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

(Revised Version)

Abstract

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

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 keys on p processors by using a partitioning scheme that achieves the requirements of efficiency (one-optimality) and insensitivity against initial data distribution. A distinction between the approach that we consider in this paper and previous BSP approaches [8] is that our method does not sample the unsorted input data but the sorted initial runs. The accuracy of the splitting keys depends solely on user defined parameters, which can be adapted to meet the worst-case requirements of parallel applications. Therefore, although we employ sampling [8, 14, 19] to realize efficiency, we give a precise worst-case estimation of the maximum imbalance which might occur. The idea of deterministic sampling is not a new one, see [19], however, our techniques improve upon previous results on (i) the amount of parallel slackness required to achieve optimality, (ii) the communication requirements, and (iii) the multiplicative constant factors involved.

2 The BSP Model

The *Bulk-Synchronous Parallel* (BSP) model of computation has been proposed [21, 22] as a unified framework for the design, analysis and programming of general purpose parallel computing systems. A BSP machine as described in [17, 18, 21, 22] consists of three parts: (i) a collection of *processor/memory components*, (ii) a *communication network* that can deliver messages point-to-point among the processors, and (iii) a *facility for global synchronization*, in barrier style, of all or a subset of the processors.

A BSP machine can be parametrized by p, the number of processors, L, the minimal time, measured in terms of basic computation steps, between successive synchronization operations, and g the ratio of the total throughput of the whole system in terms of basic computation steps, to the throughput of the router in terms of words of information delivered. A detailed discussion of the BSP model can be found in [22].

2.1 The Cost Model

Computation on the BSP model proceeds in a succession of *supersteps*. In each 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. Each superstep is charged time x and $\max\{L, g\,h\}$ for computation and communication respectively, where x is the maximum number of basic computation steps executed by any processor during a superstep and h is the maximum number of messages transmitted or received by any processor [22].

We specify the performance of a BSP algorithm $\mathcal{A}$ by comparing its performance with a sequential algorithm $\mathcal{A}^*$ [8]. To this end, 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 $C_{\mathcal{A}}$ of the BSP algorithm, over the time $C_{\mathcal{A}^*}$ of algorithm $\mathcal{A}^*$, divided by p, i.e., $\pi = (p\,C_{\mathcal{A}})/C_{\mathcal{A}^*}$. The latter, μ, is the ratio between the parallel time $M_{\mathcal{A}}$ required by the communication supersteps of the BSP algorithm and the computation time $C_{\mathcal{A}^*}$ of algorithm $\mathcal{A}^*$, divided by p, i.e., $\mu = (p\,M_{\mathcal{A}})/C_{\mathcal{A}^*}$. Conditions on n, p, L and g are then specified that are sufficient for the algorithm to make sense and the claimed bounds on π and μ to hold. In this paper we restrict ourselves to *one-optimal* and *c-optimal* algorithms in the spirit of [8]. Thus, we say that a BSP algorithm is c-optimal, for any constant $c \geq 1$, if $\pi = c + o(1)$ and $\mu = o(1)$ as $n \to \infty$. Insistence on one-optimality leads to algorithms that only work for restricted ranges of p, L and g. All asymptotic bounds refer to the problem size as $n \to \infty$. Let $\lg p$ denote the logarithm base two of p.

We also specify how to charge for local operations in BSP algorithms. 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 constant time for each operation over an associative operator in parallel prefix computations, and constant time for each comparison performed. A dominant term in the BSP algorithm to be described later is contributed by sequential sorting performed independently at each processor. We charge $c_s n \lg n$ time for sorting n keys sequentially [15]. Likewise, we charge $c_r n \lg q$ time for merging q lists of total size n [15]. In the remainder of this work we assume that c_s and c_r are constant.

3 Related Work

Previously known results on BSP sorting include deterministic [1, 5] and randomized [8, 12] algorithms. In [1], it is shown that for all values of n and p such that $n \geq p$, a BSP version of column-sort [16] requires $O((\lg n / \lg (n/p))^{3.42} ((n/p) \lg (n/p)))$ and $g\, O((\lg n / \lg (n/p))^{3.42} (n/p))$ time for computation and communication respectively. In [5], it is shown that an adaptation on the BSP model of cube-sort [7] takes $O((25^{\lg^* n - \lg^* (n/p)}) (\lg n / \lg (n/p))^2 ((n/p) \lg (n/p)))$ and $g\, O((25^{\lg^* n - \lg^* (n/p)}) (\lg n / \lg (n/p))^2 (n/p))$ time for computation and communication respectively. An adaptation of the AKS sorting network [2] on the BSP model is also shown in [5] to yield computation and communication time $O((n/p) \lg n)$ and $g\, O((n/p) \lg p)$ respectively. The constant factors hidden in the aforementioned algorithms, however, are considerably large.

