Programming Research Group

EFFICIENT DETERMINISTIC SORTING ON THE BSP MODEL

Alexandros V. Gerbessiotis and Constantinos J. Siniolakis

PRG-TR-19-96

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

Efficient Deterministic Sorting on the BSP Model*

Alexandros V. Gerbessiotis and Constantinos J. Siniolakis

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

October 1996

Abstract

We present a new algorithm for deterministic sorting on the Bulk-Synchronous Parallel (BSP) model of computation. We sort n general keys using a partitioning scheme that achieves the requirements of efficiency (1-optimality) and insensitivity against data skew. Although we employ sampling in order to realize efficiency, we can give a precise worst-case estimation of the maximum imbalance which might occur. The algorithm is 1-optimal for a wide range of the BSP parameters in the sense that its speedup on p processors is asymptotically $(1-o(1))p$. Experimental results for the algorithm are also presented.

Keywords. BSP model - algorithms - sorting - deterministic sampling

*The work of the first author was supported in part by EPSRC(UK) under grant GR/K16999, and the second author was supported by a Bodossaki Foundation Graduate Scholarship. An extended abstract of a preliminary version of this paper was presented in the 8-th ACM Symposium on Parallel Algorithms and Architectures, June 1996.

1 Introduction

We present a new Bulk-Synchronous Parallel (BSP) algorithm for deterministic sorting. We sort n general keys on p processors by using a partitioning scheme that achieves the requirements of efficiency (1-optimality) and insensitivity against data skew (load-balance). A distinction between the approach that we consider in this paper and previous BSP approaches [8] is that this one does not sample the unsorted input data but the sorted initial runs. The accuracy of the splitting keys depends solely on the step distance, which can be adapted to meet the worst-case requirements of our application. Therefore, although we employ sampling [8, 11, 18] in order to realize efficiency, we can give a precise worst-case estimation of the maximum imbalance which might occur. The idea of deterministic sampling is not a new one, see [18], however, our techniques improve upon previous results on (i) the amount of parallel slackness required to achieve 1-optimality, (ii) the communication requirements, (iii) the multiplicative constant factors involved. Previously known results on BSP sorting include deterministic [1] and randomized [8] algorithms. In [1], it is shown that for all values of n and p such that $p = n^{1-\epsilon}$, for any constant $0 < \epsilon < 1$, a version of column-sort [14] provides an upper bound of $\lceil \epsilon^{-3.42} \rceil O(n \lg n/p)$ and $O(gn \lg n/p)$ for computation and communication respectively, where g is the ratio of the total throughput of the whole system, in terms of basic computational operations, to the throughput of the router in terms of words of information delivered. The algorithm of [8] requires, with high-probability, computation and communication time $(1+o(1))(T_{seq}/p)$ and $O(gn/p)$ respectively, for all values of n and p such that $p = n^{1-\epsilon}$, $0 < \epsilon < 1$, where T_{seq} is the time required by any sequential sorting algorithm. Our deterministic algorithm complements the randomized result of [8], and improves the upper bound of [1] by reducing the communication requirements by a factor of $\lg n$; the algorithm also significantly reduces the multiplicative constant factor involved in [1]. The analysis of the algorithm presented here improves on the preliminary work in [6]. In particular, our algorithm requires time $(1 + 3\epsilon/2 + o(1))(T_{seq}/p)$ and $O(gn/p)$ for computation and communication respectively. It works even for smaller slack (ratio n/p) and in fact, if $p = n/\lg^{2+a} n$, for any constant $a > 0$, the computation and communication time of the algorithm is $(1 + 2/a)(T_{seq}/p)$ and $O(gn \lg n/(p \lg \lg n))$ respectively.

The BSP model for parallel computation has recently been proposed [20, 21] as a unified framework for the design, analysis and programming of general purpose parallel computing systems. The bulk-synchronous parallel or BSP computer as described in [21] consists of three parts: a collection of processor/memory components; a communication network with a router that can deliver messages point-to-point among the components; and facilities for globally synchronizing, in barrier style, all or a subset of the components.

A BSP computer can be parametrized by p, the number of processor/memory components, L, the minimal time, measured in terms of basic local computation steps, between successive synchronization operations, and $g \geq 1$ the ratio of the total throughput of the whole system in terms of basic computational operations, to the throughput of the router in terms of words of information delivered.

