geq \sqrt{n}$, sorts N keys with $p = N^{1+\epsilon}$ processors, for any constant $0 < \epsilon < 1$, in time*

$$S(N,p) = O\left((1/\epsilon + l)(T_{ppf}(p) + \max\{L, O(1)\} + \max\{L, O(g)\})\right),$$

where $l = \lceil \lg(1/\epsilon)/\lg(1/(1-t)) \rceil$, for any constant t such that $1/2 \leq t < 1/2 + \epsilon/2$.

3 Random Sampling

We present results related to random sampling that generalize those presented in [11] for the special case of median finding. In the discussion and proofs to follow we use the notation originally used in [10]. Let $X = \langle x_1, \ldots, x_N \rangle$ be a set of input keys ordered such that $x_i < x_{i+1}$, for $1 \leq i \leq N-1$. We are assuming that the keys are distinct because by appending to each key its index we can make them so. We are interested in finding the C-th smallest of the keys in set X. We assume, without loss of generality, that $C \leq \lceil N/2 \rceil$. Otherwise, we use symmetric arguments to find the $(N-C)$-th largest key. Let then be $c = C/N$. For integers $\lambda \geq 1$ and $s \geq 1$, let $Y = \langle y_1, \ldots, y_{2\lambda s - 1} \rangle$ be a randomly chosen subset of $2\lambda s - 1 \leq N/2$ keys of X, also ordered so that $y_i < y_{i+1}$. The following three subsets are then formed for some positive integers μ_0, μ_2, and $t \geq 1$ such that $2c\lambda s > \mu_0 t/2$.

$$\begin{aligned} X_0 &= \{x \mid x \in X \text{ and } x < y_{2c\lambda s - \mu_0 t/2}\}, \\ X_1 &= \{x \mid x \in X \text{ and } y_{2c\lambda s - \mu_0 t/2} < x < y_{2c\lambda s + \mu_2 t/2}\}, \\ X_2 &= \{x \mid x \in X \text{ and } y_{2c\lambda s + \mu_2 t/2} < x\}. \end{aligned}$$

For the binomial coefficients, we assume that $\binom{i}{j} = 0$ if $j > i$. For $0 \le i \le 2$, let $n_i = |X_i|$ and let $p_i(j)$ be the probability that $n_i = j$. The following is shown in [10].

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

$$p_i(j) = \frac{\binom{j}{2c\lambda s - \mu_i t/2 - 1}\binom{N-j-1}{2(1-c)\lambda s + \mu_i t/2 - 1}}{\binom{N}{2\lambda s - 1}}.$$

The randomized BSP selection algorithms we describe in the following sections partition the input key set into three sets as above. By establishing lower and upper bounds on the sizes of X_0 and X_2 we guarantee that the required statistic is located in set X_1. Our goal is thus to determine lower bounds on the size of set X_0, and symmetrically of X_2, by proving that the probability of having such a set of size less than $cN - f(N)$, for some function $f(N)$ of N, is negligible. We investigate bounds on $f(N)$ for this to hold. As the two cases for X_0 and X_2 are identically handled, we show the result by substituting $\mu_0 = \mu_2 = \mu$. Similarly, upper bounds on the size of set X_0, and symmetrically of X_2, can be established.

Let $B = cN - f(N) - 1$. Particularly, we would like to show that the term $\sum_{j \le B} p_i(j)$, $i = 0, 2$, is sufficiently small for appropriate choices of $f(N)$. We deduce the following relationships used later for $n \ge N$. We note that the special case $c = 1/2$ was shown in [11].

Proposition 1 *Let $C \le \lceil N/2 \rceil$ and $c = C/N$. Let $\lambda, \mu \ge 1$, $s, t \ge 1$, $2\lambda s - 1 < N/2$, $n \ge N$, $2c\lambda s > \mu t/2$, $\mu t \ge 3$, and $\rho > 0$. If $f(N) = (N\ \mu t)/(2\ \lambda s)$ and $\mu t = 2\sqrt{3\rho\ 2c\ \lambda s\ \log n}$, then the probability that set X_0 is of size more than $cN - f(N) - 1$ is at least $1 - n^{1-\rho}$.*

Proof We first observe that the combinatorial identity [20],

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

implies that

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

We use a combinatorial interpretation argument, as in [10], to argue that the sum $\sum_{j \le B} p(j)$ for $j \le B$ is the number of ways we can choose $2\lambda s - 1$ objects so that among them, the object with the $(2c\lambda s - \mu t/2)$-th highest label is labeled at most B.

In order to establish an upper bound for $\sum_{j \le B} p(j)$ 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 $2c\lambda s - \mu t/2 - 1$ successes among the first j, a success at the $(j+1)$-st trial, and $2(1-c)\lambda s + \mu t/2 - 1$ successes in the remainder is

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

Let A be the ratio of $r(j)$ over $p(j)$. The following then holds.

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

The following inequality is shown in [5].

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

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

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

It is then immediate that the following holds.

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

Thus, we obtain the following result.

$$\sum_{j\le B} p(j) \quad \le \quad \frac{q}{A}\sum_{j\le B}\binom{j}{2c\lambda s-\mu t/2-1} q^{2c\lambda s-\mu t/2-1}(1-q)^{j-2c\lambda s+\mu t/2+1}. \tag{2}$$

Our goal is to establish an upper bound for the term below.

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

We employ generating functions. The term

$$\binom{j}{2c\lambda s-\mu t/2-1} q^{2c\lambda s-\mu t/2-1}(1-q)^{j-2c\lambda s+\mu t/2+1}$$

is the coefficient of $x^{2c\lambda s-\mu t/2-1}$ in the polynomial $((1-q)+qx)^j$. Therefore, the sum over all those terms with $j \le B$ is the coefficient of $x^{2c\lambda s-\mu t/2-1}$ in the following sum.

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

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

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

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

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

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

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

We thus get for the sum of interest.

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

The desired bound follows from equations (1), (2) and (3) and is given by the following expression.

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

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

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

Substituting for $\mu t = 2\sqrt{3\rho\, 2c\, \lambda s\, \log n}$ and noting that $\sqrt{2\pi(2\lambda s-1)}\, e^{\frac{1}{6(2\lambda s-1)}} < n$, gives the desired result. ■

We can similarly establish upper bounds for the sizes of the sets X_0 and X_2. The corresponding formula is a generalization of the result shown in [11]. For the size of set X_0, and by symmetry of set X_2, we get that for $B = cN - f(N)$,

$$\sum_{j\ge B} p(j) \le \sqrt{2\pi(2\lambda s-1)}\, e^{\frac{1}{6(2\lambda s-1)}}\, e^{-\frac{(Bq-2c\lambda s+\mu t/2+1)^2}{2Bq}}.$$

Proposition 2 *Let $C \le \lceil N/2\rceil$ and $c = C/N$. Let $\lambda,\mu \ge 1$, $s,t \ge 1$, $2\lambda s-1 < N/2$, $n \ge N$, $2c\lambda s > \mu t/2$, and $\rho > 0$. If $\mu t \ge 2\sqrt{2\rho\, 2c\, \lambda s\, \log n}$ then the probability that set X_0 is of size at most $cN-1$ is at least $1-n^{1-\rho}$.*

Proof The proof follows that of proposition 1 with $f(N) = 0$. ■

The following result is also required for the analysis of a selection algorithm with improved lower order terms.

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

Proof The proof follows that of proposition 1. ■

The following bounds on the size of X_0 and X_2 can then be derived for $s = t = 1$.

Claim 3 *Let $C \le \lceil N/2\rceil$ and $c = C/N$. Let $\lambda \ge 1$, $2\lambda-1 < N/2$, $n \ge N$, and $\rho > 1$. Let $\mu_0 \ge 2\sqrt{2\rho\, 2c\, \lambda\, \log n} \ge 1$ and $2c\,\lambda > \mu_0/2$. Then, with probability at least $1-n^{1-\rho}$, set X_0 is of size at most $C-1$.*

Proof The proof follows directly from proposition 2 with $s = t = 1$. We only note that the failure probability bound $n^{1-\rho}$ is an overestimate of a tighter one $n^{\eta-\rho}$, where η is any constant greater than 0.5 and smaller than 1.0. ■

Claim 4 *Let $C \le \lceil N/2\rceil$ and $c = C/N$. Let $\lambda \ge 1$, $2\lambda-1 < N/2$, $n \ge N$, and $\rho > 1$. Let $\mu_2 \ge 2\sqrt{2\rho\, 2(1-c)\, \lambda\, \log n} \ge 1$ and $2(1-c)\,\lambda > \mu_2/2$. Then, with probability at least $1-n^{1-\rho}$, set X_2 is of size at most $N-C-1$.*

Proof The proof follows directly from the previous claim substituting c and C, with $(1-c)$ and $N-C$ respectively. ■

Claim 5 *Let $C \leq \lceil N/2 \rceil$ and $c = C/N$. Let $\lambda \geq 1$, $2\lambda - 1 < N/2$, $n \geq N$, and $\rho > 1$. Let $2c\lambda > \mu_0/2$ and $\mu_0 = 2\sqrt{3\rho\, 2c\, \lambda\, \log n} \geq 3$. Then, with probability at least $1 - n^{1-\rho}$, set X_0 is of size at least $cN - N\sqrt{3\rho\, 2c\, \log n}/\sqrt{\lambda}$.*

Proof The claim is a direct consequence of proposition 1 by substituting for $s = t = 1$ and $\mu = 2\sqrt{3\rho\, 2c\, \lambda\, \log n}$ for $f(N)$. ■

Claim 6 *Let $C \leq \lceil N/2 \rceil$ and $c = C/N$. Let $\lambda \geq 1$, $2\lambda - 1 < N/2$, $n \geq N$, and $\rho > 1$. Let $2(1-c)\lambda > \mu_2/2$ and $\mu_2 = 2\sqrt{3\rho\, 2(1-c)\, \lambda\, \log n} \geq 3$. Then, with probability at least $1 - n^{1-\rho}$, set X_2 is of size at least $N - cN - N\sqrt{3\rho\, 2(1-c)\, \log n}/\sqrt{\lambda}$.*

Proof The proof follows directly from the previous claim substituting c and C, with $(1-c)$ and $N-C$ respectively. ■

Claim 7 *Let $C \leq \lceil N/2 \rceil$ and $c = C/N$. Let $y_i \geq 36$, $y_i < y_{i+1}/2$, $n \geq N$, $y_{i+1} = dy_i$, $d > 36$, and $\rho > 1$. Let $\mu_0 = 2\sqrt{3\rho\, c\, (y_i+1)\, \log n} \geq 1$ and $c\,(y_i+1) > \mu_0/2$. Then, with probability at least $1 - n^{1-\rho}$, set X_0 is of size at most $c(y_{i+1}+1) - \sqrt{3\rho c(y_{i+1}+1)\log n} - 1$.*

Proof The proof follows directly from proposition 3 with $s = t = 1$, $2\lambda - 1 = y_i$, $n = y_{i+1}$, and $f(n) = \sqrt{3\rho(y_{i+1}+1)\log n/c} - 1$ there. The restrictions $d > 36$ and $y_i > 36$ are required for inequality $\sqrt{(y_{i+1}+1)(y_i+1)/y_{i+1}^2} < 1/6$ to hold. ■

A claim symmetric to claim 7, and similar to claim 4, can also be established.

4 Randomized Selection

The randomized selection algorithm of this section derives from ideas in [9] and extends the results presented in [11]. The general formulation of the algorithm is as follows. A random sample of the input is initially chosen, and two keys within this sample are also selected so that the C-th smallest key is between them. The iterative version of the algorithm, which improves the lower order terms of the non-iterative version, proceeds so that in the following step a new sample is selected that is a superset of the sample selected in the previous iteration. Depending on the size of the sample in the first iteration and the number of iterations, the sequential version requires time $n + C + O(n^{1/2+\beta})$ comparisons, where $\beta > 0$ depends on these choices. The number of iterations determines a lower bound on the value of β. A BSP algorithm is derived from this selection method that uses essentially parallel-prefix, broadcast, and regular-routing operations for sample sorting.

In order to simplify the exposition of the BSP selection algorithm we first present its sequential non-iterative counterpart.

Theorem 2 *For any constant $\rho > 1$, there exists an algorithm that, with probability at least $1 - O(n^{1-\rho})$, determines the C-th smallest, $C \leq \lceil n/2 \rceil$, of n keys within $n + C + o(n)$ comparisons.*

Remark. We note that if $C > \lceil n/2 \rceil$, by reversing the meaning of the inequalities in procedure Select, we can employ it to determine the $(n-C)$-th largest key of the input key set, which is the C-th smallest key of the set.

Proof Procedure Select that determines the C-th smallest key of a set X of size n is given in pseudocode in figure 1. A high-level description of the algorithm is also given below.

Procedure Select (X, C)

begin

begin SELECT (X, C)

1. **let** $c = C/n$, $\lambda = n^{2/3}(3\rho \log n)^{1/3}((\sqrt{2c} + \sqrt{2(1-c)})/2)^{2/3}$, such that $2\lambda - 1 < n/2$;
2. **let** $\mu_0 = 2\sqrt{3\rho\, 2c\, \lambda \log n}$, $\mu_2 = 2\sqrt{3\rho\, 2(1-c)\, \lambda \log n}$;
3. **select** randomly a sample $S = \{y_1, y_2, \ldots, y_{2\lambda-1}\}$ of size $2\lambda - 1$;
4. denote by $\langle y_1^*, y_2^*, \ldots, y_{2\lambda-1}^* \rangle$ the sequence of keys in S in sorted order ;
5. **if** $(c \geq 2\rho \log n/\lambda)$ **then**
6.   **let** $m = y^*_{2c\lambda-\mu_0/2}$, $M = y^*_{2c\lambda+\mu_2/2}$;
7. **else**
8.   **let** $m = -\infty$, $M = y^*_{2c\lambda+\mu_2/2}$;
9. **let** $r = l = 0$, $S = R_r = R_l = \{\}$;
10. **for** $i = 1$ **to** n
11. **do if** $(x_i > M)$ **then**
12.   **let** $r = r + 1$, $R_r = R_r \bigcup \{x_i\}$;
13.   **elseif** $(x_i < m)$ **then**
14.   **let** $l = l + 1$, $R_l = R_l \bigcup \{x_i\}$;
15.   **else**
16.   **let** $S = S \bigcup \{x_i\}$;
17. **if** $(l < C$ and $r < n - C)$ **then**
18.   **return** SELECT_DET$(S, C - l)$;
19. **else**
20.   **reexecute** procedure SELECT ;

end SELECT

Figure 1: Procedure SELECT, n - problem size, SELECT_SEQ* - a deterministic selection algorithm.