For an ample range of the parameters n and p, i.e., $p = n / \lg^{1+\alpha} n$, $\alpha = \Omega(1)$, the algorithm of [8] requires – under realistic assumptions – computation and communication time $(1 + o(1)) (T_{srt}(n)/p)$ and $g\, O((\lg n / \lg (n/p)) (n/p))$ respectively, with high-probability, where $T_{srt}(n)$ refers to the time required by sequential sorting. In [12], this result is improved by demonstrating that the same time bounds is achieved for any $p = o(n \lg \lg n / \lg n)$. Under the same realistic assumptions as [8, 12], our deterministic sorting algorithm complements the randomized results of [8, 12], and improves upon the upper bounds of [1, 2, 5, 7, 16] by reducing the computation and/or the communication requirements of the underlying algorithms. In particular, for an ample range of the parameters n and p, i.e., $p = n / \lg^{2+\alpha} n$, $\alpha = \Omega(1)$, our algorithm requires computation and communication time $(1 + 2/\alpha + o(1)) (T_{srt}(n)/p)$ and $g\, O((\lg n / \lg (n/p)) (n/p))$ respectively.

Recently, in [13] a BSP adaptation of parallel merge-sort [6] is presented that for all values of n and p such that $p \leq n$, requires computation and communication time $O((n/p) \lg n)$ and $g\, O((n/p)(\lg n / \lg (n/p)))$ respectively. As the BSP adaptation of the algorithm is at least as involved as the original parallel merge-sort algorithm [6], the constant factors involved in the analysis are considerably large and the algorithm seems to be of little practical use. In contrast, our algorithm has been implemented and encouraging experimental results on several variations of the algorithm have been reported in [9, 11].

4 Primitive Operations

In this section we claim BSP results for the operations of *broadcasting* and *parallel-prefix*. These primitive operations are auxiliary routines for the algorithm to be described later. Proofs for the claims below can be found in [10].

Lemma 1 *There exists a p-processor BSP algorithm for broadcasting an n-word vector that requires time*[1]

$$M^n_{brd}(p) = O\left(\max\{L, g\,O(n)\}\right)\,[n \geq p] + O\left(h\,\max\{L, g\,O(\lceil n/h \rceil)\}\right)\,[n < p],$$

for communication, where $h = O(\lg p)$. The overall time complexity of the algorithm is denoted by $T^n_{brd}(p) = M^n_{brd}(p)$.

Lemma 2 *There exists a p-processor BSP algorithm for realizing n disjoint prefix operations, each of size p, that requires time*

$$C^n_{ppf}(p) = O(n)\,[n \geq p] + O(h\,\lceil n/h \rceil)\,[n < p]$$

and

$$M^n_{ppf}(p) = O\left(\max\{L, g\,O(n)\}\right)\,[n \geq p] + O\left(h\,\max\{L, g\,O(\lceil n/h \rceil)\}\right)\,[n < p],$$

for computation and communication respectively, where $h = O(\lg p)$. The overall time complexity of the algorithm is denoted by $T^n_{ppf}(p) = C^n_{ppf}(p) + M^n_{ppf}(p)$.

5 Deterministic BSP Sorting

In this section we present a deterministic BSP algorithm for the problem of sorting n keys. The algorithm is one-optimal for an ample range of the parameters n, p, L and g. For a wider range of the parameters the algorithm is c-optimal, where $c \geq 1$ is a small constant. The algorithm is based on an efficient partitioning scheme that splits – almost evenly and independently of the input distribution – an arbitrary number of sorted sequences among the processors. The algorithm consists of the following phases:

(1) *Local Sorting.* Each processor, in parallel with all the other processors, sorts the subsequence of size $\lceil n/p \rceil$ that resides in its local memory.

(2) *Partitioning.* Processors cooperate to evenly split the sorted sequences among the processors.