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

If x is the maximum number of local computation steps executed by any processor, h_r and h_s are, respectively, the maximum number of packets that any processor receives or sends during a superstep, then each superstep is charged $\max\{L, x+hg\}$ basic time units, where we $h = \max\{h_r, h_s\}$, although $h = h_r + h_s$ is also possible [20].

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

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

We specify the performance of a BSP algorithm by comparing its performance with a sequential or PRAM algorithm [8]. In order to achieve this, we measure ratios between runtimes on pairs of models

that have the same set of local instructions. We specify two ratios π and μ. The former, π, is the ratio between the computation time of the BSP algorithm, over the time of the comparing say, sequential algorithm, divided by p. The latter, μ, is the ratio between the parallel time required by the communication supersteps of the BSP algorithm and the computation time of the sequential algorithm divided by p. The time of the sequential algorithm divided by p is the best achievable theoretical parallel time for an optimal parallelization of the sequential algorithm. Conditions on n, p, and L are specified that are sufficient for the algorithm to make sense and the claimed bounds on π and μ to hold. In this paper we restrict ourselves to *1-optimal* and *c-optimal* algorithms in the spirit of [8]. Thus we shall say that a BSP algorithm is *c-optimal*, for any constant c, if $\pi = c + o(1)$ and $\mu = o(1)$ as $n \to \infty$. Insistence on 1-optimality leads to algorithms that only work in restricted ranges of n and L. All asymptotic bounds refer to the problem size as $n \to \infty$.

One also needs to specify how the local operations in BSP algorithms are charged. Since we are interested in ratios of performance of two algorithms we can define operations in a higher level of abstraction than machine level instructions. We charge unit time for an operation over an associative operator in parallel prefix computations, and unit time for each comparison performed. A dominant term in the BSP algorithm is contributed by sequential sorting performed independently at each processor. We charge $n \lg n$ for sorting n keys sequentially. Likewise, we charge $n \lg q$ for merging q lists of total size n. If we had charged $cn \lg n$ and $dn \lg q$ for some $c \geq d > 1$, for sorting and multi-way merging respectively, then neither π nor μ would have increased. Finally we assume that a key and its index can be transmitted by the router as one packet. As this model is the one used in [8], our claims for sorting related operations on the BSP drawn from [8] are also consistent with this model.

2 Primitive Operations

In this section we cite BSP algorithms for the fundamental operations of *broadcasting* and *parallel-prefix*. These primitive operations are auxiliary routines for the algorithm to be described in a latter section. Proofs for the claims below can be found in [6].

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* $2 \leq t \leq p$, *where* $h = \lceil \log_t ((t-1)p + 1) \rceil - 1$.

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* $M^n_{ppf}(p) = 2(\lceil n/\lceil n/h \rceil \rceil + h - 1) \max\{L, g2t\lceil n/h \rceil\}$, *for computation and communication respectively, for any integer* $2 \leq t \leq p$, *where* $h = \lceil \log_t p \rceil$, *for a total time of* $T^n_{ppf}(p) = C^n_{ppf}(p) + M^n_{ppf}(p)$.

For $n = 1$, we write $T_{ppf}(p)$ (respectively $C_{ppf}(p)$, $M_{ppf}(p)$) for $T^1_{ppf}(p)$ (respectively $C^1_{ppf}(p)$, $M^1_{ppf}(p)$).

3 Deterministic BSP Sorting

It was recently presented [8] a randomized 1-optimal BSP algorithm for the problem of sorting n general keys. In this section we present a deterministic BSP sorting algorithm that achieves comparable performance. The algorithm is 1-optimal in computation for an ample range of the parameters n, p, and L, and is also 1-optimal in communication if g is appropriately bounded. For a wider range of the parameters the algorithm is c-optimal, where $c \leq 1.75$ and c depends on the ratio $\lg(n/p)/\lg n$. The algorithm is based on an efficient partitioning scheme which splits an arbitrary number of sorted sequences among all the processors almost evenly (independently of the input distribution).