Stage 1 : [Lines 1-8 of SELECT in figure 1]
Let $c = C/n$ and $\lambda = n^{2/3}(3\rho \log n)^{1/3}((\sqrt{2c} + \sqrt{2(1-c)})/2)^{2/3}$, so that $2\lambda - 1 < n/2$. Variables μ_0 and μ_2 are set to $\mu_0 = 2\sqrt{3\rho\ 2c\ \lambda\ \log n}$ and $\mu_2 = 2\sqrt{3\rho\ 2(1-c)\ \lambda\ \log n}$ respectively. A random sample of size $2\lambda - 1$ is selected.

If $c \geq 2\rho \log n/\lambda$, the $(2c\lambda - \mu_0/2)$-th and $(2c\lambda + \mu_2/2)$-th keys of the sample are determined using the linear time selection algorithm in [4]. Otherwise, the $(2c\lambda + \mu_2/2)$-th key of the sample is only determined using the algorithm in [4]. Let $m = y_{2c\lambda-\mu_0/2}$ (resp. $m = -\infty$) and $M = y_{2c\lambda+\mu_2/2}$ be the so identified keys for the two cases respectively.

Stage 2 : [Lines 9-16 of SELECT in figure 1]
Let $l = r = 0$ and $S = \{\}$. For each key x in X, x is compared to M first. If x is larger than M, then $r = r + 1$. If x is smaller than M, then x is compared to m. If it is found smaller than m, then $l = l + 1$. Otherwise, $S = S \cup \{x\}$.

Stage 3 : [Lines 17-20 of SELECT in figure 1]
If $l < C$ and $r < n - C$, the $(C - l)$-th smallest key of set S is determined using the linear-time selection algorithm in [4]. Otherwise failure of the algorithm is declared and the algorithm is reexecuted.
end

Analysis of the algorithm.
Case 1: $c \geq 3\rho \log n/\lambda$.

Stage 1 of the algorithm, by the choice of λ requires $O(\lambda)$ comparisons. By claims 3, 4, 5 and 6, for $N = n$, at least $C - n\sqrt{3\rho\ 2c\ \log n}/\sqrt{\lambda}$ and at most $C - 1$ keys have value less than m, and at least $n - C - n\sqrt{3\rho\ 2(1-c)\ \log n}/\sqrt{\lambda}$ and at most $n - C - 1$ keys have values larger than M, with probability at least $1 - n^{1-\rho}$, for any constant $\rho > 1$. With the stated probability the number of comparisons performed at stage 2 is at most $(n - C) + 2(C + n\sqrt{3\rho\ 2(1-c)\ \log n}/\sqrt{\lambda})$ which is $n + C + 2n\sqrt{3\rho\ 2(1-c)\ \log n}/\sqrt{\lambda}$.

The size of set S is also bounded above by $(n\sqrt{3\rho\ \log n}/\sqrt{\lambda})(\sqrt{2c} + \sqrt{2(1-c)})$. This latter term is 2λ. Stage 3 then, by way of the algorithm in [4] and the value of λ requires $O(n^{2/3}(6\rho\ \lg n)^{1/3}(\sqrt{2c} +$

$\sqrt{2(1-c)})^{2/3})$ comparisons. Altogether, an upper bound of $n+C+O(n^{2/3}(6\rho\lg n)^{1/3}(\sqrt{2c}+\sqrt{2(1-c)})^{2/3})$ comparisons is proved with probability at least $1-n^{1-\rho}$, for any constant $\rho>1$.
Case 2: $c < 3\rho\log n/\lambda$.

Stages 1 and 2 require the same number of comparisons as for the previous case. In stage 3 the number of keys with value larger than M is at least $n-C-n\sqrt{3\rho\, 2(1-c)\,\log n}/\sqrt{\lambda}$ and therefore, $|S|$ is at most $C+n\sqrt{3\rho\, 2(1-c)\,\log n}/\sqrt{\lambda}$ with probability at least $1-n^{1-\rho}$, for any constant $\rho>1$. Since $c<3\rho\log n/\lambda$, with the stated probability, the size of S is bounded above by $3\rho n\log n/\lambda+n\sqrt{3\rho\, 2(1-c)\,\log n}/\sqrt{\lambda}$. The first term is thus insignificant and the same upper bound as for the previous case can be claimed for the total comparison count. ■

4.1 A BSP Selection Algorithm

In this section, we show how algorithm Select presented previously can be modified to give a BSP algorithm for the selection problem. The procedure to manage the selection process is summarized in the following proposition.

Proposition 4 *For any constant $\rho>1$, there exists a randomized BSP selection algorithm that, with probability $1-O(n^{1-\rho})$, determines the C-th smallest, $C\le\lceil n/2\rceil$, of n keys that for any integers n and $p\le n$, requires time*

$$T_{sel}(n,C,p)=\begin{cases}\frac{n+C}{p}+O\left(\sqrt{\frac{C\log n}{p}}+\frac{n^{\frac{2}{3}+\delta}\lg^{\frac{4}{3}}n}{p}+g\frac{n^{\frac{2}{3}+\delta}\lg^{\frac{1}{3}}n}{p}+L\right)+O(T_{ppf}(p)) & \text{if } p\le n^{\frac{2}{3}+\zeta},\\ \frac{n+C}{p}+O\left(\sqrt{\frac{C\log n}{p}}+\log n+g+L\right)+O(T_{ppf}(p)) & \text{otherwise,}\end{cases}$$

for any arbitrarily small constants ζ and δ, such that $0<\zeta<\delta<1/3$.

Proof We assume that the keys are randomly and evenly distributed among the processors. If the keys are not initially randomly distributed, then a prerandomization phase is required for optimal time bounds to be derived. For a random distribution and given $p=o(n/\lg n)$, the BSP algorithm requires, with high probability, $(n+C)/p+o(n/p)$ comparisons and $o(gn/p)$ communication. For a non-random distribution, the BSP algorithm requires, with high probability, $2n/p+o(n/p)$ comparisons; the communication time, however, might be increased to $O(gn/p)$. In the latter case, the algorithm is work-optimal in computation in the sense that the total comparison count over all processors is still $n+C+o(n)$. The process to manage the selection operation is outlined in procedure Select_BSP(X, C) (figure 2), where X denotes the input set and C the required statistic.

Procedure Select_BSP (X,C)

begin
Stage 1 : [Lines 1-3 of Select_BSP in figure 2]
A sample of size $2\lambda-1=2n^{2/3}(3\rho\log n)^{1/3}((\sqrt{2c}+\sqrt{2(1-c)})/2)^{2/3}-1$ is selected, where $c=C/n$. The sample selection method is as follows.

(1) Each processor selects approximately the same number of processor identifiers ($\lceil(2\lambda-1)/p\rceil$ or $\lfloor(2\lambda-1)/p\rfloor$) uniformly at random compensating for the fact that one processor may have one more key ($\lceil n/p\rceil$) than another one ($\lfloor n/p\rfloor$). A sample of size $2\lambda-1$, over all processors, is thus selected.

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

(3) Each processor p_i that receives t_i identifiers selects among its keys uniformly at random without replacement t_i keys.

(4) A parallel-prefix on the number t_i of keys selected per processor is performed.

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

begin SELECT_BSP (X, C)
1. **let** $c = C/n$, $\lambda = n^{2/3}(3\rho \log n)^{1/3}((\sqrt{2c} + \sqrt{2(1-c)})/2)^{2/3}$,
  such that $2\lambda - 1 < n/2$ and $\rho > 1$;
2. **let** $\mu_0 = 2\sqrt{3\rho\ 2c\ \lambda \log n}$, $\mu_2 = 2\sqrt{3\rho\ 2(1-c)\ \lambda \log n}$;
3. **select** randomly a sample $S = \{y_1, y_2, \ldots, y_{2\lambda-1}\}$ of size $2\lambda - 1$;
4. SORT(S) ;
5. denote by $\langle y_1^*, y_2^*, \ldots, y_{2\lambda-1}^* \rangle$ the sequence of keys in S in sorted order ;
6. **if** $(c \geq 2\rho \log n/\lambda)$ **then**
7.   **let** $m = y^*_{2c\lambda-\mu_0/2}$, $M = y^*_{2c\lambda+\mu_2/2}$;
8. **else**
9.   **let** $m = -\infty$, $M = y^*_{2c\lambda+\mu_2/2}$;
10. BROADCAST($\{m, M\}$) ;
11. **for** each processor $p^{\langle k \rangle}$, $k \in \{0, 1, \ldots, p-1\}$, in parallel
12.   **do let** $r^{\langle k \rangle} = l^{\langle k \rangle} = 0$, $R^{\langle k \rangle} = R_r^{\langle k \rangle} = R_l^{\langle k \rangle} = \{\}$;
13.   **for** $i = 1$ **to** $|X^{\langle k \rangle}|$
14.   **do if** $(X^{\langle k \rangle}[i] > M)$ **then**
15.   **let** $r^{\langle k \rangle} = r^{\langle k \rangle} + 1$, $R_r^{\langle k \rangle} = R_r^{\langle k \rangle} \bigcup \{X^{\langle k \rangle}[i]\}$;
16.   **elseif** $(X^{\langle k \rangle}[i] < m)$ **then**
17.   **let** $l^{\langle k \rangle} = l^{\langle k \rangle} + 1$, $R_l^{\langle k \rangle} = R_l^{\langle k \rangle} \bigcup \{X^{\langle k \rangle}[i]\}$;
18.   **else**
19.   **let** $R^{\langle k \rangle} = R^{\langle k \rangle} \bigcup \{X^{\langle k \rangle}[i]\}$;
20. PARALLEL_PREFIX_SUM($\{r^{\langle k \rangle}, l^{\langle k \rangle}\}$) ;
21. BROADCAST($\{\sum_k r^{\langle k \rangle}, \sum_k l^{\langle k \rangle}\}$) ;
22. **if** $(\sum_k l^{\langle k \rangle} < C$ and $\sum_k r^{\langle k \rangle} < n - C)$ **then**
23.   SORT($\bigcup_k R^{\langle k \rangle}$) ;
24.   denote by R^* the sequence of keys in $\bigcup_k R^{\langle k \rangle}$ in sorted order ;
25.   BROADCAST($R^*[C - \sum_k l^{\langle k \rangle}]$) ;
26.   **return** $R^*[C - \sum_k l^{\langle k \rangle}]$;
27. **else**
28.   **reexecute** procedure SELECT_BSP ;

end SELECT_BSP

Figure 2: Procedure SELECT_BSP, n - problem size, p - number of processors.

[**Analysis of Stage 1:** In step 1, at most $\max\{L, \lceil 2\lambda/p \rceil\}$ time is spent on each processor for random processor identifier generation. We distinguish four cases in the analysis of step 2.
Case 1: If $\lceil 2\lambda/p \rceil \geq 3\rho \log n$ and since processor identifiers are selected uniformly at random, then the the probability bound in [1], $\mathrm{Prob}[S_{n,P} \geq (1+\varepsilon)\, nP] < e^{-(1/3)\varepsilon^2 nP}$, $0 < \varepsilon < 1$, where $S_{n,P}$ is the number of successes in n independent Bernoulli trials with individual probability of success P, for $n = 2\lambda - 1$, $P = 1/p$ and $\varepsilon = 1 - o(1)$, shows that with probability $1 - O(n^{1-\rho})$, for any $\rho > 1$, no processor gets more than $2\lceil 2\lambda/p \rceil$ processor identifiers.
Case 2: If $1 \leq \lceil 2\lambda/p \rceil < 3\rho \log n$ and since processor identifiers are selected uniformly at random, then the probability bound in [5], $\mathrm{Prob}[S_{m,P} \geq umP] \leq (e/u)^{umP}$, for $mP \geq 1$ and $u(1-P) > 2$, where $S_{m,P}$ is the number of successes in m independent Bernoulli trials with individual probability of success P, for $m = 2\lambda - 1$, $P = 1/p$ and $u = 6e(p/m)\rho \log n$, implies that each processor receives no more than $6e\rho \log n$ processor identifiers with probability $1 - O(n^{-6\rho})$.
Case 3: If $n^{-\beta} \leq \lceil 2\lambda/p \rceil < 1$, for any $0 < \beta < 1/4$, and since processor identifiers are selected uniformly at random, then by way of probability bound in [5], $\mathrm{Prob}[S_{m,P} \geq eumP/\log u] \leq e^{-umP}$, for $u \geq e$ and $u^2 mP \geq \log u$, where $S_{m,P}$ is the number of successes in m independent Bernoulli trials with individual probability of success P, $m = 2\lambda - 1$, $P = 1/p$ and $u/\log u = 6(p/m)\rho \log n$, each processor gets no more than $6e\rho \log n$ processor identifiers with probability $1 - O(n^{-6\rho})$.

Case 4: If $\lceil 2\lambda/p \rceil < n^{-\zeta}$, for any constant $0 < \zeta < 1/3$, and since processor identifiers are selected uniformly at random, then the probability bound of case 3 is employed to show that each processor gets no more that $O(\rho)$ processor identifiers with probability $1 - O(n^{-6\rho})$.

Remark. Cases 3 and 4 may overlap for certain ranges of $\lceil 2\lambda/p \rceil$. Case 4 is applied to such cases.

The time complexity of this step is thus bounded above by $\max\{L, g\max\{2\lceil 2\lambda/p\rceil, 6e\rho\log n\}\}$ if $p \leq n^{2/3+\zeta}$, for any constant $0 < \zeta < 1/3$, and $\max\{L, O(g)\}$ otherwise. Step 3 takes time bounded above by $\max\{L, \max\{2\lceil 2\lambda/p\rceil, 6e\rho\log n\}\}$, if $p \leq n^{2/3+\zeta}$, for any constant $0 < \zeta < 1/3$, and $\max\{L, O(1)\}$ otherwise. Step 4 requires time $T_{ppf}(p)$ for the parallel-prefix operation. Finally, the time complexity of step 5 is at most that of step 2. If constant slack ($n/p = \mathrm{O}(1)$) is available only, then the sample selection process is modified to follow the one outlined in procedure SORT of section 2.1.2.]