(3) *Merging.* Each processor, in parallel with all the other processors, merges a small number of subsequences of total size $(1 + o(1))\,(n/p)$.

The iterative version of the algorithm performs m iterations of phases 2 and 3. In each iteration the data set is partitioned into k buckets of approximately equal size. In the next iteration a similar process of sub-partitioning each of these k buckets into k further buckets is performed, and so on. The maximum size of every bucket formed in any of the m iterations is shown to be $(1 + O(1/\omega_n))\,(n/p)$, for any slowly growing function ω_n. By employing multi-way merging [15] we obtain the desired sorted sequence. Our main result is summarized in the following lemma.

Lemma 3 *For any $\alpha = \Omega(1)$, there exists a p-processor BSP algorithm for sorting n keys that for all integers n and p such that $p = n/\lg^{2+\alpha} n$, requires time $(c_s + 2c_s/\alpha + o(1))\,((n/p)\,\lg n)$ and $L\,O((\lg n/\lg(n/p))\,\lg^2 p) + g\,O((\lg n/\lg(n/p))\,(n/p))$, for computation and communication (including synchronization) respectively.*

[1] In general, if S is any statement that can be true or false, the bracketed notation [S] evaluates to 1 if S is true, and 0 otherwise.

The proof of the aforementioned result is postponed until the end of this section. The remainder of this section is devoted to the proof of the following proposition and its corresponding corollaries. Then, lemma 3 follows easily as a consequence of these results.

Proposition 1 *There exists a p-processor BSP algorithm for sorting n keys, that for any integers n, p, m and r, and any $\bar{m}$ such that $p \leq n$, $1 \leq \bar{m} < \lg p$, $1 \leq m \leq \lceil \bar{m} \rceil$ and $0 \leq r \leq m$, and any function ω_n of n such that $\omega_n = \Omega(1)$, requires time,*

$$c_s \, \tfrac{n}{p} \lg \tfrac{n}{p} + n_{max} \left(c_r \left(\tfrac{r\,(\bar{m}-m)}{\bar{m}} + \tfrac{r\,(r+1)}{2\bar{m}} \right) \lg p + c_s \,(m-r) \lg \tfrac{n}{p} \right) + \\ m \, O\left(n_{max} + \omega_n \, m \, p^{1/\bar{m}} \lg^2 p \right) + m \, C_{ppf}^{\lceil p^{1/\bar{m}} \rceil}(p)$$

and

$$m \, \left(\max\{L, g\, n_{max}\} + O\left(\lg^2 p\right) \, \max\left\{L, g\, O\left(\omega_n \, m \, p^{1/\bar{m}}\right)\right\} \right) + \\ m \left(M_{brd}^{\lceil p^{1/\bar{m}} \rceil - 1}(p) + M_{ppf}^{\lceil p^{1/\bar{m}} \rceil}(p) \right),$$

for computation and communication respectively, where $n_{max} = (1 + O(1/\omega_n))\,(n/p)$, provided that $\omega_n^2 \, m^2 \, p^{1/\bar{m}} = O(n/p)$ is also satisfied.

Proof: The algorithm is recursive in nature and each recursive call is divided into a sequence of stages. We assume that the input is evenly but otherwise arbitrarily 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 their memory location. We later explain how we can remove such an assumption and still maintain optimality.

Algorithm Sort

begin

Sequential_Sort

Recursive_Sort $(0, p-1)$

end

Procedure Sequential_Sort

begin

Each processor sorts the keys in its possession.

Analysis When this step is executed, each processor holds at most $\lceil n/p \rceil$ keys. Thus, the cost of this stage is that of sequentially sorting $\lceil n/p \rceil$ keys by some deterministic algorithm [15] and requires time $c_s \lceil n/p \rceil \lg \lceil n/p \rceil$.

end

In procedure Recursive_Sort below the variables l and u have initially values 0 and $p-1$ respectively; in any recursive call of the algorithm, these two variables give the range of the processors that participate in the call.

Procedure Recursive_Sort (l, u)

begin

Stage 1 Each processor selects locally $\lceil \omega_n \rceil \, m \, k - 1$ evenly spaced sample keys, that partition its input into $\lceil \omega_n \rceil \, m \, k$ evenly sized segments, where $k = \min\{q, \lceil p^{1/\bar{m}} \rceil\}$, $q = u - l + 1$ and $\bar{m}$ satisfies the conditions in the statement of the proposition. Additionally, each processor appends to this sorted list the largest key in its input. Let $s = \lceil \omega_n \rceil \, m \, k$ be the size of the so identified list.