The algorithm consists of the following distinct phases:

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

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

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

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

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

Proposition 1 *Let ω_n be any function of n such that $\omega_n \to \infty$ as $n \to \infty$ and $\omega_n \geq 3$. For any integers p and k such that p is an integral power of k, $m = \lg p / \lg k$ and $r = \lfloor (\lg (n/p) / \lg p) m \rfloor$, there is a deterministic BSP sorting algorithm that with the charging policy of section 1 requires computation and communication time, denoted by $C(n)$ and $M(n)$ respectively,*

$$\begin{aligned} C(n) \leq\ & \max\{L, \lceil n/p \rceil \lg \lceil n/p \rceil\} + m \max\{L, m\lceil \omega_n \rceil p^{1/m}\} + \\ & m(\lceil \lg p \rceil (\lceil \lg p \rceil + 1)/2) \max\{L, 2m\lceil \omega_n \rceil p^{1/m}\} + \\ & m \max\{L, 2((e^{1/\omega_n} \tfrac{n}{p} + m^2 \lceil \omega_n \rceil^2 p^{1/m}) + p^{1/m} - 1)\} + m C_{ppf}^{p^{1/m}}(p) + mL + \\ & (e^{1/\omega_n} \tfrac{n}{p} + m^2 \lceil \omega_n \rceil^2 p^{1/m}) \left(m + \frac{r(r+1)\lg p}{2m} + (m-r)\lg \frac{n}{p} + o(m) + \frac{1.5 p^{1/m} m^3 \lceil \omega_n \rceil^2}{\frac{n}{p} e^{1/\omega_n}} \right) \end{aligned}$$

and

$$\begin{aligned} M(n) \leq\ & m(\lceil \lg p \rceil (\lceil \lg p \rceil + 1)/2) \max\{L, 2gm\lceil \omega_n \rceil p^{1/m}\} + \\ & m \max\{L, g(p^{1/m} - 1)\} + m M_{brd}^{p^{1/m}-1}(p) + \\ & m M_{ppf}^{p^{1/m}}(p) + m \max\{L, g(e^{1/\omega_n} \tfrac{n}{p} + m^2 \lceil \omega_n \rceil^2 p^{1/m})\}. \end{aligned}$$

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

Algorithm SORT (l, u)

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

Procedure SEQUENTIAL_SORT (l, u)

begin
Stage 0 :
Every processor, in parallel with all the other processors, sorts the keys in its possession.

[**Analysis of Stage 0:** When this step is executed, every processor holds at most $\lceil n/p \rceil$ keys. The cost of this stage is, therefore, that of sequential sorting $\lceil n/p \rceil$ keys by some deterministic algorithm such as [13], which is assumed to take time $\max\{L, \lceil n/p \rceil \lg \lceil n/p \rceil\}$.]
end

Procedure RECURSIVE_SORT $(l,\ u)$

begin
Stage 0 :
Each processor selects locally $m\lceil \omega_n \rceil k - 1 \geq 1$ evenly spaced splitters, where $k = p^{1/m}$, that partition its input into $m\lceil \omega_n \rceil k$ evenly sized segments, where ω_n is any function of n such that $\omega_n \to \infty$ as $n \to \infty$. Moreover, each processor appends to this sorted list the largest key in its subsequence. Let $s = m\lceil \omega_n \rceil k$ be the size of the so identified list.
[**Analysis of Stage 0:** Since the keys residing in the local memory of any processor have already been sorted prior to the execution of this stage, this stage requires time $\max\{L, s\}$. We note that if x keys are local to a processor, each segment has either $\lceil x/s \rceil$ or $\lfloor x/s \rfloor$ keys.]

Stage 1 :
The $q = u - l + 1$ sorted lists, each consisting of s splitters, are merged using Batcher's merge-sort algorithm [2]. Let the sequence $S_{l,u} = \{s_{ls+1}, s_{ls+2}, ..., s_{ls+qs}\}$ be the result of the merge operation. Moreover, assume that $S_{l,u}$ is evenly distributed among the q processors, i.e. the subsequence $\{s_{is+1}, ..., s_{(i+1)s}\}$, $l \leq i \leq u$, resides in the local memory of processor p_i.
[**Analysis of Stage 1:** The cost of this stage is that of parallel sorting by one of Batcher's methods [2], appropriately modified [3, 13] to handle lists of size s. The computation and communication time required for this stage is, respectively, $(\lceil \lg q \rceil (\lceil \lg q \rceil + 1)/2) \max\{L, 2s\}$ and $(\lceil \lg q \rceil (\lceil \lg q \rceil + 1)/2) \max\{L, 2gs\}$.]

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

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

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

Stage 5 :
Each processor, in parallel with all other processors, merges the at most q sorted sequences that were received in the previous stage. Note that, since each processor will receive at most S keys, no processor will merge more than $\min\{q, S\}$ sorted sequences.