Stage 2 : [Lines 4-5 of SELECT_BSP in figure 2]
The sample is sorted by employing the BSP sorting algorithm of [10] or procedure SORT depending on the number of available processors. Let the result of the sorting operation be sequence $Y = \langle y_1, \ldots, y_{2\lambda-1}\rangle$.
[**Analysis of Stage 2:** We distinguish three cases for the analysis of this stage.
Case 1: If $p = \lambda^{1-\epsilon}$ for any constant $0 < \epsilon < 1$, then the algorithm implied in the proof of theorem 5 of [10] is used. The time required for this case is therefore bounded above by $O(T_{ppf}(p)) + O(\lambda\lg\lambda/p) + O(g\lambda/p) + O(L)$.
Case 2: If $p = \lambda^{1+\epsilon}$, for any constant $1/3 < \epsilon \leq 1/2$, then procedure SORT is used. By invoking theorem 1, for $N = 2\lambda$, we get that the time complexity of this stage is $O(T_{ppf}(p)) + O(\max\{L, O(g)\})$.
Case 3: Otherwise, the algorithm implied in the proof of theorem 5 of [10] is employed to sort $2\lambda - 1$ keys with p/n^{δ} processors, for any arbitrarily small constant $0 < \delta < 1/3$. The time required for this stage is bounded above by $O(T_{ppf}(p)) + O(n^{\delta}\lambda\lg\lambda/p) + O(gn^{\delta}\lambda/p) + O(L)$.]

Stage 3 : [Lines 6-10 of SELECT_BSP in figure 2]
Let $\mu_0 = 2\sqrt{3\rho\, 2c\, \lambda\, \log n}$ and $\mu_2 = 2\sqrt{3\rho\, 2(1-c)\, \lambda\, \log n}$. If $c > 2\rho\log n/\lambda$ let $m = y_{2c\lambda-\mu_0/2}$ and $M = y_{2c\lambda+\mu_2/2}$. Otherwise, let $m = -\infty$ and M as previously. Pivot keys m and M are broadcast to all processors.
[**Analysis of Stage 3:** Time $T^2_{brd}(p) \leq T_{ppf}(p)$ suffices for the completion of this stage.]

Stage 4 : [Lines 11-19 of SELECT_BSP in figure 2]
The keys stored in each processor p_i, $0 \leq i < p$, are compared with pivot keys m and M using the comparison sequence outlined in stage 2 of algorithm SELECT of section 4. Local copies of variables l_i and r_i, which count the number of keys local to processor p_i that are less than m and greater than M, are also maintained.
[**Analysis of Stage 4:** This stage is implemented in one superstep. We estimate the number of comparisons required by taking into consideration the the analysis of procedure SELECT related to the number of keys less than m and greater than M. By way of cases 1 and 2 of algorithm SELECT of section 4 we obtain that the the number of input keys between m and M is at most 2λ with probability $1 - O(n^{1-\rho})$.

The number of keys, over all processors, which are smaller than M is by the analysis of procedure SELECT at most $(C + n\sqrt{3\rho\, 2(1-c)\, \log n}/\sqrt{\lambda})$ with probability $1 - O(n^{1-\rho})$. Since these keys are uniformly at random distributed among the processors, we distinguish cases identical to those of stage 1 to decide how many of these keys reside in each processor. As each processor has at least $\lfloor n/p \rfloor$ keys altogether, we can prove that with probability $1 - O(n^{1-\rho})$, each processor has in its local memory at most $(1 + \sqrt{(3\rho\log n)p/C})C/p + O(\max\{2\lceil 2\lambda/p\rceil, 6e\rho\log n\})$ such keys if $C/p > 3\rho\log n$ and $C/p + O(\log n)$ keys otherwise. Therefore, the overall time complexity (comparison count) of this stage is bounded above by $\max\{L, (n+C)/p + \sqrt{(3\rho\log n)C/p} + O(\max\{2\lceil 2\lambda/p\rceil, 6e\rho\log n\})\}$ if $C/p > 3\rho\log n$ and $\max\{L, (n+C)/p + O(\log n)\}$ otherwise.]

Stage 5 : [Line 20 of SELECT_BSP in figure 2]
A parallel-prefix on the calculated values l_i and r_i is performed.
[**Analysis of Stage 5:** The time complexity of this stage is $T^2_{ppf}(p) \leq 2T_{ppf}(p)$.]

Stage 6 : [Line 21 of SELECT_BSP in figure 2]
A broadcast of the two sums $\sum_i l_i$ and $\sum_i r_i$ is realized.
[**Analysis of Stage 6:** This stage takes time $T^2_{brd}(p) \leq T_{ppf}(p)$ time.]

Stage 7 : [Line 22-28 of SELECT_BSP in figure 2]
Let the keys greater than m and less than M form set R. The processors use the results of the prefix operation in stages 5 to compute the $(C - \sum_i l_i)$-th smallest key of set R, by sorting R. The so identified statistic is broadcast to all processors.
[**Analysis of Stage 7:** The results of stages 5 and 6 are used to evenly distribute set R among the processors. By way of the analysis of stage 4 we have that the the number of input keys between m and M and thus the size of R is at most 2λ with probability $1 - O(n^{1-\rho})$. By way of the analysis of step 2 of stage 1, this distribution operation requires time $\max\{L, g\max\{2\lceil 2\lambda/p\rceil, 6e\rho\log n\}\}$ if $p \le n^{2/3+\zeta}$, for any constant $0 < \zeta < 1/3$, and $\max\{L, O(g)\}$ otherwise. Since the number of keys to be sorted is at most 2λ, the time complexity of the sorting operation is that of stage 2. Finally, the broadcast operation requires time $T_{brd} \le T_{ppf}$.]
end

By summing up the computation and communication requirements of procedure SELECT_BSP we obtain the desired time bound. ■

Definition 1 *Let $\hat{g}$ be the value of g in proposition 4 such that $M_{sel}(n,C,p) = o(C_{sel}(n,C,p))$, where $C_{sel}(n,C,p)$ and $M_{sel}(n,C,p)$ denote respectively the computation and communication time of the BSP algorithm implied in proposition 4. In particular,*

$$\hat{g} = \begin{cases} o\left(\frac{n^{\frac{1}{3}-\delta}}{\lg^{\frac{1}{3}} n}\right) & \text{if } p \le n^{\frac{2}{3}+\zeta}, \\ o\left(\frac{n}{p\lg p}\right) & \text{otherwise,} \end{cases}$$

for any arbitrarily small constants ζ and δ, such that $0 < \zeta < \delta < 1/3$.

In the parallel priority-queue algorithms we describe in a following section, it is sometimes desirable that h independent selection operations are performed simultaneously. Each such selection operates on disjoint data sets of size n and the same statistic is involved in all operations. The probabilistic arguments in the analysis of procedure SELECT related to the data distribution, can be extended, due to the independency of the involved problems, to handle simultaneously all h selection problems. This observation can be exploited to decrease the amount of parallel slack required by as much as h. Moreover, since the operations involved in the selection process include broadcast and parallel-prefix operations we could improve the selection process by pipelining these operations (refer to section 2). Hence, we can verify the following analogue of proposition 4.

Proposition 5 *For any constant $\rho > 1$, there exists a randomized BSP algorithm that, with probability $1 - O(n^{1-\rho})$, solves h independent selection problems on the same statistic $C < \lceil n/2\rceil$, and on disjoint data sets, each of size n, and for any integers n and $p \le n$, requires time,*

$$T^h_{sel}(n,C,p) = \begin{cases} h\left(\frac{n+C}{p} + O\left(\sqrt{\frac{C\log n}{hp}} + \frac{n^{\frac{2}{3}+\delta}\lg^{\frac{4}{3}} n}{p} + g\frac{n^{\frac{2}{3}+\delta}\lg^{\frac{1}{3}} n}{p} + L\right)\right) + O(T^h_{ppf}(p)) & \text{if } p \le n^{\frac{2}{3}+\zeta}, \\ h\left(\frac{n+C}{p} + O\left(\sqrt{\frac{C\log n}{hp}} + g\right)\right) + O(\log n + L) + O(T^h_{ppf}(p)) & \text{otherwise,} \end{cases}$$

for any arbitrarily small constants ζ and δ, such that $0 < \zeta < \delta < 1/3$.

Definition 2 *Let $\hat{g}^h$ be the value of g in proposition 5 such that $M^h_{sel}(n,C,p) = o(C^h_{sel}(n,C,p))$, where $C^h_{sel}(n,C,p)$ and $M^h_{sel}(n,C,p)$ denote respectively the computation and communication time of the BSP algorithm implied in proposition 5. In particular,*

$$\hat{g}^h = \begin{cases} o\left(\frac{n^{\frac{1}{3}-\delta}}{\lg^{\frac{1}{3}} n}\right) & \text{if } p \le n^{\frac{2}{3}+\zeta}, \\ o\left(\frac{hn}{p\lg p}\right) & \text{otherwise,} \end{cases}$$

for any arbitrarily small constants ζ and δ, such that $0 < \zeta < \delta < 1/3$.

4.2 Improving lower order terms

In order to improve lower order terms an iterative version of SELECT and subsequently, of SELECT_BSP, has also been designed. The algorithm is based on a bootstrapping technique of [9]. In the discussion to follow, we assume that we can sort r keys in $r \log r$ comparisons. If $dr \log r$ comparisons are required for sorting, for some constant d, this will only affect by a multiplicative constant the lower order terms in the analysis to follow.

Let X be the input key set of cardinality n. At iteration 0, a set $Y_1 \subset X$ of size $|Y_1| = y_1 = m = n^a$, for some constant $\alpha < 1$, is randomly chosen among the keys of the input set. Let m_1 and M_1 be the $(c(y_1 + 1) - \mu_0^1/2)$-th and $(c(y_1+1)+\mu_2^1/2)$-th smallest keys of Y_1 respectively, where $\mu_0^1 = 2\sqrt{3\rho c(y_1+1)\log n}$ and $\mu_2^1 = 2\sqrt{3\rho(1-c)(y_1+1)\log n}$. If $c > 3\rho \log n/(y_1+1)$, then m_1 is defined, otherwise, $m_1 = -\infty$. Pivot keys m_1 and M_1 are selected by sorting set Y_1 in $m \log m$ comparisons (a deterministic selection algorithm may alternately be used).

At iteration 1, a sample Y_2 of size $|Y_2| = y_2 = dm$, $d > 36$, is randomly chosen so that $Y_1 \subset Y_2$. The random selection of $(d-1)m$ additional keys is required. The value of the parameter $d > 36$ is fixed later.

Set Y_1 can be considered as a sample for the larger set Y_2. Let Z_1 be the set of keys of Y_2 between pivot keys m_1 and M_1. Then, the number z_1 of such keys is by claims 5 and 6, for $\lambda = (y_1+1)/2$, $\mu_0^1 = 2\sqrt{3\rho c(y_1+1)\log n}$, $\mu_2^1 = 2\sqrt{3\rho(1-c)(y_1+1)\log n}$, $y_1 = m$, and $N = y_2 = dm$, at most $z_1 = (dm/\sqrt{(y_1+1)/2})(\sqrt{3\rho 2c \log n} + \sqrt{3\rho 2(1-c)\log n})$, with probability $1 - O(n^{1-\rho})$. Set Z_1 is found within $(d-1)m + c(d-1)m + (2dm/\sqrt{(m+1)/2})\sqrt{3\rho 2(1-c)\log n}$ comparisons, by using the comparison sequence of stage 2 of procedure SELECT. It is noted that the relative position of the keys of set Y_1 with respect to m_1 and M_1 is already known, hence no comparisons are required for these keys. The number of keys of set Y_2 which are less than m_1 is also known.

Let m_2 and M_2 of set Z_1 be the $(c(y_2+1) - \mu_0^2/2)$-th and $(c(y_2+1) + \mu_2^2/2)$-th smallest keys of Y_2, where $\mu_0^2 = 2\sqrt{3\rho c(y_2+1)\log n}$ and $\mu_2^2 = 2\sqrt{3\rho(1-c)(y_2+1)\log n}$. If $c > 3\rho\log n/(y_2+1)$, then m_2 is defined, otherwise, $m_2 = -\infty$. Set Z_1 is sorted in $z_1 \log z_1 \leq z_1 \log n$ comparisons to obtain m_2 and M_2. It is noted that the number of keys less than m_1 and greater than M_1 became available when Z_1 was formed. By claim 7, m_2 (and similarly, M_2), with the stated high probability, belongs to set Z_1.

By induction, in the i-th iteration, a set Y_{i+1} of size $|Y_{i+1}| = y_{i+1} = d^i m$ is randomly chosen so that $Y_i \subset Y_{i+1}$, by randomly selecting $d^{i-1}(d-1)m$ additional keys of the input key set. Set Y_i can be considered as a sample for the larger set Y_{i+1}. The keys of Y_{i+1} with values between m_i and M_i form set Z_i and by claims 5 and 6, for $\lambda = (y_i+1)/2$, $\mu_0^i = 2\sqrt{3\rho c(y_i+1)\log n}$, $\mu_2^i = 2\sqrt{3\rho(1-c)(y_i+1)\log n}$, $y_i = d^{i-1}m$, and $N = y_{i+1} = d^i m$ there, the cardinality of Z_i is at most $z_i = (d^i m/\sqrt{(d^{i-1}m+1)/2})(\sqrt{3\rho 2c\log n} + \sqrt{3\rho 2(1-c)\log n})$, with probability $1 - O(n^{1-\rho})$. Set Z_i is found in at most $d^{i-1}(d-1)m + cd^{i-1}(d-1)m + (2d^i m/\sqrt{(d^{i-1}m+1)/2})\sqrt{3\rho 2(1-c)\log n}$ comparisons, using the comparison sequence of stage 2 of procedure SELECT. It is noted that the position of keys of set Y_i relative to m_i and M_i is already known. Let m_{i+1} and M_{i+1} of Z_i, be the $(c(y_{i+1}+1) - \mu_0^{i+1}/2)$-th and $(c(y_{i+1}+1) + \mu_2^{i+1}/2)$-th smallest keys of Y_{i+1}, where $\mu_0^{i+1} = 2\sqrt{3\rho c(y_{i+1}+1)\log n}$ and $\mu_2^{i+1} = 2\sqrt{3\rho(1-c)(y_{i+1}+1)\log n}$. Set Z_i is sorted in $z_i \log z_i \leq z_i \log n$ comparisons and pivot keys m_{i+1} and M_{i+1} can be thus determined.