Analysis of Stage 1 Since the keys residing in the local memory of any processor are sorted prior to the sample selection process, this stage requires time $O(s)$.

Stage 2 The q sorted lists, each consisting of s sample keys, are merged; let sequence $\langle s_{ls+1}, s_{ls+2}, \ldots, s_{ls+qs} \rangle$ be the result of the merge operation. By assumption, the sequence is evenly distributed among the q processors, i.e., subsequence $\langle s_{is+1}, \ldots, s_{(i+1)s} \rangle$, $l \leq i \leq u$, resides in the local memory of the i-th processor.

Analysis of Stage 2 The cost of this stage is that of parallel sorting by one of Batcher's methods [3], appropriately modified [4, 15] to handle sorted sequences of size s. The computation and communication time required for this stage is, respectively, $O(s \lg^2 q)$ and $O(\lg^2 q) \max\{L, g\, O(s)\}$.

Stage 3 A broadcast operation is initiated, where splitter $s_{ls+iqs/k}$, $1 \leq i < k$, along with its index in the sequence of sample keys is sent to all processors.

Analysis of Stage 3 The so identified $k-1$ splitter keys are first sent to processor p_l and then broadcast by employing the algorithm of lemma 1. This stage takes time $O(1)$ and $\max\{L, g\, O(k)\} + M_{brd}^{k-1}(p)$ for computation and communication respectively.

Stage 4 (i) Each processor decides the position of every key it holds with respect to the $k-1$ splitters it received in the previous stage, by way of merging, and counts the number of keys that fall into each of the k so identified buckets; (ii) k independent parallel prefix operations are initiated (one for each bucket) 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 4 When step (i) is executed, by way of lemma 4 shown later, each processor possesses $n_{max} = (1 + O(1/\omega_n))\,(n/p)$ keys. Thus, step (i) takes $O(n_{max} + k)$ time. The k disjoint parallel prefix operations in step (ii) are realized by employing the algorithm of lemma 2 in $T_{ppf}^{k}(p)$ time.

Stage 5 Each processor knows where to route its keys, and performs the routing. Processors route keys in "column-major" order, that is, keys destined to some bucket fill the processors assigned to this bucket in linear order, the first processor being determined by the information gathered in step (ii) of stage 4.

Analysis of Stage 5 This stage takes time $\max\{L, g\, O(n_{max})\}$.

Stage 6 Each processor merges the at most q sorted subsequences that it received in the previous stage.

Analysis of Stage 6 When this step is executed, each processor possesses at most $\min\{q, n_{max}\}$ sorted sequences for a total of n_{max} keys. The cost of this stage is that of sequential multi-way merging n_{max} keys by any deterministic algorithm [15], which is by $n_{max} \min\{c_r \lg q, c_s \lg n_{max}\}$. Thus, the cost of this stage over the m iterations of the procedure is bounded above as follows (in the subsequent analysis the j-th recursive call is denoted by the subscript notation $_{(j)}$).

$$n_{max} \sum_{j=1}^{m} \min\{c_r \lg q_{(j)}, c_s \lg n_{max}\} \leq n_{max} \left(c_r \sum_{j=0}^{r-1} \lg q_{(m-j)} + c_s \sum_{j=r}^{m-1} \lg n_{max} \right).$$

The value of r that minimizes the above expression is determined by comparing $q_{(j)}$ and n_{max}. Substituting $(1+O(1/\omega_n))\,(n/p)$ for n_{max} we obtain the following bound on the cost of this stage.

$$n_{max}\,\left(c_r \sum_{j=0}^{r-1} \lg q_{(m-j)} + c_s \sum_{j=r}^{m-1} \lg\left(\left(1+O\left(\frac{1}{\omega_n}\right)\right)\,\frac{n}{p}\right)\right).$$

The following relations are noted: $q_{(1)}=p$, $\lfloor q_{(j-1)}/k_{(j-1)}\rfloor \le q_{(j)} \le \lceil q_{(j-1)}/k_{(j-1)}\rceil$, $k_{(j)}=\lceil p^{1/\bar{m}}\rceil$ if $j\neq m$ and $k_{(m)}=q_{(m)}$. To simplify the exposition, in the remainder of the analysis we assume that $q_{(j-1)}/k_{(j-1)}$ is integer valued; this assumption can be removed without affecting the asymptotic analysis of the algorithm. Hence, we have that $q_{(j)}=p/\lceil p^{1/\bar{m}}\rceil^{j-1}$ if $j\neq m$. We conclude that this stage requires time bounded as follows.