[**Analysis of Stage 5:** When this step is executed, every processor holds at most q sorted sequences for a total of at most S keys. The cost of this stage is that of sequential q-way merging of at most S keys by some deterministic multi-way merging algorithm such as that presented in [13], which is bounded above by the term $S\min\{\lceil\lg S\rceil, \lceil\lg q\rceil\}$. Thus, the cost of this stage over the m iterations of this procedure is bounded above, for $q = p^{j/m}$, as follows.

$$\sum_{j=1}^{m} \max\{L, S\min\{\lceil\lg S\rceil, \lceil\lg q\rceil\}\} \leq$$
$$mL + \sum_{j=1}^{m} S\min\{\lceil\lg S\rceil, \lceil\lg q\rceil\} \leq$$
$$mL + \sum_{j=1}^{r} S\lceil\lg q\rceil + \sum_{j=r+1}^{m} S\lceil\lg S\rceil. \quad (1)$$

The value of r that minimizes the above expression is determined by comparing q and S. A suitable choice of such an r is $r = \lfloor(\lg(n/p)/\lg p)m\rfloor$. Inequality $\lg(x+y) \leq \lg x + 1.5y/x$, $x > y > 0$, is used below, derived from the fact that $\log(1+x) \leq x$, $0 < x < 1$. Expression 1 is rewritten as follows.

$$mL + \sum_{j=1}^{r} S\lceil\lg q\rceil + \sum_{j=r+1}^{m} S\lceil\lg(e^{1/\omega_n}\tfrac{n}{p} + m^2\lceil\omega_n\rceil^2 p^{1/m})\rceil \leq$$
$$mL + \sum_{j=1}^{r} S(\frac{j}{m}\lg p + 1) + \sum_{j=r+1}^{m} S\left(1 + \frac{1}{\omega_n\log 2} + \lg\frac{n}{p} + \frac{1.5p^{1/m}m^2\lceil\omega_n\rceil^2}{\frac{n}{p}e^{1/\omega_n}}\right) \leq$$
$$mL + Sr + S\frac{r(r+1)\lg p}{2m} + S(m-r) + \frac{S(m-r)}{\omega_n\log 2} + S(m-r)\lg\frac{n}{p} + S\frac{1.5p^{1/m}m^3\lceil\omega_n\rceil^2}{\frac{n}{p}e^{1/\omega_n}} \leq$$
$$mL + S\left(m + \frac{r(r+1)\lg p}{2m} + (m-r)\lg\frac{n}{p} + o(m) + \frac{1.5p^{1/m}m^3\lceil\omega_n\rceil^2}{\frac{n}{p}e^{1/\omega_n}}\right). \quad (2)$$

]

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

Analysis of the Algorithm. By summing up the computation requirements of procedures SEQUENTIAL_SORT and RECURSIVE_SORT we obtain the following bound on $C(n)$.

$$\begin{aligned} C(n) \leq\ & \max\{L, \lceil n/p\rceil\lg\lceil n/p\rceil\} + m\max\{L, m\lceil\omega_n\rceil p^{1/m}\} + \\ & m(\lceil\lg p\rceil(\lceil\lg p\rceil+1)/2)\max\{L, 2m\lceil\omega_n\rceil p^{1/m}\} + \\ & m\max\{L, 2((e^{1/\omega_n}\tfrac{n}{p} + m^2\lceil\omega_n\rceil^2 p^{1/m}) + p^{1/m} - 1)\} + mC_{ppf}^{p^{1/m}}(p) + mL + \\ & (e^{1/\omega_n}\tfrac{n}{p} + m^2\lceil\omega_n\rceil^2 p^{1/m})\left(m + \frac{r(r+1)\lg p}{2m} + (m-r)\lg\frac{n}{p} + o(m) + \frac{1.5p^{1/m}m^3\lceil\omega_n\rceil^2}{\frac{n}{p}e^{1/\omega_n}}\right). \end{aligned}$$

The communication requirements of the sorting algorithm are those of procedure RECURSIVE_SORT and are summarized in the expression below.

$$
\begin{aligned}
M(n) \quad \leq \quad & m(\lceil \lg p \rceil (\lceil \lg p \rceil + 1)/2) \max\{L, 2gm\lceil \omega_n \rceil p^{1/m}\} + \\
& m \max\{L, g(p^{1/m} - 1)\} + mM_{brd}^{p^{1/m}-1}(p) + \\
& mM_{ppf}^{p^{1/m}}(p) + m \max\{L, g(e^{1/\omega_n}\frac{n}{p} + m^2\lceil \omega_n \rceil^2 p^{1/m})\}.
\end{aligned}
$$

■

The correctness of the algorithm follows from its description and is omitted. It remains to prove that procedure RECURSIVE_SORT partitions the input into evenly sized buckets. One way to simplify the exposition below is to assume that $n/(ps)$ is an integer, so that the padding operations to be described below become unnecessary. We choose instead to give a full detailed analysis. The main result is summarized in the following lemma.