If $y_{i+1} = n$, then $Y_{i+1} = X$ and Z_i contains the C-th smallest key, with high probability. The desired statistic is computed by selecting the $(C-l)$-th smallest key of Z_i, where l is the number of keys of Y_{i+1} less than m_i. Set Z_i is sorted for this task to be performed. The number of comparisons performed at iteration i is thus,

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

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

Let us fix $m = n^a$, for any constant $a < 1/2$. Given that $md^t = n$, $d > 36$, the total number of comparisons performed in all t iterations is bounded above by $n + C + O(\sqrt{\rho} m^{1/2} d^{(t+1)/2} \log^{3/2} n)$. By substituting for m,d and t this is $n + C + O(\sqrt{\rho} n^{1/2+(1-\alpha)/(2t)} \log^{3/2} n)$. Set Z_t contains with high probability the C-th smallest key.

The total number of comparisons is $n + C + O(n^{1/2+\zeta})$, for any chosen constant $\zeta < 1/2$, by choosing parameters α, t so that $\zeta > (1-\alpha)/(2t)$. Smaller values of m and d can be chosen if t is chosen to be say, logarithmic in n. The smallest value of m, however, is limited by the choices in claims 5 and 6. The probability of failure per iterative step is $O(n^{1-\rho})$ and the number of iterations is constant or at most logarithmic in n. The failure probability of the algorithm can thus be made arbitrarily small by choosing an appropriate constant $\rho > 1$.

The procedure that implements this algorithm for a constant number of iterations is given below.

Theorem 3 *Algorithm* SELECTB *with probability* $1 - O(n^{1-\rho})$, *for any constant* $\rho > 1$, *finds the* C*-th smallest,* $C \leq \lceil n/2 \rceil$, *of* n *keys within* $n + C + O(n^{1/2+\zeta})$ *comparisons, for any constant* $\zeta > 0$.

Proof Algorithm SELECTB that computes the C-th smallest key of a set X of size n is given below. Initially, the variables l_i and r_i are set to zero and S is set to the empty set.

Program SELECTB (X, C)

begin
Stage 0 :
Parameter $c = C/n$ is computed. Constant $a < 1$ and constant integer t are chosen such that $\zeta > (1-a)/(2t)$. Finally integer $d = n^{(1-a)/t} > 32$ is also chosen. Let $m = n^a$.
Set $Y_1 \subset X$ of size $y_1 = m$ is randomly chosen. Let m_1 and M_1 be the $(c(y_1+1) - \mu_0^1/2)$-th and $(c(y_1+1) + \mu_2^1/2)$-th smallest keys of Y_1, where $\mu_0^1 = 2\sqrt{3\rho c(y_1+1)\log n}$ and $\mu_2^1 = 2\sqrt{3\rho(1-c)(y_1+1)\log n}$. If $c > 3\rho \log n/(y_1+1)$, then m_1 is defined, otherwise $m_1 = -\infty$.

for $i = 1$ **to** t **do**
Stage i :
Set $Y_{i+1} \subset X$ is randomly chosen so that $Y_i \subset Y_{i+1}$ and $|Y_{i+1}| = y_{i+1} = d^i m$. Random selection of $d^{i-1}(d-1)m$ additional keys, among the keys of X, is required. The keys of set Y_{i+1} between m_i and M_i are found as follows.

For each key $x \in Y_{i+1}$, x is compared to M_i first. If $x > M_i$, then $r_i = r_i + 1$. If $x < M_i$, then x is compared to m_i. If $x < m_i$ then $l_i = l_i + 1$. Otherwise, $Z_i = Z_i \cup \{x\}$. If $i < t$, let m_{i+1} and M_{i+1} be the keys of Z_i which are the $(c(y_{i+1}+1) - \mu_0^{i+1}/2)$-th and $(c(y_{i+1}+1) + \mu_2^{i+1}/2)$-th smallest keys of Y_{i+1}, where $\mu_0^{i+1} = 2\sqrt{3\rho c(y_{i+1}+1)\log n}$ and $\mu_2^{i+1} = 2\sqrt{3\rho(1-c)(y_{i+1}+1)\log n}$. If $c > 3\rho\log n/(y_{i+1}+1)$, then m_i is defined, otherwise $m_i = -\infty$. The pivot keys are found by sorting Z_i. If any of the keys does not belong to Z_i, failure of the algorithm is declared and the algorithm is reexecuted.

Stage t+1 :
If $l_t < C$ and $r_t < n - C$, then the $(C - l_t)$-th smallest key of set Z_t is found by sorting Z_t. Otherwise, failure of the algorithm is declared and the algorithm is reexecuted.
end

Analysis of the algorithm. This follows from the discussion prior to the description of the algorithm. The cases where $c \leq 3\rho \log n/(y_i+1)$ are handled similarly to the analysis of SELECT. ■

4.3 Outline of another BSP selection algorithm

The BSP implementation of SELECTB is a direct one of SELECT_BSP and we call it SELECTB_BSP. We omit the details here, except to note that it consists of t iterations of the stages outlined in SELECT_BSP. An outline of the operations performed in various stages of SELECTB_BSP in terms of stages of SELECT_BSP is given below. We refer to the analysis of SELECT_BSP for more details.

Program SELECTB_BSP

begin
Stage 1 : This is stage 1 of SELECT_BSP with sample parameters as those described in stage 0 of SELECTB.
Stage 2 : This is stage 2 of SELECT_BSP.
Stage 3 : This is stage 3 of SELECT_BSP.
for $i = 1$ **to** t

begin
Stage 4 : The sample augmentation stage, by randomly sampling additional keys, of stage i of SELECTB is stage 1 of SELECT_BSP.
Stage 5 : This executes stage 4 of SELECT_BSP on Y_{i+1} using M_i, m_i.
Stage 6 : This is stage 5 of SELECT_BSP.
Stage 7 : This is stage 6 of SELECT_BSP.
Stage 8 : This is stage 7 of SELECT_BSP, and set Z_i is sorted.
Stage 9 : If $i < t$ two selection operations are performed similar to the single selection of stage 7 of SELECT_BSP. Since Z_i is sorted, such selections are direct. Otherwise, if $i = t$, a single selection gives the desired statistic as in stage $t+1$ of SELECTB.
end

end

A proposition similar to proposition 4 can then be established.

Proposition 6 *For any constant $\rho > 1$, there exists a randomized BSP selection algorithm that, with probability $1 - O(n^{1-\rho})$, determines the C-th smallest, $C \leq \lceil n/2 \rceil$, of n keys, and for any integers n and $p \leq n$, requires time,*

$$T_{sel}(n, C, p) = \begin{cases} \frac{n+C}{p} + O\left(\sqrt{\frac{C \log n}{p}} + \frac{n^{1/2+\delta} \lg^{3/2} n}{p} + g\frac{n^{1/2+\delta} \lg^{1/2} n}{p} + L\right) + O(T_{ppf}(p)) & \textit{if } p \leq n^{1/2+\zeta}, \\ \frac{n+C}{p} + O\left(\sqrt{\frac{C \log n}{p}} + \log n + g + L\right) + O(T_{ppf}(p)) & \textit{otherwise,} \end{cases}$$

for any arbitrarily small constants ζ and δ, such that $0 < \zeta < \delta < 1/3$.

A similar result to proposition 5 can be obtained from proposition 6 as well.

5 BSP Priority Queues

In this section we study a recently introduced data structure, namely the *Parallel Priority Queue* (PPQ) [27], and formulate a PPQ implementation on the BSP model. We present results on PPQs that improve upon the communication requirements of previous approaches [27] and extend the randomized algorithms of [18, 29, 33] by improving upon both the communication requirements and the amount of parallel slack required to achieve optimal performance. We also show that optimality to within small multiplicative constant factors can be achieved for a wide range of the BSP parameters.

5.1 Preliminaries

The scope of this preliminary discussion is to introduce concepts and notation related to *Parallel Priority Queues (PPQs)*. A PPQ is an abstract data structure for maintaining a collection of (possibly duplicate) items and selecting the n items associated with the smallest values. We assume that the items belong to a universe U equipped with a total order $<$ that can be evaluated in constant time by one processor for any given pair of arguments. The following operations are defined on a PPQ Q.

(1) INSERT($\{y_1, y_2, \ldots, y_n\}$, Q) for the insertion of items $\{y_1, y_2, \ldots, y_n\} \in U$ into Q.

(2) FINDMIN(n, Q) for the determination of the n smallest items of Q.

(3) DELETEMIN(n, Q) for the determination and deletion of the n smallest items of Q.

The implicit assumption is that there exists an operation MAKEQUEUE(Q) for the construction of an empty queue Q. The randomized algorithms we present in this paper for supporting the prementioned PPQ operations are based on a new randomized processor-mapping of a variant of n-Bandwidth-Heaps introduced in [27]. Formally, the new data structure, which we call (n,d)-*Bandwidth-Concurrent-Heap* (in short, (n,d)-BCH) is defined as a concurrent d-ary heap whose nodes, each containing n items, are arranged in *extended heap order*. By extended heap order we imply that the value of an item at any node is no greater than the value of the items at each of its children. By employing (n, d)-BCHs (in contrast to $(n, 2)$-BCHs) we achieve a speedup for insertions at the expense of deletions. We can therefore adapt the parameter d so as to select a data structure that fits the relative frequencies of the operations.

Let m be the number of items stored in a (n, d)-BCH Q, with $k = m/n$ being the number of nodes in Q. The height h of Q is $\lceil \log_d((d-1)k+1) \rceil$, i.e., a (n, d)-BCH Q consisting of a single node has height one. Such a heap Q can be conveniently implemented in terms of the following data structure. Each processor $p^{\langle i \rangle}$, $i \in \{0, 1, \ldots, p-1\}$, maintains a one-dimensional array $A^{\langle i \rangle}[1 \ldots k]$ of size k, where each position of the array is associated with a list $L_j^{\langle i \rangle}$ of items, $j \in \{1, 2, \ldots, k\}$. Each such list $L_j^{\langle i \rangle}$ contains the items of node v_j that reside on processor $p^{\langle i \rangle}$. We note that the order of the items within these lists is irrelevant.

Along the lines of [30], we associate with Q data fields *full_level* and *last_elem*. Data field *full_level* is the index of the leftmost node in the deepest non-empty level h of Q, and *last_elem* is the index of the last non-empty node of Q. For an empty heap, *full_level* and *last_elem* are both set to 0. The path from the root to the first empty node (target) of Q, denoted by σ_{ins}, is called the *insertion-path* of Q. Then, as shown in [31] for the binary case, the insertion path σ_{ins} of Q can be represented by a sequence of $h-1$ digits in radix d, and hence, the next node in the insertion path can be determined in $O(1)$ time. In particular, if we denote by D the displacement of target at the deepest level, $D = \textit{last\_elem} - \textit{full\_level} + 1$, then we can view D as a $(h-1)$-digit radix-d number that prescribes the insertion path σ_i of Q. This path can be traversed as follows. Let $(i_{h-1} i_{h-2} \ldots i_1)_d$ be the radix-d representation of D, then the digits representing D (from the most significant to the least significant) indicate which node (child) to visit when we move from the root downward, i.e., a 0 indicates the leftmost child and a $d-1$ indicates the rightmost child with all intermediate values interpreted accordingly.

5.2 The BSP PPQ Algorithm

We present a new approach towards the implementation of PPQs on the BSP model. Our technique improves previous results, on other parallel models, upon both the communication requirements and the amount of parallel slack required to achieve optimal performance. The general formulation of our randomized technique is as follows. The n insert (respectively, delete-min) processes are realized in a sequence of phases. The number of phases corresponds to the height h of the underlying (n, d)-BCH. At each phase l, $l \in \{1, 2, \ldots, h-1\}$, the processes advance in a pipelined top-down fashion through the data structure along their insert (respectively, delete-min) paths from level l to level $l+1$. Each phase is realized by utilizing the selection algorithm of section 4. The algorithm terminates within h phases. In each phase, the amount of communication is strictly less than the amount of computation required. We note that previous approaches employing Bandwidth-Heaps [27] are quite complex and are based on communication intensive merging and sorting algorithms. We also show that our algorithms achieve optimality to within a small multiplicative constant factor for a wide range of the BSP parameters. The procedures to manage the insert and delete-min operations are summarized in the following proposition.

Proposition 7 *Let ω_n be any function of n, such that $\omega_n > 1$ for sufficiently large n. Then, for any constant $\rho > 0$, there exists a BSP algorithm for supporting the PPQ operations* Insert *and* DeleteMin, *that, with probability $1 - O(dhn^{-\rho})$, for all integers n, p, and h, such that $n/p = \omega_n \lg n$, requires time,*

$$h\left(T_{sel}(2pr, n, p) + T_{ppf}(p) + \max\{L, r + O(1)\} + 2\max\{L, O(1)\}\right) + \max\{L, gr\},$$

and

$$\begin{gathered}(h-1)\left(\sum_{i=0}^{d-2} T_{sel}((d-i)pr, n, p) + T_{sel}(2pr, n, p) + T_{brd}^{d}(p) + \right. \\ \left. 2\max\{L, O(1)\} + \max\{L, O(d)\} + \max\{L, O(d \lg d)\}\right),\end{gathered}$$

for procedures Insert *and* DeleteMin *respectively, where h is the height of the underlying (n, d)-BCH, $r = \lceil (1+\varepsilon)(n/p) \rceil$, and $\varepsilon = O(\sqrt{1/\omega_n})$.*

Proof We initially present the procedures to realize the insertion and deletion operations on a (n,d)-BCH; later, we provide a correctness proof and an analysis of the resources required. Some notation is required before proceeding to the details of the algorithms. Let v be a node in a (n,d)-BCH Q. Along the lines of [27], by E_v we denote the set of n items stored in node v of Q, and by $\max\{E_v\}$ the item of E_v with the maximum value. Moreover, let nodes $\{u^{(1)}, u^{(2)}, \ldots, u^{(d)}\}$ be the children of v. By $\min_j\{u^{(j)}\}$ we denote the child of v that has the minimum $\max\{E_{u^{(j)}}\}$. Finally, the path from the root to the first empty node of Q, denoted by σ_{ins}, is called the insertion-path of Q.