$$n_{max}\,\left(c_r\,\left(\frac{r\,(\bar{m}-m)}{\bar{m}}+\frac{r\,(r+1)}{2\bar{m}}\right)\lg p + c_s\,(m-r)\,\lg\frac{n}{p}+O\,(m)\right).$$

Stage 7 If $l\ge u$, then exit; otherwise, recursively sort each of the k buckets generated in stages 1–6 by assigning to each bucket $\lceil q/k\rceil$ (or $\lfloor q/k\rfloor$) consecutive processors.

Analysis of Stage 7 The depth of the recursion is $m\le\lceil\bar{m}\rceil$.
end

Analysis of the Algorithm The correctness of the algorithm follows from its description and is omitted. By summing up the computation requirements of procedures SEQUENTIAL_SORT and RECURSIVE_SORT we obtain the stated bound on computation. The communication bound follows from the analysis of procedure RECURSIVE_SORT.

Remark In case the (sorted) output sequence needs to be balanced among the processors at the end of the sorting algorithm, a balancing operation is performed in time $O(n_{max})+C^1_{ppf}(p)$ and $\max\{L, g\,O(n_{max})\}+M^1_{ppf}(p)$ for computation and communication respectively. We note that this latter term does not affect the asymptotic complexity of the algorithm. ■

It remains to prove that procedure RECURSIVE_SORT partitions the input into evenly sized buckets. One way to simplify the forthcoming analysis is to assume that $n_{(j)}/(q_{(j)}s_{(j)})$ is an integer, so that the padding operations to be described in the following analysis become unnecessary. We choose instead to give a fully detailed analysis. The main result is summarized in the following lemma.

Lemma 4 *The maximum number of elements n_{max} per processor in any of the m iterations of procedure* RECURSIVE_SORT *is $(1+O(1/\omega_n))\,(n/p)$, for any ω_n such that $\omega_n=\Omega(1)$, provided that $\omega_n^2\,m^2\,p^{1/\bar{m}}=O(n/p)$ is satisfied.*

Proof: Although it is not explicitly mentioned in the description of algorithm RECURSIVE_SORT we assume that we initially 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 can be such a choice). Before performing stage 1 we also pad the input so that after stage 1 all segments have the same number of keys that is, $x=\lceil\lceil n/p\rceil/s\rceil$. The padding operation requires time at most $O(s)$ per iteration, which is within the lower order terms of the analysis of proposition 1, and therefore, does not affect the asymptotic complexity of the algorithm. We note that padding operations introduce duplicate keys. We have shown how to handle duplicate keys in the beginning of the section. A more refined approach is to be shown in a later section.

Refer to the j-th recursive call of stage 3 of procedure RECURSIVE_SORT and consider an arbitrary splitter $s_{l s_{(j)}+(i q_{(j)} s_{(j)})/k_{(j)}}$, where $1 \leq i < k_{(j)}$. There are at least $((i\, q_{(j)}\, s_{(j)})/k_{(j)})\, x$ keys which are not larger than $s_{l s_{(j)}+(i q_{(j)} s_{(j)})/k_{(j)}}$, since there are $(i\, q_{(j)}\, s_{(j)})/k_{(j)}$ segments each of size x whose keys are not larger than $s_{l s_{(j)}+(i q_{(j)} s_{(j)})/k_{(j)}}$. Likewise, there are at least $(q_{(j)}\, s_{(j)} - (i\, q_{(j)}\, s_{(j)})/k_{(j)} - q_{(j)} + 1)\, x$ keys which are not smaller than $s_{l s_{(j)}+(i q_{(j)} s_{(j)})/k_{(j)}}$, since there are $q_{(j)}\, s_{(j)} - (i\, q_{(j)}\, s_{(j)})/k_{(j)} - q_{(j)} + 1$ segments each of size x whose keys are not smaller than $s_{l s_{(j)}+(i q_{(j)} s_{(j)})/k_{(j)}}$. Thus, by noting that the total number of keys has been increased (by way of padding operations) from $n_{(j-1)}$ to $q_{(j)}\, s_{(j)}\, x$, the number of keys $b^i_{(j)}$ that are smaller than $s_{l s_{(j)}+(i q_{(j)} s_{(j)})/k_{(j)}}$ is bounded as follows.