Lemma 3 *The size of every bucket produced in the m iterations of procedure* RECURSIVE_SORT *is at most $e^{1/\omega_n} n/p + m^2\lceil \omega_n \rceil^2 p^{1/m}$ for any $m \geq 1$, $s \geq 2$ and $\omega_n \geq 3$.*

Proof: Although it was not mentioned in the description of the algorithm we assume that we pad the input so that each processor owns exactly $\lceil n/p \rceil$ keys. At most one key is added to each processor (the maximum key of that processor can be such a choice). As a result, each processor possesses $\lceil n/p \rceil$ keys. Before performing stage 0 we also pad the input so that after stage 0 all segments have the same number of keys that is, $x = \lceil \lceil n/p \rceil / s \rceil$. The padding operations require time at most $O(s)$ per iteration, which is within the lower order terms of the analysis of proposition 1 and therefore do not affect the asymptotic behavior of our algorithm. We note that padding operations introduce duplicate keys. We have shown how to handle duplicate keys in the beginning of section 3. A more refined approach is to be shown in a later section.

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

$$\frac{ips}{k}x \leq n_{(i)} \leq psx - (ps - \frac{ips}{k} - p + 1)x.$$

Similarly we obtain the following bound.

$$\frac{(i+1)ps}{k}x \leq n_{(i+1)} \leq psx - (ps - \frac{(i+1)ps}{k} - p + 1)x.$$

We therefore conclude the following.

$$n_{(i+1)} - n_{(i)} \leq \frac{ps}{k}x + px.$$

Since $s = m\lceil \omega_n \rceil k$, the following is then derived.

$$n_{(i+1)} - n_{(i)} \leq pm\lceil \omega_n \rceil x + px = px(1 + m\lceil \omega_n \rceil).$$

Considering that $x \leq (n + p + ps)/(ps)$ and substituting for $s = m\lceil \omega_n \rceil k$, the following bound is derived.

$$n_{(i+1)} - n_{(i)} \leq (1 + 1/m\lceil \omega_n \rceil)\frac{n + p(s+1)}{k}.$$

By solving the recurrence relation, we obtain that the size of any bucket after m iterations of procedure RECURSIVE_SORT is bounded above by the following expression.

$$a^m \frac{n}{p} + (s+1)\frac{a}{a-1}(a^m - 1),$$

where $a = (1 + 1/m\lceil\omega_n\rceil)$. Finally, for $m \geq 1$, $s \geq 2$ and $\omega_n \geq 3$ and since $e^{1/\omega_n} \leq 1 + 1/(\omega_n - 1)$ (using Taylor series expansion) we conclude that the size of any bucket at any iteration is bounded above as follows.

$$e^{1/\omega_n}\frac{n}{p} + m^2\lceil\omega_n\rceil^2 p^{1/m}.$$

■

The following theorem is derived from proposition 1 and establishes more concrete bounds for the parameters π and μ of the BSP model.

Theorem 1 *For any integers p and k such that p is an integral power of k, $m = \lg p/\lg k$ and $r = \lfloor(\lg(n/p)/\lg p)m\rfloor$, there is a BSP sorting algorithm for sorting n keys on $p = n^{1-t}$ processors, where $0 < t < 1$, that with the charging policy of section 1 has*

$$\pi \quad = \quad t + \frac{r(r+1)(1-t)}{2m} + (m-r)t + o(1)$$

and

$$\mu \quad = \quad O\left(g\frac{m^2\omega_n p^{1+1/m}\lg^2 p}{n\lg n} + g\frac{m}{\lg n}\right).$$

for $L = o(n\lg n/(mp))$ provided that $m = o(\lg n)$, $p^{1/m}m^2\omega_n^2 = o(n/p)$ and $\omega_n p^{1/m}m^2\lg^2 p = o(n\lg n/p)$.