The procedure to manage the insertion operation on a (n,d)-BCH Q is INSERT($\{y_1, y_2, \ldots, y_n\}, Q$) (figure 3), where $Y = \{y_1, y_2, \ldots, y_n\}$ denotes the set of n items to be inserted into Q. We note that in order to achieve communication efficiency we maintain the following randomized mapping of Q on the processors. The items $\{y_1, y_2, \ldots, y_n\}$ are randomly distributed among the p processors at the beginning of the insertion process (line 8). In later phases, whenever items are to be transferred from one node to another they are simply relabeled locally (refer to lines 13, 15-16), and therefore, no inter-processor communication is required during these subsequent phases. This random distribution (line 8) is the only operation where physical movement of items occurs (beyond input and output). Thus, in the remainder of this analysis whenever a data movement is implied, a *local relabeling operation* actually takes place. This relabeling technique guarantees the low-contention of our algorithms and preserves the random distribution of items to processors.

begin INSERT ($\{y_1, y_2, \ldots, y_n\}$, Q)
1. **for** $i = 1$ **to** h
2.   **do** denote by Y set $\{y_1, y_2, \ldots, y_n\}$;
3.   **let** node $v^{(i)}$ at level i, $i \in \{1, 2, \ldots, h\}$, be the next node in the insertion path σ_{ins} of Q ;
4.   **if** $(i = 1)$ **then**
5.   **let** *last_elem* = *last_elem* + 1 ;
6.   **if** (level of *full_level* is full) **then**
7.   **update** *full_level*;
8.   *distribute* randomly the items of Y among the p processors ;
9.   **let** $e = \text{SELECT}(E_{v^{(i)}} \bigcup Y, n)$;
10.   **for** each processor $p^{\langle k \rangle}$, $k \in \{0, 1, \ldots, p-1\}$, in parallel
11.   **do if** $(i = h)$ **then**
12.   **let** $E^{\langle k \rangle}_{v^{(i)}} = Y^{\langle k \rangle}$;
13.   **else**
14.   **let** $E^{\langle k \rangle}_{v^{(i)}} = \{x \mid x \in E^{\langle k \rangle}_{v^{(i)}} \bigcup Y^{\langle k \rangle} \text{ and } x \le e\}$;
15.   **let** $Y^{\langle k \rangle} = \{x \mid x \in E^{\langle k \rangle}_{v^{(i)}} \bigcup Y^{\langle k \rangle} \text{ and } x > e\}$;
16. PARALLEL_PREFIX_MAX($E_{v^{(i)}}$) ;

end INSERT

Figure 3: Procedure INSERT, p - number of processors, h - height of Q.

A graphical example of the insertion process on a $(3,3)$-BCH of 33 items is given in figure 4.

We observe that the insertion process preserves the extended heap order of Q. In particular, for each node $v^{(i)}$, $i \in \{1, 2, \ldots, |\sigma_{ins}|\}$, in the insertion path σ_{ins} of Q, the value of $\max\{E_{v^{(i)}}\}$ is not incremented by procedure INSERT. Consequently, the items stored in the children of $v^{(i)}$ are no less than $\max\{E_{v^{(i)}}\}$ and the correctness of the procedure follows.

The resources required for a single iteration of procedure INSERT are bounded as follows. The determination of the next node in the insertion path of Q (line 3), by way of the analysis presented in section 5.1, requires $O(1)$ sequential time. The local operations (lines 5, 7) also require $O(1)$ time.

The random distribution of items to processors (line 8) takes at most $\max\{L, gr\}$ time with probability $1 - O(n^{-\rho})$, for any constant $\rho > 0$, where $r = \lceil (1+\varepsilon)(n/p) \rceil$, and $\varepsilon = O(\sqrt{1/\omega_n})$. This is true by way of Chernoff bounds on the right tail of the binomial distribution. In particular, we employ the probability bound in [1], $\text{Prob}[S_{n,P} \ge (1+\varepsilon)\, nP] < e^{-(1/3)\varepsilon^2 nP}$, $0 < \varepsilon < 1$, where $S_{n,P}$ is the number of successes

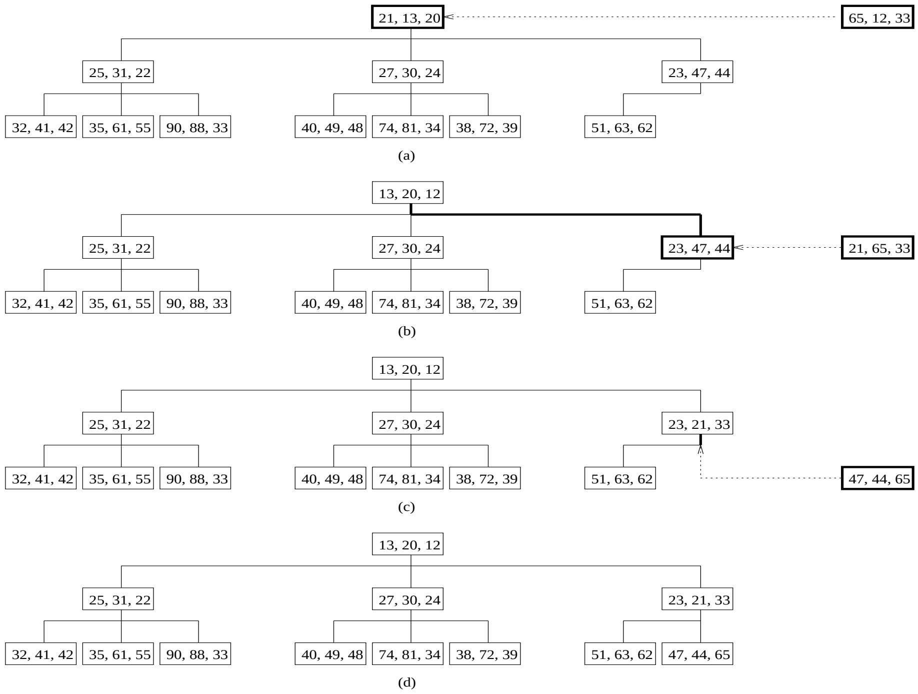

Figure 4: A (3, 3)-BCH of 33 items. Insertion of item set $\{65, 12, 33\}$.

in n independent Bernoulli trials with individual probability of success P. By substituting in this bound for $P = 1/p$ and $\varepsilon = O(\sqrt{1/\omega_n})$, where p is the number of processors, we obtain the desired result. For the remainder of this proof, we therefore assume that each processor has at most $r = \lceil (1+\varepsilon)(n/p) \rceil$ items from any node in Q.

The median selection process (line 9) takes at most $T_{sel}(2pr, n, p)$ time. We note that we use the expression $2pr$ to compensate for the uneven distribution of items to processors, i.e., r items per processors instead of n/p. The actual value of T_{sel} depends on the choice of the selection algorithm as described in propositions 4 and 6. Finally, the relabeling operation can be realized as a by-product of the selection process, and thus, the local operations (lines 12, 14-15) require $O(1)$ time. For implementation purposes, we require that the maximum item of the newly added node is determined and maintained (line 16). This step can be implemented by performing a linear scan over the items of the node followed by a parallel-prefix operation, for a total time of $\max\{L, r + O(1)\} + T_{ppf}(p)$. The time bound on the insertion process follows by observing that the prementioned steps (except from the random distribution in line 8) are iterated $h = |\sigma_{ins}|$ times.

The process to manage the deletion operation on a (n, d)-BCH Q is rather more complicated. We first define an auxiliary routine ADJUST on a set of sibling nodes that does not violate the extended heap property and facilitates the deletion process [27]. In particular, this procedure rearranges the items stored in the children of $v^{(i)}$ so that we can shift up the items of one of the children and still maintain the extended heap order. The process is outlined in procedure $\text{ADJUST}(n, \{u^{(1)}, u^{(2)}, \ldots, u^{(d)}\})$ (figure 5), where n is the size of a node and $\{u^{(1)}, u^{(2)}, \ldots, u^{(d)}\}$ is a set of at most d sibling nodes.

Procedure ADJUST relabels the items stored in nodes $\{u^{(1)}, u^{(2)}, \ldots, u^{(d)}\}$, while maintaining the extended heap order, so that after its execution, node $u^{(\pi(1))}$ holds the n smallest items of the set, node $u^{(\pi(2))}$ holds the next n smallest items of the set, and so on, where initially $\langle u^{(\pi(1))}, u^{(\pi(2))}, \ldots, u^{(\pi(d))} \rangle$ is the sequence of nodes in $\{u^{(1)}, u^{(2)}, \ldots, u^{(d)}\}$, such that $\max\{E_{u^{(\pi(k))}}\} \leq \max\{E_{u^{(\pi(m))}}\}$, for all $k < m \in \{1, 2, \ldots, d\}$. The preservation of the extended heap order follows directly from the fact that $\max\{E_{u^{(j)}}\}$, $j \in \{1, 2, \ldots, d\}$, is not incremented by the process. The time complexity of procedure ADJUST is derived as follows. The determination of sequence $\langle u^{(\pi(1))}, u^{(\pi(2))}, \ldots, u^{(\pi(d))} \rangle$ (line 1) can be realized by broadcasting items $\max\{E_{u^{(j)}}\}$, for all $j \in \{1, 2, \ldots, d\}$, to all processors in time $T^d_{brd}(p)$, and subsequently,

begin ADJUST $(n, \{u^{(1)}, u^{(2)}, \ldots, u^{(d)}\})$
1. **do let** $\langle u^{(\pi(1))}, u^{(\pi(2))}, \ldots, u^{(\pi(d))}\rangle$ be the sequence of nodes in $\{u^{(1)}, u^{(2)}, \ldots, u^{(d)}\}$,
   such that $\max\{E_{u^{(\pi(k))}}\} \leq \max\{E_{u^{(\pi(m))}}\}, \forall\, k < m \in \{1, 2, \ldots, d\}$;
2. **let** $e_j = \text{SELECT}(\bigcup_{l=1}^{d} E_{u^{(l)}}, jn), \forall\, j \in \{1, 2, \ldots, d-1\}$;
3. **for** each processor $p^{\langle k\rangle}$, $k \in \{0, 1, \ldots, p-1\}$, in parallel
4. **do let** $E^{\langle k\rangle}_{u^{(\pi(1))}} = \{x \mid x \in \bigcup_j E^{\langle k\rangle}_{u^{(j)}}$ and $x \leq e_1\}$;
5. **let** $E^{\langle k\rangle}_{u^{(\pi(j))}} = \{x \mid x \in \bigcup_j E^{\langle k\rangle}_{u^{(j)}}$ and $e_{j-1} < x \leq e_j\}, \forall\, j \in \{2, \ldots, d-2\}$;
6. **let** $E^{\langle k\rangle}_{u^{(\pi(d))}} = \{x \mid x \in \bigcup_j E^{\langle k\rangle}_{u^{(j)}}$ and $x > e_{d-1}\}$;

end ADJUST

Figure 5: Procedure ADJUST, p - number of processors.

by sorting locally these items in time $\max\{L, O(d \lg d)\}$ [21]. The selection process (line 2) takes time at most $\sum_{i=0}^{d-2} T_{sel}((d-i)pr, n, p)$. We note that for sufficiently large d it might be more efficient to sort the dn items instead of performing $d-1$ selection processes. The relabeling operation can be realized as a by-product of the selection process, and therefore the local operations (lines 4-6) take $O(d)$ time.

The deletion process is realized by procedure DELETEMIN(n, Q) (figure 6), where n denotes the number of items to be deleted from Q.

begin DELETEMIN (n, Q)
1. **for** $i = 1$ **to** $h - 1$
2. **do if** $(i = 1)$ **then**
3. **return** $E_{root(Q)}$;
4. **let** $root(Q) = v_{last\_elem}$;
5. **let** node $v^{(1)} = root(Q)$ be the first node in the deletion path of Q ;
6. **let** $last\_elem = last\_elem - 1$;
7. **if** $((last\_elem < full\_level))$ **then**
8. **update** $full\_level$;
9. denote by $u^{(1)}, u^{(2)}, \ldots, u^{(d)}$ the children of node $v^{(i)}$;
10. ADJUST $(n, \{u^{(1)}, u^{(2)}, \ldots, u^{(d)}\})$;
11. **let** $e = \text{SELECT}(E_{v^{(i)}} \bigcup E_{\min_j \{u^{(j)}\}}, n)$;
12. **for** each processor $p^{\langle k\rangle}$, $k \in \{0, 1, \ldots, p-1\}$, in parallel
13. **do let** $E^{\langle k\rangle}_{v^{(i)}} = \{x \mid x \in E^{\langle k\rangle}_{v^{(i)}} \bigcup E^{\langle k\rangle}_{\min_j \{u^{(j)}\}}$ and $x \leq e\}$;
14. **let** $E^{\langle k\rangle}_{\min_j \{u^{(j)}\}} = \{x \mid x \in E^{\langle k\rangle}_{v^{(i)}} \bigcup E^{\langle k\rangle}_{\min_j \{u^{(j)}\}}$ and $x > e\}$;
15. **let** node $v^{(i)} = \min_j \{u^{(j)}\}$ be the next node in the deletion path of Q ;

end DELETEMIN

Figure 6: Procedure DELETEMIN, p - number of processors, h - height of Q, d - degree of Q.

A graphical example of the deletion process on a $(3, 3)$-BCH of 36 items is given in figure 7.

We note that the deletion process maintains the extended heap order of Q. In fact, for each node $v^{(i)}$, $i \in \{1, 2, \ldots, h\}$ in the deletion path of Q, the value of $\max\{E_{v^{(i)}}\}$ is not incremented by procedure DELETEMIN by way of procedure ADJUST (line 10). Consequently, the items stored in the children of $v^{(i)}$ are no less than $\max\{E_{v^{(i)}}\}$ and the correctness of the procedure follows. We note that the preservation of the extended heap order is guaranteed by way of the auxiliary procedure ADJUST (line 10), that enables to shift up the items of one of the children of $v^{(i)}$ and still maintains the extended heap order.

The time required for a single iteration of procedure DELETEMIN is determined as follows. The deletion of the n smallest items (line 3) and the local operations (lines 4-5, 6, 8, 15) can be realized in $O(1)$ computation time. The time required for the ADJUST procedure (line 10) was analyzed in the previous paragraphs. The selection process (line 11) takes at most $T_{sel}(2pr, n, p)$ time. The relabeling operations can be realized as a by-product of the selection process, and therefore, the local operations (lines 13-14)

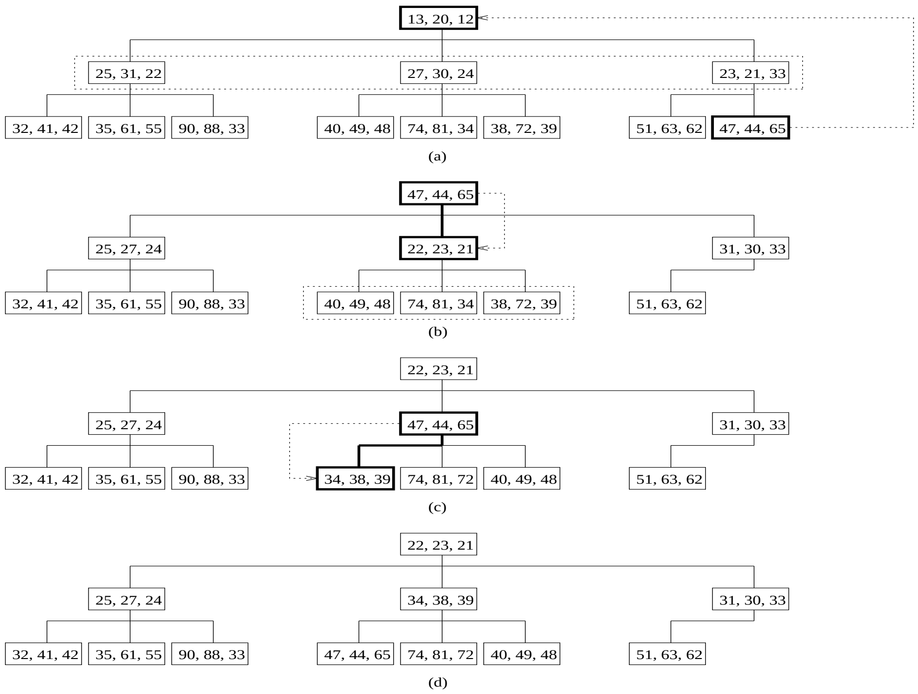

Figure 7: A (3, 3)-BCH of 36 items. Deletion of 3 smallest items.

take $O(1)$ time. Finally, the time bound on the deletion process follows by observing that the prementioned steps are iterated $h-1$ times, where h is the height of the underlying (n,d)-BCH Q.

Procedures INSERT, ADJUST and DELETEMIN perform a total of at most dh selection operations. By way of proposition 4, the failure probability of each such selection operation is $O(n^{-\rho})$. We have also shown that the random distribution of elements to processor (line 8, procedure INSERT) fails with probability $O(n^{-\rho})$, and therefore, the desired probability bound follows. ■

Similarly the following result can be obtained for the find-min operation.

Proposition 8 *There exists a BSP algorithm for supporting the PPQ operation* FINDMIN*, that for all integers n and p, requires time $O(1)$.*

Proof The find-min process is realized by procedure FINDMIN(n, Q) (refer to figure 8), where n denotes the number of minimal items in Q to be determined. ■

begin FINDMIN $(n,\ Q)$
1. **for** each processor $p^{\langle k \rangle}$, $k \in \{0, 1, \ldots, p-1\}$, in parallel
2. **do return** $E^{\langle k \rangle}_{root(Q)}$;

end FINDMIN

Figure 8: Procedure FINDMIN, p - number of processors.

5.3 Improving the Performance of Procedure ADJUST

The performance of auxiliary procedure ADJUST, presented in section 5.2 to facilitate the deletion process, can be improved by utilizing a rather more complicated algorithmic scheme. Recall that procedure ADJUST (figure 5) rearranges the items stored in the children of $v^{(i)}$ so that we can shift up the items of one of the children and still maintain the extended heap order. As outlined in figure 5, the procedure realizes the desired rearrangement by performing $d-1$ selections on a set of size dn. We next describe how procedure ADJUST can be realized by performing 1 and $(d-2)$ selections on sets of sizes dn and n respectively. The process is outlined in procedure ADJUST_H$(n, \{u^{(1)}, u^{(2)}, \ldots, u^{(d)}\})$ (figure 9), where n is the size of a node and $\{u^{(1)}, u^{(2)}, \ldots, u^{(d)}\}$ is a set of at most d sibling nodes.

begin ADJUST_H $(n, \{u^{(1)}, u^{(2)}, \ldots, u^{(d)}\})$
1. **do let** $\langle u^{(\pi(1))}, u^{(\pi(2))}, \ldots, u^{(\pi(d))}\rangle$ be the sequence of nodes in $\{u^{(1)}, u^{(2)}, \ldots, u^{(d)}\}$, such that $\max\{E_{u^{(\pi(k))}}\} \leq \max\{E_{u^{(\pi(m))}}\}$, $\forall\, k < m \in \{1, 2, \ldots, d\}$;
2. **let** $e_1 = \text{SELECT}(\bigcup_{l=1}^{d} E_{u^{(l)}}, n)$;
3. **for** each processor $p^{\langle k\rangle}$, $k \in \{0, 1, \ldots, p-1\}$, in parallel
4. **do let** $r_j^{\langle k\rangle} = |E^{\langle k\rangle}_{u^{(\pi(j))}}|$, $\forall\, j \in \{2, \ldots, d-1\}$;
5. **let** $E^{\langle k\rangle}_{u^{(tmp)}} = E^{\langle k\rangle}_{u^{(\pi(1))}}$;
6. **let** $E^{\langle k\rangle}_{u^{(\pi(1))}} = \{x \mid x \in \bigcup_j E^{\langle k\rangle}_{u^{(\pi(j))}} \text{ and } x \leq e_1\}$;
7. **let** $E^{\langle k\rangle}_{u^{(tmp)}} = E^{\langle k\rangle}_{u^{(tmp)}} - (E^{\langle k\rangle}_{u^{(\pi(1))}} \cap E^{\langle k\rangle}_{u^{(tmp)}})$;
8. **let** $E^{\langle k\rangle}_{u^{(\pi(j))}} = E^{\langle k\rangle}_{u^{(\pi(j))}} - (E^{\langle k\rangle}_{u^{(\pi(1))}} \cap E^{\langle k\rangle}_{u^{(\pi(j))}})$, $\forall\, j \in \{2, \ldots, d\}$;
9. **let** $r_j^{\langle k\rangle} = r_j^{\langle k\rangle} - |E^{\langle k\rangle}_{u^{(\pi(j))}}|$, $\forall\, j \in \{2, \ldots, d-1\}$;
10. PARALLEL_PREFIX_SUM$(\{r_j^{\langle k\rangle}\})$, $\forall\, j \in \{2, \ldots, d-1\}$;
11. BROADCAST$(\{\sum_k r_j^{\langle k\rangle}\})$, $\forall\, j \in \{2, \ldots, d-1\}$;
12. **do let** $e_j = \text{SELECT}(E_{u^{(tmp)}}, \sum_{i=2}^{j} \sum_k r_i^{\langle k\rangle})$, $j \in \{2, \ldots, d-1\}$;
13. **for** each processor $p^{\langle k\rangle}$, $k \in \{0, 1, \ldots, p-1\}$, in parallel
14. **do let** $E^{\langle k\rangle}_{u^{(\pi(2))}} = E^{\langle k\rangle}_{u^{(\pi(2))}} \cup \{x \mid x \in E^{\langle k\rangle}_{u^{(tmp)}} \text{ and } x \leq e_2\}$;
15. **let** $E^{\langle k\rangle}_{u^{(\pi(j))}} = E^{\langle k\rangle}_{u^{(\pi(j))}} \cup \{x \mid x \in E^{\langle k\rangle}_{u^{(tmp)}} \text{ and } e_{j-1} < x \leq e_j\}$, $\forall\, j \in \{3, \ldots, d-1\}$;
16. **let** $E^{\langle k\rangle}_{u^{(\pi(d))}} = E^{\langle k\rangle}_{u^{(\pi(d))}} \cup \{x \mid x \in E^{\langle k\rangle}_{u^{(tmp)}} \text{ and } x > e_{d-1}\}$;

end ADJUST_H

Figure 9: Procedure ADJUST_H, p - number of processors.

Procedure ADJUST relabels the items stored in nodes $\{u^{(1)}, u^{(2)}, \ldots, u^{(d)}\}$, while maintaining the extended heap order, so that after its execution, node $u^{(\pi(1))}$ holds the n smallest items of the set, where initially $\langle u^{(\pi(1))}, u^{(\pi(2))}, \ldots, u^{(\pi(d))}\rangle$ is the sequence of nodes in $\{u^{(1)}, u^{(2)}, \ldots, u^{(d)}\}$, such that $\max\{E_{u^{(\pi(k))}}\} \leq \max\{E_{u^{(\pi(m))}}\}$, for all $k < m \in \{1, 2, \ldots, d\}$. The preservation of the extended heap order follows directly from the fact that $\max\{E_{u^{(j)}}\}$, $j \in \{1, 2, \ldots, d\}$, is not incremented by the adjustment process. The time complexity of procedure ADJUST is derived as follows. The determination of sequence $\langle u^{(\pi(1))}, u^{(\pi(2))}, \ldots, u^{(\pi(d))}\rangle$ (line 1) can be realized by broadcasting items $\max\{E_{u^{(j)}}\}$, for all $j \in \{1, 2, \ldots, d\}$, to all processors in time $T^d_{brd}(p)$, and subsequently, by sorting locally these items in time $\max\{L, O(d \lg d)\}$ [21]. The selection process (line 2) takes time at most $T_{sel}(dpr, n, p)$. The local operations (lines 4-5, 9) require time $O(d)$. The relabeling operation can be realized as a by-product of the selection process, and therefore the local operations (lines 6-8) take $O(1)$ time. The parallel-prefix and broadcast operations (lines 10-11) can be realized in time at most $T^d_{ppf}(p)$ and $T^d_{brd}(p)$ respectively. The selection process (line 12) takes time at most $(d-2)T_{sel}(pr, \lceil n/2\rceil, p) + O(d)$. We note that for sufficiently large d it might be more efficient to sort the at most n items instead of performing $d-1$ selection processes. Finally, the relabeling operation can be realized as a by-product of the selection process, and therefore the local operations (line 14-16) take $O(d)$ time. We therefore obtain the following analogue of proposition 7.

Proposition 9 *Let ω_n be any function of n, such that $\omega_n > 1$ for sufficiently large n. Then, for any constant $\rho > 0$, there exists a BSP algorithm for supporting the PPQ operations* INSERT *and* DELETEMIN, *that, with probability $1 - O(dhn^{-\rho})$, for all integers n, p, and h, such that $n/p = \omega_n \lg n$, requires time,*

$$h\,(T_{sel}(2pr, n, p) + T_{ppf}(p) + \max\{L, r + O(1)\} + 2\max\{L, O(1)\}) + \max\{L, gr\},$$

and

$$(h-1)\left(T_{sel}(dpr,n,p)+(d-2)T_{sel}(pr,\left\lceil\frac{n}{2}\right\rceil,p)+T_{sel}(2pr,n,p)+2T^{d}_{brd}(p)+T^{d}_{ppf}(p)+\right.$$
$$\left. 2\max\{L,O(1)\}+2\max\{L,O(1)\}+\max\{L,O(d\lg d)\}\right),$$

for procedures INSERT *and* DELETEMIN *respectively, where h is the height of the underlying (n,d)-BCH, $r=\lceil(1+\varepsilon)(n/p)\rceil$, and $\varepsilon=O(\sqrt{1/\omega_n})$.*

By substituting appropriate values for the parameters in propositions 9 and 8 a multiplicity of bounds on the time required for PPQ operations can be obtained. As noted in [10], the best time bounds are obtained by selecting these parameters with regard to the actual values of n, p, g, and L. By making however more generic selections for the parameters and worst case bounds for p, g, and L, more general results in terms of n can be obtained. The following is such a bound.

Theorem 4 *For any constant $\rho>0$, there exists a BSP algorithm for supporting the PPQ operations* INSERT, FINDMIN, *and* DELETEMIN, *that, with probability $1-O(dhn^{-\rho})$, for all integers n, p, L, g, d, and h, such that $n/p=\omega(\lg n)$, $\lg d=o(n/p)$, $L=o(n/(p\lg p))$, and $g=o(\min\{\hat{g},n/(p\lg p),h\})$, requires time, $h(4(n/p)+o(n/p))$ per* INSERT, *$O(1)$ per* FINDMIN, *and $(h-1)((5d/2+1)(n/p)+o(dn/p))$ per* DELETEMIN *operation, where h is the height of the underlying (n,d)-BCH and $\hat{g}$ is as in definition 1.*

Proof Substitute in proposition 9 for T_{sel} the corresponding expression in proposition 4. For T_{brd} and T_{ppf} apply lemmas 1 and lemma 2 respectively, after substituting for $t=2$ in these lemmas. ■

We note that in the above theorem if we replace $o(.)$ and $\omega(.)$ by $O(.)$, then the time bounds still hold.

5.4 Introducing Concurrency

The procedures presented in the previous section to manage the insert and delete-min operations can be conveniently pipelined, since both manipulate the underlying (n,d)-BCH heap from top to bottom. The pipelining of these operations can be exploited on a twofold basis. First, procedures INSERT and DELETEMIN heavily utilize parallel-prefix and broadcast operations, and therefore, the underlying algorithms are expected to benefit from pipelining these operations. Second, pipelining can be exploited in order to decrease the amount of parallel slack required to obtain optimality. In the remainder of this section we focus on the latter case. The former case can be treated by a similar line of arguments.

The pipelining process is straightforward when procedures INSERT and DELETEMIN do not operate simultaneously. Then we can verify the following analogue of proposition 9.