Proof: Let $p = n^{1-t}$, where $0 < t < 1$. We are going to simplify the expressions for computation and communication time given in the statement of proposition 1. We first consider the computation time of the sorting algorithm, where $S = e^{1/\omega_n}n/p + m^2\lceil\omega_n\rceil^2 p^{1/m}$ below.

$$\begin{aligned} C(n) \quad \leq \quad & \max\{L, \lceil n/p\rceil\lg\lceil n/p\rceil\} + m\max\{L, m\lceil\omega_n\rceil p^{1/m}\} + \\ & m(\lceil\lg p\rceil(\lceil\lg p\rceil + 1)/2)\max\{L, 2m\lceil\omega_n\rceil p^{1/m}\} + \\ & m\max\{L, 2(S + p^{1/m} - 1)\} + mC_{ppf}^{p^{1/m}}(p) + mL + \\ & S\left(m + \frac{r(r+1)\lg p}{2m} + (m-r)\lg\frac{n}{p} + o(m) + \frac{1.5p^{1/m}m^3\lceil\omega_n\rceil^2}{\frac{n}{p}e^{1/\omega_n}}\right). \end{aligned}$$

We employ for the broadcasting and parallel prefix operations binary trees and thus the parameter t in lemmas 1 and 2 is set to 2. Since $p^{1/m}m^2\omega_n^2 = o(n/p)$ the expression for $C(n)$ can be bounded as follows.

$$\begin{aligned} C(n) \quad \leq \quad & O(mL) + (n/p)\lg(n/p) + O(m^2\omega_n p^{1/m}\lg^2 p) + \\ & O(m\frac{n}{p}) + S(m + \frac{r(r+1)\lg p}{2m} + (m-r)\lg\frac{n}{p} + o(m)). \end{aligned}$$

We bound $L = o(n\lg n/(pm))$, $m = o(\lg n)$ and take into consideration that $m^2\lg^2 p\omega_n p^{1/m} = o(n\lg n/p)$. The expression for $C(n)$ can then be written in the following way.

$$C(n) \quad \leq \quad (n/p)\lg(n/p) + S(\frac{r(r+1)\lg p}{2m} + (m-r)\lg\frac{n}{p}) + o(n\lg n/p).$$

We set $S = e^{1/\omega_n}n/p + m^2\lceil\omega_n\rceil^2 p^{1/m}$, $p = n^{1-t}$ and note that $e^{1/\omega_n} \leq 1 + 1/(\omega_n - 1) = 1 + o(1)$. We then get the following bound for $C(n)$.

$$C(n) \quad \leq \quad \frac{n\lg n}{p}(t + \frac{r(r+1)(1-t)}{2m} + (m-r)t) + o(n\lg n/p).$$

The desired expression for π in the statement of the theorem then follows. The expression for μ is established similarly. ■

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

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

Proof: Let in theorem 1, $\omega_n = \lg \lg n$ and $m = 1$ so that $k = p$ and $r = 1$. Then, the conditions of theorem 1 are satisfied and we obtain that $\pi = 1 + o(1)$. If in addition, $g = o(\lg n)$, then we also have $\mu = o(1)$. ■

By allowing more than one iteration the algorithm achieves almost one optimality for values of p much closer to n than those of corollary 1.

Corollary 2 *For any constant t, $0 < t \leq 1/2$, there exists a BSP algorithm for sorting n keys that for all integers n, p, and L satisfying $p = n^{1-t}$ and $L = o(n/(p \lg n))$ has $\pi = 1 + 3t/2 + o(1)$, and if $g = o(\lg n)$ also, $\mu = o(1)$.*

Proof: The conditions $p^{1/m} m^2 \omega_n^2 = o(n/p)$ and $\omega_n p^{1/m} m^2 \lg^2 p = o(n \lg n/p)$ of the statement of theorem 1 are both satisfied by considering $m = \lfloor \frac{1-t}{t} \rfloor + 1$. It is then, $\frac{1-t}{t} \leq m \leq \frac{1-t}{t} + 1$. Substituting for $p = n^{1-t}$ in the statement of theorem 1 we obtain $r = \lfloor \frac{t}{1-t} m \rfloor$. By using the bounds on r and m previously derived we obtain the following bound.

$$\begin{aligned} t + \frac{r(r+1)(1-t)}{2m} + (m-r)t &\leq t + \frac{t(r+1)}{2} + (m-r)t \\ &\leq \frac{3t}{2} + mt - rt/2 \\ &\leq \frac{3t}{2} + mt \\ &\leq 1 + \frac{3t}{2}. \end{aligned}$$

We thus conclude that $\pi = 1 + 3t/2 + o(1)$. For the communication time, if $g = o(\lg n)$ we get that $\mu = o(1)$. ■

When smaller slack is available the following corollary can be derived.