Proposition 10 *Let ω_n be any function of n, such that $\omega_n>1$ for sufficiently large n. Then, for any constant $\rho>0$, there exists a BSP algorithm for supporting the PPQ operations* INSERT *and* DELETEMIN *(not simultaneously) in a pipelined fashion, that, with probability $1-O(dhn^{-\rho})$, for all integers n, p, and $\bar{h}$, such that $\bar{h}n/p=\omega_n\lg n$, requires amortized time,*

$$T^{h}_{sel}(2pr,n,p)+T^{h}_{ppf}(p)+h\left(\max\{L,r+O(1)\}+2\max\{L,O(1)\}\right)+M_{distr},$$

and

$$2\left(T^{\frac{h-1}{2}}_{sel}(dpr,n,p)+(d-2)T^{\frac{h-1}{2}}_{sel}(pr,\left\lceil\frac{n}{2}\right\rceil,p)+T^{\frac{h-1}{2}}_{sel}(2pr,n,p)+T^{d\frac{h-1}{2}}_{brd}(p)+T^{d\frac{h-1}{2}}_{ppf}(p)\right)+$$
$$(h-1)\left(2\max\{L,O(1)\}+2\max\{L,O(d)\}+\max\{L,O(d\lg d)\}\right),$$

for procedures INSERT *and* DELETEMIN *respectively, where h is the height of the underlying (n,d)-BCH, $r=\lceil(1+\varepsilon)(n/p)\rceil$, $\varepsilon=O(\sqrt{1/\omega_n})$, $\bar{h}=\min\{(h-1)/2,\omega_n\lg n\}$, and*

$$M_{distr}=\begin{cases}\max\left\{L,g\,O\left(\min\left\{\lg n,\frac{n}{p}\frac{\lg n}{\lg\lg n}\right\}\right)\right\} & \textit{if } \frac{n}{p}=O(\lg n),\\ \max\left\{L,g(1+o(1))\frac{n}{p}\right\} & \textit{otherwise.}\end{cases}$$

Proof The proof follows a similar line of arguments to that of proposition 9 and is briefly sketched in the following discussion. Pipelining is possible since procedures INSERT and DELETEMIN both manipulate the underlying (n, d)-BCH from top to bottom and they both require a small portion of the heap at a time, consisting of one level for the INSERT procedure and two levels for the DELETEMIN procedure. By substituting therefore in the analysis of proposition proposition 9 for the broadcast and parallel-prefix operations, their pipelined versions (refer to lemmas 1 and 2), and by employing proposition 5 we obtain the desired result. We note that the expression for $\bar{h}$ expresses the portion of the degree of concurrency that is exploited to decrease the amount of parallel slack required in proposition 9. The only operation that cannot be efficiently pipelined is the initial distribution of items (line 8) of procedure INSERT (this is due to the synchronous nature of the BSP model). In the remainder of this proof we focus on bounding the time M_{distr} required for this initial distribution. We distinguish two cases.

Case 1: If $\frac{n}{p} = O(\lg n)$, for any constant $\rho > 0$, then the random distribution of items to processors results, with probability $1 - O(n^{-\rho})$, in at most $O(\min\{\lg n, (n/p)\lg n/\lg\lg n\})$ items per processor. This claim can be derived by way of Chernoff bounds on the right tail of the binomial distribution. In particular, we employ the probability bound in [5], $\text{Prob}[S_{n,P} \geq unP] < (u/e)^{-unP}$, where $S_{n,P}$ is the number of successes in n independent Bernoulli trials with individual probability of success P. By substituting in this bound for $u = O(\lg n/\lg\lg n)$ and $P = 1/p$, where p is the number of processors, we obtain the desired result.
Case 2: Otherwise, for any constant $\rho > 0$, the random distribution of items to processors results, with probability $1 - O(n^{-\rho})$, in at most $(1 + o(1))(n/p)$ items per processor. This is true by way of Chernoff bounds on the right tail of the binomial distribution. In particular, we employ the probability bound in [1], $\text{Prob}[S_{n,P} \geq (1+\varepsilon)\, nP] < e^{-(1/3)\varepsilon^2 nP}$, $0 < \varepsilon < 1$, where $S_{n,P}$ is the number of successes in n independent Bernoulli trials with individual probability of success P. By substituting in this bound for $\varepsilon = o(1)$ and $P = 1/p$, where p is the number of processors, we obtain the desired result. ■

Therefore, we derive the following analogue of theorem 4.

Theorem 5 *For any constant $\rho > 0$, there exists a BSP algorithm for supporting in a pipelined fashion the PPQ operations* INSERT, FINDMIN, *and* DELETEMIN, *that for all integers n, p, L, g, d, h, and $\bar{h}$ such that $\bar{h}n/p = \omega(\lg n)$, $\lg d = o(n/p)$, $L = o(\min\{n/p, hn/(p\lg p)\})$, and $g = o(\min\{\hat{g}^h, n/p, hn/(p\lg n), h\})$, with probability $1 - O(dhn^{-\rho})$, requires amortized time, $h(4(n/p) + o(n/p))$ per* INSERT, *$O(n/p)$ per* FINDMIN, *and $(h-1)((5d/2+1)(n/p) + o(dn/p))$ per* DELETEMIN *operation, where h is the height of the underlying (n, d)-BCH, $\bar{h} = \min\{(h-1)/2, \lg n\}$, and $\hat{g}^h$ is as in definition 2.*

Although procedures INSERT and DELETEMIN both manipulate the underlying heap from top to bottom, the problem of realizing them concurrently turns out to be more complicated [30]. In particular, if procedure INSERT is still in progress when procedure DELETEMIN begins its execution, then the target node for procedure DELETEMIN is empty. We are then faced with two options, neither of which is satisfactory: (i) procedure DELETEMIN selects some other leaf node, which might result in an unbalanced heap, and (ii) procedure DELETEMIN delays its operation until the most recent INSERT procedure finishes, which results in limited concurrency. To alleviate these limitations we adopt an implementation that is based on the locking scheme of [30], that achieves concurrency, while preserving the consistent and deadlock-free operation of the underlying algorithms. Instead of locking the entire heap, we lock only a small portion of the heap at a time, consisting of one node for the INSERT procedure and $d + 1$ nodes for the DELETEMIN procedure. Distinct insert and delete-min operations access the contents of a node only after locking it to ensure mutual exclusion. The implementation details follow. We associate a field called *status* with every node in the heap. This field may obtain two distinct values, each associated with the following semantics [30]: *Pending* - an insertion is in progress which will ultimately insert a key at the node, and *Wanted* - a delete-min operation is waiting for the key. When procedure INSERT starts, the status of its target is set to *Pending*. Accordingly, when procedure DELETEMIN starts (and an insert operation is still in progress), the status of the target of the last insert operation is set to *Wanted*, and the DELETEMIN procedure is delayed for one phase. Moreover, at each iteration of procedure INSERT, the status of the target node is checked, and if it has been set to *Wanted*, it is placed at the root and the insertion process for that node is abandoned. Once the node is placed at the root, the DELETEMIN procedure resumes its operation. Finally, data fields *full_level* and *last_elem*, are modified only in the initialization phase of procedures Insert and DeleteMin.

This locking scheme can be implemented through a broadcast operation and a constant number of local operations, and therefore requires time $T_{brd}(p)+O(1)$. This additional cost does not affect (asymptotically) the time bounds of section 5.2. We can therefore obtain a result similar to that of proposition 10 for the case that procedures Insert and DeleteMin operate concurrently.

Remark. It should be noticed, however, that the cost of procedures Insert and DeleteMin can no longer be separated (due to the concurrent nature of the implementation and the synchronous nature of the BSP model), and therefore, we should consider, in each iteration, the combined cost of the two procedures. We can therefore verify the following analogue of proposition 9, where h_I and h_D denote respectively the number of levels locked by operations Insert and DeleteMin. We note that each Insert operation locks one node (corresponding to one level of the underlying heap) and each DeleteMin operation locks $d+1$ nodes (corresponding to two levels of the underlying heap).

Proposition 11 *Let ω_n be any function of n, such that $\omega_n > 1$ for sufficiently large n. Then, for any constant $\rho > 0$, there exists a BSP algorithm for supporting the PPQ operations* Insert *and* DeleteMin *in a pipelined fashion, that, with probability $1 - O(dhn^{-\rho})$, for all integers n, p, and $\bar{h}$, such that $\bar{h}n/p = \omega_n \lg n$, requires amortized time,*

$$
\begin{gathered}
2\left(T_{sel}^{\frac{h_D-1}{2}}(dpr,n,p) + (d-2)T_{sel}^{\frac{h_D-1}{2}}\left(pr,\left\lceil \tfrac{n}{2}\right\rceil,p\right) + 2T_{sel}^{\frac{h-1}{2}}(2pr,n,p) + T_{brd}^{d\frac{h_D-1}{2}+1}(p) + T_{ppf}^{d\frac{h_D-1}{2}+h_I}(p)\right) + \\
h_I\left(\max\{L, r+O(1)\} + 2\max\{L,O(1)\}\right) + \\
(h_D-1)\left(2\max\{L,O(1)\} + 2\max\{L,O(d)\} + \max\{L, O(d\lg d)\}\right) + M_{distr},
\end{gathered}
$$

where h_I and h_D denote, respectively, the height of the underlying (n,d)-BCH that is exploited by operations Insert *and* DeleteMin*, $h = h_I + h_D$, $\bar{h} = \min\{(h_D-1)/2, \omega_n \lg n\}$, $r = \lceil (1+\varepsilon)(n/p)\rceil$, $\varepsilon = O(\sqrt{1/\omega_n})$, and*

$$
M_{distr} = \begin{cases} \max\left\{L, g\,O\left(\min\left\{\lg n, \frac{n}{p}\frac{\lg n}{\lg\lg n}\right\}\right)\right\} & \text{if } \frac{n}{p} = O(\lg n), \\ \max\left\{L, g(1+o(1))\frac{n}{p}\right\} & \text{otherwise.} \end{cases}
$$

The main inefficiency of the above proposition derives from the fact that h_D is a limiting factor in the degree of available concurrency of expressions $T_{sel}^{(h_D-1)/2}(dpr,n,p)$, $T_{sel}^{(h_D-1)/2}(pr,\lceil n/2\rceil,p)$ and $T_{brd}^{d(h_D-1)/2+1}(p)$. We can remedy the situation in several ways. We can modify proposition 5 to handle multiple-selection on disjoint data sets of different sizes, and therefore, by combining low degree of concurrency expressions $T_{sel}^{(h_D-1)/2}(dpr,n,p)$, $T_{sel}^{(h_D-1)/2}(pr,\lceil n/2\rceil,p)$ and $T_{sel}^{(h-1)/2}(2pr,n,p)$ we would achieve a higher degree of concurrency (and similarly for broadcast and parallel-prefix operations). Alternatively, we can limit the notion of d-ary heaps to that of binary ones. The latter approach is summarized in the following proposition.

Proposition 12 *Let ω_n be any function of n, such that $\omega_n > 1$ for sufficiently large n. Then, for any constant $\rho > 0$, there exists a BSP algorithm for supporting the PPQ operations* Insert *and* DeleteMin *in a pipelined fashion, that, with probability $1-O(hn^{-\rho})$, for all integers n, p, and $\bar{h}$, such that $\bar{h}n/p = \omega_n \lg n$, requires amortized time,*

$$
2\left(3T_{sel}^{\frac{h-1}{2}}(2pr,n,p) + T_{brd}^{h}(p) + T_{ppf}^{h-1}(p)\right) + h\left(\max\{L, r+O(1)\} + 5\max\{L,O(1)\}\right) + M_{distr},
$$

where h is the height of the underlying $(n,2)$-BCH, $\bar{h} = \min\{(h-1)/2, \omega_n \lg n\}$, $r = \lceil (1+\varepsilon)(n/p)\rceil$, $\varepsilon = O(\sqrt{1/\omega_n})$, and

$$
M_{distr} = \begin{cases} \max\left\{L, g\,O\left(\min\left\{\lg n, \frac{n}{p}\frac{\lg n}{\lg\lg n}\right\}\right)\right\} & \text{if } \frac{n}{p} = O(\lg n), \\ \max\left\{L, g(1+o(1))\frac{n}{p}\right\} & \text{otherwise.} \end{cases}
$$

By substituting appropriate values for the parameters in proposition 12 a multiplicity of bounds on the time required for PPQ operations can be obtained. The following is such a bound.

Theorem 6 *For any constant $\rho > 0$, there exists a BSP algorithm for supporting in a pipelined fashion the PPQ operations* INSERT*,* FINDMIN*, and* DELETEMIN*, that for all integers n, p, L, g, h, and $\bar{h}$ such that $\bar{h}n/p = \omega(\lg n)$, $L = o(\min\{n/p, hn/(p \lg p)\})$, and $g = o(\min\{\hat{g}^h, n/p, hn/(p \lg n), h\})$, with probability $1 - O(dhn^{-\rho})$, requires amortized time, $10h(n/p) + o(hn/p)$ per* INSERT*, $O(n/p)$ per* FINDMIN*, and $10h(n/p) + o(hn/p)$ per* DELETEMIN *operation, where h is the height of the underlying $(n, 2)$-BCH, $\bar{h} = \min\{h, \lg n\}$, and $\hat{g}^h$ is as in definition 2.*

We note that in the above theorem if we replace $o(.)$ and $\omega(.)$ by $O(.)$, then the time bounds still hold.

6 A Simple PPQ Implementation

In [17, 18], a parallel randomized technique to approximate a priority queue is presented. This technique is extended in [29] to handle exact priority queues. In particular, a randomized variant of the approximate PPQs of [18] is presented in [29] that delivers optimal performance for the insert operation but sub-optimal performance for the find-min and delete-min operations. The main inefficiency of the delete-min and find-min operations derives from a global sorting operation that is required for the implementation of these operations. Sorting has two major drawbacks, (i) it is communication intensive, and (ii) it has super-linear performance. In this work, we propose selection, in order to alleviate these limitations imposed by sorting (see also [33]).

In the remainder of the analysis we assume that the reader is familiar with the work of [29]. Let us outline the BSP adaptation of [29]. The PPQ is maintained as a collection of disjoint local sequential priority queues. For the insertion of n new items, each processor sends its n/p items to uniformly at random chosen processors, where they are inserted into their local sequential priority queues. The procedure to manage the insertion operation on a PPQ Q is INSERT*($\{y_1, y_2, \ldots, y_n\}$, Q) (figure 10).

begin INSERT* ($\{y_1, y_2, \ldots, y_n\}$, Q)
1. denote by Y set $\{y_1, y_2, \ldots, y_n\}$;
2. *distribute* randomly the items of Y among the p processors ;
3. **for** each processor $p^{\langle k \rangle}$, $k \in \{0, 1, \ldots, p-1\}$, in parallel
4. **do** INSERT_SEQ($Y^{\langle i \rangle}, Q^{\langle i \rangle}$) ;

end INSERT*

Figure 10: Procedure INSERT*, p - number of processors.

By way of Chernoff bounds [1], with probability $1 - n^{-\Omega(1)}$, the random distribution of elements to processors causes $O(n/p + \lg n)$ items to be inserted into any local priority queue. Therefore, the resources required for procedure INSERT* are bounded as follows. The random distribution of items to processors (line 2) takes at most $\max\{L, O(g(n/p + \lg n))\}$ time with probability $1 - n^{-\Omega(1)}$. Note that with probability $1 - n^{-\Omega(1)}$ each processor maintains a local PPQ of size $O(m/p + \lg n)$, where m denotes the total number of items in Q (prior to the insertion). The insertion of the $O(n/p + \lg n)$ items in the local priority queues (line 4) requires time $O((n/p + \lg n) \lg(m/p + \lg n))$ [35]. Thus, insertion can be implemented optimally by employing this simple randomized scheme in time $O((n/p + \lg n) \lg(m/p + \lg n)) + \max\{L, O(g(n/p + \lg n)\}$.

For the deletion of the n smallest items, the following procedure is employed [29]. The $2n/p$ smallest items in each processor are duplicated. Let X denote the set of these $2n$ items. By employing selection, the n-th smallest item e of X is determined and broadcast. Let R_i denote the set of items in processor $p^{\langle i \rangle}$, $i \in \{0, 1, \ldots, p-1\}$, that are not larger than e. By using selection, the n-th smallest item e^* from $\bigcup_i R_i$ is determined and broadcast to all processors. Finally, the set S of the n items smaller than e^* is removed from the local priority queues and returned as the result of the delete-min operation. The procedure to manage the deletion operation on a PPQ Q is DELETEMIN*(n, Q) (figure 11). A find-min operation is implemented similarly.

```
begin DeleteMin* (n, Q)
1.    if |Q| = n then
2.        return Q⟨i⟩ ;
3.    else
4.        for each processor p⟨k⟩, k ∈ {0, 1, ..., p − 1}, in parallel
5.            do X⟨i⟩ = FindMin_Seq(2n/p, Q⟨i⟩) ;
6.                let R⟨i⟩ = {} ;
7.                let e = Select(⋃_i X⟨i⟩, n) ;
8.                let x = DeleteMin_Seq(1, Q⟨i⟩) ;
9.                while x ≤ e
10.                   let R⟨i⟩ = R⟨i⟩ ⋃ {x} ;
11.                   let x = DeleteMin_Seq(1, Q⟨i⟩) ;
12.               Insert_Seq(R⟨i⟩ ⋃ {x}, Q⟨i⟩) ;
13.               let S⟨i⟩ = {} ;
14.               let e* = Select(⋃_i R⟨i⟩, n) ;
15.               let x = DeleteMin_Seq(1, Q⟨i⟩) ;
16.               while x ≤ e*
17.                   let S⟨i⟩ = S⟨i⟩ ⋃ {x} ;
18.                   let x = DeleteMin_Seq(1, Q⟨i⟩) ;
19.               Insert_Seq({x}, Q⟨i⟩) ;
20.               return S⟨i⟩ ;
end DeleteMin*
```

Figure 11: Procedure DeleteMin*, p - number of processors.

Let $m \geq n$ denote the total number of items in Q. By replacing sorting by selection in [29] the improvement in the running-time of the delete-min and find-min operations is substantial. In particular, the time required for procedure DeleteMin* is determined as follows. The determination of the $2n/p$ smallest items per local priority queue (line 5) requires $O((n/p)\lg(m/p + \lg n))$ time. The median selection process (line 7) takes time $T_{sel}(2n, n, p)$.

Denote by x the number of items over all processors with value no more than that of item e. We next bound the number of such items per processor. In particular, we employ the probability bound in [5], $\text{Prob}[S_{n,P} \geq unP] < (u/e)^{-unP}$, where $S_{n,P}$ is the number of successes in n independent Bernoulli trials with individual probability of success P. By substituting in this bound for $P = 1/p$ and $u = O(\max\{p \lg n/n, 1\})$, where p is the number of processors, we obtain that with probability $1 - n^{-\Omega(1)}$ no processor maintains more than $O(\max\{x/p, \lg n\})$ items with value no more than that of item e. Therefore, the selection process (line 14) takes time $T_{sel}(O(\max\{x, p\lg n\}), n, p)$. By similar probabilistic arguments we can prove that the sequential delete-min and insert operations per local priority queue (lines 8-12, 15-18) involve $O(n/p + \lg n)$ items and therefore require $O((n/p + \lg n)\lg(m/p + \lg n))$ time. Finally, the local operations (lines 6, 13, 19, 20) take time $O(\lg(m/p + \lg n))$. By arguments similar to those used in lemma 1 of [29] we can upper bound x by $O(n)$. Hence, deletion can be implemented optimally by employing this simple randomized scheme in time $2T_{sel}(O(\max\{n, p\lg n\}), n, p) + O((n/p + \lg n)\lg(m/p + \lg n))$. We therefore obtain the following result.

Theorem 7 *For any constant $\rho > 0$, there exists a BSP algorithm for supporting the PPQ operations* Insert, FindMin, *and* DeleteMin, *that for all integers n, p, L, g and h such that $n/p = \Omega(\lg n)$, $L = O((n\lg(m/p))/(p\lg p))$, and $g = O(\min\{\hat{g}, (n\lg(m/p))/(p\lg p), \lg(m/p)\})$, with probability $1 - O(n^{-\rho})$, requires time, $O((n/p)\lg(m/p))$ per* Insert, *$O((n/p)\lg(m/p))$ per* FindMin, *and $O((n/p)\lg(m/p))$ per* DeleteMin *operation, where m refers to the total number of elements in the underlying PPQ and $\hat{g}$ is as in definition 1.*

To simplify the exposition, in the remainder of this section we assume that $n/p = \Omega(\lg n)$. We note that we can improve the performance of the Insert* and DeleteMin* operations in several ways. For exam-

Processors		$p=2$			$p=4$			$p=8$		
Size ($\times$ *1000)*		256	1024	4096	256	1024	4096	256	1024	4096
Machine	*Statistic*									
IBM SP2	1/2	1.9	1.9	1.9	3.7	3.7	3.8	5.3	6.6	7.2
	1/4	1.8	1.9	1.8	3.5	3.4	3.8	5.0	6.0	7.2
SGI	1/2	1.9	1.9	1.9	2.6	3.1	3.4	-	-	-
	1/4	1.8	1.9	1.9	2.4	3.0	3.5	-	-	-

Table 3: Speedup for Randomized Selection.

ple, the underlying sequential data-structure maintained by each processor is assumed to be an ordinary sequential heap, and therefore, the insertion of $O(n/p)$ items per processor requires time $O((n/p)\lg(m/p))$. If instead of using ordinary heaps, we use leftist heaps [35], then an INSERT* operation can be performed in $O(n/p+\lg(m/p))$ time as follows. Each processor creates an auxiliary leftist heap with the $O(n/p)$ items it receives (line 2) in time $O(n/p)$ [35], and subsequently melds this auxiliary leftist heap with its local leftist heap that is part of Q. The latter operation requires time $O(\lg(n/p)+\lg(m/p))$ [35]. Overall, an improved $O(n/p+\lg(m/p))$ computation time bound can be claimed for insertions. The usage of leftist heaps can also be exploited to obtain $O(n/p)$ computation time for operation FINDMIN*. The general idea is as follows. Each processor maintains a copy of the $2n/p$ smallest elements of its local leftist heap in a separate list. Therefore, the determination of the $2n/p$ items per local heap (line 5 of algorithm DELETEMIN*) requires time $O(1)$. We note that the $2n/p$ smallest items can be maintained in a separate local list in time $O(n/p)$ under the operations INSERT* and DELETEMIN* by employing a linear time sequential selection algorithm. As it is observed in [35], the heap structure of leftist heaps allows the determination of all items not exceeding a prespecified value β in time proportional to the number of such items: we perform a preorder traversal of the underlying leftist heap starting from the root, listing every encountered item with key not exceeding β and immediately retreating from each item with key exceeding β. Therefore, the determination of items with values not exceeding e and e^* (lines 8-12, 15-19 of algorithm DELETEMIN*) requires time $O(n/p)$. The $O(n/p)$ time bound follows. We finally note that on leftist heaps the delete-min operation, and consequently the contributing term $O((n/p)\lg(m/p))$, can be further optimized by using lazy evaluation techniques [35] for an increase in the time of insert operations. Alternatively, we can employ tree-based data structures [6] to improve the performance of delete-min and find-min operations without a corresponding increase in the time required for the insert operation. The following theorem can then be derived.

Theorem 8 *For any constant $\rho > 0$, there exists a BSP algorithm for supporting the PPQ operations* INSERT, FINDMIN, *and* DELETEMIN, *that for all integers n, p, L, g and h such that $n/p = \Omega(\lg n)$, $L = O((n\lg(m/p))/(p\lg p))$, and $g = O(\min\{\hat{g}, (n\lg(m/p))/(p\lg p), \lg(m/p)\})$, with probability $1 - O(n^{-\rho})$, requires time, $O(n/p+\lg(m/p))$ per* INSERT, *$O(n/p)$ per* FINDMIN, *and $O((n/p)\lg(m/p))$ per* DELETEMIN *operation, where m refers to the total number of elements in the underlying PPQ and $\hat{g}$ is as in definition 1.*

7 Experimental Results for Selection

We briefly outline experimental results obtained for procedure SELECT using the Oxford BSP Library. Two statistics were selected, the $n/4$-th smallest and the median of a set of n keys. The speedup indicated in table 3 refers to the average speedup over a series of experiments on a 4-processor SGI Power Challenge shared memory system and an 8-processor IBM SP2 distributed-memory system. The input consisted of integers drawn uniformly at random and the input size varied from 256000 to 4096000 items.

8 Conclusion

In this paper we have presented a new randomized selection algorithm on the BSP model of computation along with an application of this result to dynamic data structures, namely parallel priority queues.

We have showed that our randomized techniques improve previous results upon both the communication requirements and the amount of parallel slack required to achieve optimal performance. We have also established that optimality to within a small constant factor can be achieved for a wide range of parallel machines. We have presented our techniques on the BSP model of computation that allows the description of the computational and communication requirements of algorithms in sufficient detail to allow portable software to be written on a variety of parallel machines whose performance can be consistently predicted. While these algorithms have been fairly simple themselves, descriptions of their performance in terms of these parameters has been somewhat involved. As noted in [10], the main reward of quantifying these complications is that it allows transportable software to be written - once and for all - for parallel machines that fit the model.

References

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

[2] A. Baumker, W. Dittrich, F. Meyer auf der Heide and I. Rieping. Realistic Parallel Algorithms: Priority Queue Operations and Selection for the BSP* Model. *Euro-Par '96 Parallel Processing*, Lecture Notes in Computer Science, Volume 1124, Springer Verlag, 1996.

[3] J. Biswas and J. C. Browne. Simultaneous update of priority structures. In *Proceedings of the 1987 International Conference on Parallel Processing*, pp. 124-131, 1987.

[4] 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*, 7:448-461, 1973.

[5] B. Bollobás. *Random Graphs.* Academic Press, New York, 1984.

[6] H. T. Cormen, C. E. Leiserson, and L. R. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[7] W. Cunto and J. I. Munro. Average case selection. In *Proceedings of the 16-th Annual ACM Symposium on Theory of Computing*, pp. 369-375, 1984.

[8] N. Deo and S. Prasad. Parallel heap: An optimal parallel priority queue. *The Journal of Supercomputing*, 6(1):87-98, 1992.

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

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

[11] A. V. Gerbessiotis and C. J. Siniolakis. Deterministic sorting and randomized median finding on the BSP Model. In *Proceedings of the 8-th ACM Symposium on Parallel Algorithms and Architectures*, June 1996.

[12] A. V. Gerbessiotis and C. J. Siniolakis. Communication efficient data structures on the BSP model with applications. Technical Report PRG-TR-13-96, Computing Laboratory, Oxford University, May 1996.

[13] A. V. Gerbessiotis and C. J. Siniolakis. Selection on the Bulk-Synchronous Parallel model with applications to priority queues. In *Proceedings of the 1996 International Conference on Parallel and Distributed Processing Techniques and Applications*, Sunnyvale, California, USA, August 9-11, 1996.

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

[15] D. W. Jones. Concurrent operations on priority queues. *Communications of the ACM*, 32(1):132-137, 1989.

[16] R. M. Karp, M. Saks, and A. Wigderson. On a search problem related to branch-and-bound procedures. In *Proceedings of the 27-th IEEE Symposium on Foundations of Computer Science*, pp. 19-28, 1986.

[17] R. M. Karp and Y. Zhang. A randomized parallel branch-and-bound procedure. In *Proceedings of the ACM Symposium on theory of computing*, pp. 290-300, 1988.

[18] R. M. Karp and Y. Zhang. Parallel algorithms for backtrack search and-branch and bound. *Journal of the ACM*, 40(3):765-789, 1993.

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

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

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

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

[23] 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.

[24] W. F. McColl. An architecture independent programming model for scalable parallel computing. In *Portability and Performance for Parallel Processors,* (J. Ferrante and A. J. G. Hey, eds.), John Wiley and Sons, 1994.

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

[26] D. Nassimi and S. Sahni. Parallel permutation and sorting algorithms and a new generalized connection network. *Journal of the ACM*, 29(3):642-667, 1982.

[27] M. C. Pinotti and G. Pucci. Parallel priority queues. *Information Processing Letters*, 40:33-40, 1990.

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

[29] A. Ranade, S. Cheng, E. Deprit, J. Jones, and S. Shih. Parallelism and locality in priority queues. In *Proceedings of the 6-th IEEE Symposium on Parallel and Distributed Processing*, pp. 97-103, 1994.

[30] V. N. Rao and V. Kumar. Concurrent access of priority queues. *IEEE Transactions on Computers*, C-37(12):1657-1665, 1988.

[31] V. N. Rao and V. Kumar. Concurrent access of priority queues. Technical Report TR88-06, Computer Science Department, University of Texas at Austin, February 1988.

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

[33] P. Sanders. Fast priority queues for parallel branch-and-bound. In *Proceedings of the 2-nd International Workshop, IRREGULAR '95*, pp. 379-393, LNCS 980, Springer Verlag, 1995.

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

[35] R. E. Tarjan. *Data structures and network algorithms.* Society for Industrial and Applied Mathematics, 1983.

[36] L. G. Valiant. Bulk-synchronous parallel computers. In *Parallel Processing and Artificial Intelligence*, (M. Reeve and S. E. Zenith, eds.), Wiley, 1989.

[37] L. G. Valiant. General purpose parallel architectures. In *Handbook of Theoretical Computer Science*, (J. van Leeuwen, ed.), North Holland, Amsterdam, 1990.

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

[39] L. G. Valiant. A combining mechanism for parallel computers. In *Parallel Architectures and Their Efficient Use*, pages 1-10, LNCS 678, Springer-Verlag, 1993.