Corollary 3 *For any constant $a > 0$, there exists a BSP algorithm for sorting n keys that for all integers n, p, and L such that $p = n/\lg^{3+a} n$ and $L = o(n/(p \lg n))$ has $\pi = 1 + 3/a + o(1)$, and if $g = o(\lg \lg n)$ also, $\mu = o(1)$.*

Proof: Let $m = \lg p/(a \lg \lg p)$, for any constant $a > 0$. The claims below hold even if a is not constant provided that $m = o(\lg n)$. In order to simplify the analysis and the claims, let us assume that m is an integer. As in the proof of the previous corollary, the conditions $p^{1/m} m^2 \omega_n^2 = o(n/p)$ and $\omega_n p^{1/m} m^2 \lg^2 p = o(n \lg n/p)$ in the statement of theorem 1 are both satisfied by considering $\omega_n = \lg \lg n$ for any $t \geq (3+a) \lg \lg p / \lg p$. Substituting for $p = n^{1-t}$ we get $r = \lfloor \frac{t}{1-t} m \rfloor$. By using the bounds on r and m previously derived and by taking $t = (3+a) \lg \lg p / \lg p$ we obtain the following bound.

$$\begin{aligned} t + \frac{r(r+1)(1-t)}{2m} + (m-r)t &\leq t + \frac{t(r+1)}{2} + (m-r)t \\ &\leq \frac{3t}{2} + mt - rt/2 \\ &\leq \frac{3t}{2} + mt \end{aligned}$$

$$\leq \quad mt + O(a \lg\lg p / \lg p)$$
$$\leq \quad 1 + \frac{3}{a} + O(a \lg\lg p / \lg p).$$

We thus conclude that $\pi = 1 + 3/a + o(1)$. For the communication time, if $g = o(\lg\lg n)$ we get that $\mu = o(1)$. ■

We note that not only can the bounds given above be expressed in a variety of ways, but also the algorithm itself leaves room for several improvements. As stated, the algorithm sorts the splitters using Batcher's merge sort algorithm. In certain circumstances its performance can be improved by any of several alternative methods. For example, the sorting of the splitters could be performed by using either our algorithm recursively or some completely different algorithm [8]. Particularly, since the set of splitters per processor is already sorted we may use a merging algorithm for sorting the splitters. Then, k sorted sequences each of size s and distributed evenly among k processors are merged. We perform this merging using a binary tree execution sequence of depth $\lceil \lg k \rceil$, where in the i-th iteration pairs of sequences of size $2^{i-1}s$ are merged into sequences of size $2^i s$, where each such sequence is evenly distributed among 2^i processors. This algorithmic idea can be implemented by employing the BSP merging algorithm of [7]. The latter algorithm derives from [19] as adapted in [10], and uses the idea of deterministic oversampling. The BSP merging algorithm runs in time $O(n/p + gn/p + T_{ppf}(p) + T_{brd}^{n/p}(p))$. The contribution to the running time of stage 1 of RECURSIVE_SORT is then reduced by a multiplicative factor of $O(\lg p)$. The conditions of theorem 1 are relaxed to $p^{1/m} m^2 \omega_n^2 = o(n/p)$ and $\omega_n p^{1/m} m^2 \lg p = o(n \lg n/p)$. As a result, corollary 3 takes the following form.

Corollary 4 *For any constant $a > 0$, there exists a BSP algorithm for sorting n keys that for all integers n, p, and L such that $p = n/\lg^{2+a} n$ and $L = o(n/(p \lg n))$ has $\pi = 1 + 2/a + o(1)$, and if $g = o(\lg\lg n)$ also, $\mu = o(1)$.*

The proof of this corollary follows that of corollary 3.

4 Duplicate Keys

The algorithm, as described, can not handle duplicate keys. One way to allow duplicate keys is by making them distinct. This can be achieved by attaching to each key the address of the memory location it is stored in (array index etc). For data types whose bit or byte length is comparable to that of the address of a memory location, such a transformation may lead, in most cases, to doubling the overall number of comparisons performed. For more complex data types such as strings of characters the extra cost may be negligible.

An alternative way to handle duplicate keys is the following one. SEQUENTIAL_SORT uses any stable sequential algorithm. In RECURSIVE_SORT all keys have attached to them the address of the memory location they are stored in. The attached address is only used in stages 1-3. These stages require distinct keys to achieve stability. The computational and communication cost of these stages is within the lower order terms of the analysis of corollaries 1, 2, 3 and 4 and therefore our optimality claims still hold.

5 Experimental Results

Below we present indicative experimental results for the sorting algorithm on (a) a Silicon Graphics Power Challenge shared memory system, consisting of four 64-bit Streaming Superscalar TFP processors with 512MBytes RAM, (b) an IBM 9076 close-coupled distributed memory system, consisting of eight IBM RS/6000 model 580 processors, each with 128MByte RAM, connected by a High Performance Switch. All software has been written in C - by the authors - and was linked with the Oxford BSP Library Version 1.2. The source code was compiled using `cc` with option `-O3` under the operating systems `IRIX 6.0.1` and `AIX 3.2` on the Silicon Graphics and IBM systems respectively. The Oxford BSP Library function

`bsp_time` was used for measuring running times. For sequential sorting we used the `qsort` standard C library function.

For each test problem instance, we performed a series of 8 experiments, and the running times indicated are the average of those 8 experiments. We tested our algorithms on two different machines with a varying number of input sizes. Figures 1 and 2 compare the optimal and actual speedup of our algorithm for 32,000 to 1,024,000 elements.

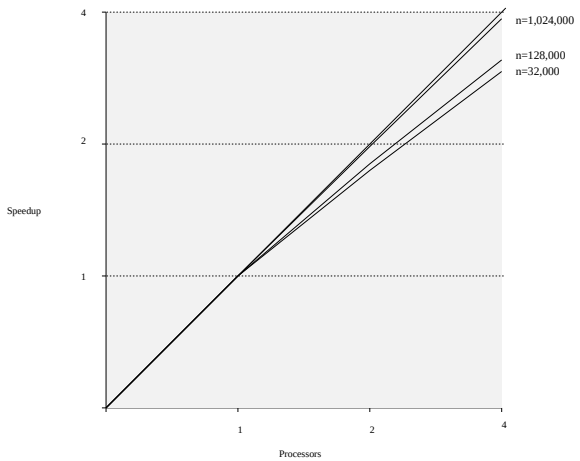

Figure 1: Speedup achieved by deterministic sorting on a SGI Power Challenge system.

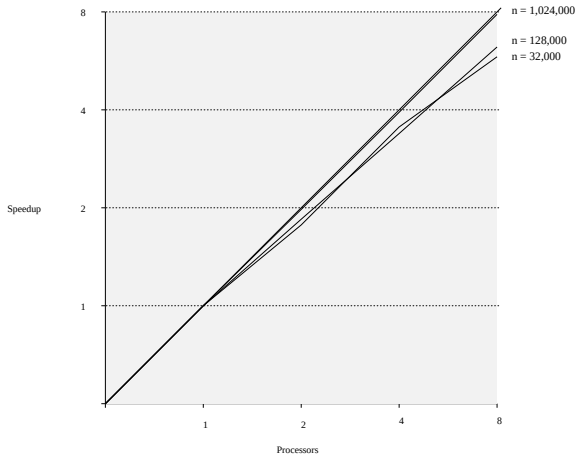

Figure 2: Speedup achieved by deterministic sorting on an IBM 9076 SP1 system.

References

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

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

[3] G. Baudet and D. Stevenson. Optimal sorting algorithms for parallel computers. *IEEE Transactions on Computers*, C-27(1):84-87, January 1978.

[4] W. D. Frazer and A. C. McKellar. Samplesort: A sampling approach to minimal storage tree sorting. *Journal of the ACM*, 17(3):496-507, 1970.

[5] 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*, Padova, Italy, June 1996.

[6] 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.

[7] A. V. Gerbessiotis and C. J. Siniolakis. Primitive Operations on the BSP model. Technical Report, Computing Laboratory, Oxford University, October 1996.

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

[9] C. A. R. Hoare. Quicksort. *The Computer Journal*, 5:10-15, 1962.

[10] T. Hagerup and C. Rub. Optimal Merging and Sorting on the EREW PRAM. *Information Processing Letters,* 33:181-185, 1989.

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

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

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

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

[15] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays - Trees - Hypercubes.* Morgan Kaufmann, California, 1991.

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

[17] 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.

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

[19] Y. Shiloach and U. Vishkin. Finding the maximum, merging, and sorting in a parallel computation model. *Journal of Algorithms,* 2:88-102, 1981.

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

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