$$\frac{i\, q_{(j)}\, s_{(j)}}{k_{(j)}}\, x \leq b^i_{(j)} \leq q_{(j)}\, s_{(j)}\, x - \left(q_{(j)}\, s_{(j)} - \frac{i\, q_{(j)}\, s_{(j)}}{k_{(j)}} - q_{(j)} + 1 \right) x.$$

A similar bound can be obtained for $b^{i+1}_{(j)}$. Substituting $s_{(j)} = \lceil \omega_n \rceil\, m\, k_{(j)}$ we therefore conclude the following.

$$b^{i+1}_{(j)} - b^i_{(j)} \leq \frac{q_{(j)}\, s_{(j)}}{k_{(j)}}\, x + q_{(j)}\, x = \lceil \omega_n \rceil\; m\, q_{(j)}\, x + q_{(j)}\, x.$$

The difference $n_{(j)} = b^{i+1}_{(j)} - b^i_{(j)}$ is independent of i and gives the maximum number of keys per bucket in the j-th iteration. Considering that $x \leq (n_{(j-1)} + q_{(j)} + q_{(j)}\, s_{(j)})/(q_{(j)}\, s_{(j)})$ and substituting $s_{(j)} = \lceil \omega_n \rceil\, m\, k_{(j)}$, the following bound is derived.

$$n_{(j)} \leq \left(1 + \frac{1}{\lceil \omega_n \rceil\; m}\right) \frac{n_{(j-1)} + q_{(j)}\, (s_{(j)} + 1)}{k_{(j)}}.$$

Solving the recurrence relation, we obtain the following upper bound on $n_{(j)}$.

$$n_{(j)} \leq \left(1 + \frac{1}{\lceil \omega_n \rceil\, m}\right)^j \frac{n_{(0)} + \sum_{i=1}^{j} \left((1 + 1/(\lceil \omega_n \rceil\, m))^{-i} q_{(i)}\, (s_{(i)}+1)\, \prod_{l=1}^{i-1} k_{(l)} \right)}{\prod_{i=1}^{j} k_{(i)}}.$$

It is first observed that $(1 + 1/(\lceil \omega_n \rceil\, m))^m = (1 + O(1/\omega_n))$ and by way of the analysis and assumptions of stage 6 that $\prod_j k_{(j)} = p$. By substituting in the previous expression $s_{(j)} \leq \lceil \omega_n \rceil\, m\, \lceil p^{1/\bar{m}} \rceil$, $n_{(0)} = n$ and $q_{(1)} = p$, we conclude that the maximum number of elements n_{max} per processor in any of the m iterations of procedure RECURSIVE_SORT is bounded above as follows.

$$n_{max} \leq \left(1 + O\left(\frac{1}{\omega_n}\right)\right) \frac{n}{p} + O\left(\omega_n\, m^2\, p^{1/\bar{m}}\right).$$

The lemma follows. ∎

The following theorem is derived from proposition 1 and establishes concrete bounds on the parameters π and μ.

Theorem 1 *There exists a p-processor BSP algorithm for sorting n keys, that for any integers p, m and r, and any $\bar{m}$ and t such that $p = n^{1-t}$, $1 \leq \bar{m} < \lg p$, $1 \leq m \leq \lceil \bar{m} \rceil$, $0 \leq r \leq m$, and $0 < t < 1$, and any function ω_n of n such that $\omega_n = \Omega(1)$, has,*

$$\pi = (1 + (m - r))\; t + \frac{c_r}{c_s} \left(\frac{r\, (\bar{m} - m)}{\bar{m}} + \frac{r\, (r+1)}{2\bar{m}} \right) (1 - t) + O\left(\frac{1}{\omega_n}\right)$$

and

$$\mu = g\,O\left(\frac{\omega_n\, m^2\, \lg n}{n^{t+(1/\bar{m})(t-1)}}\,(1-t)^2 + \frac{m}{\lg n}\right) + O\left(\frac{1}{\omega_n}\right),$$

provided that $L = O((n/p)\,(\lg n/\ (\omega_n\, m\, \lg^2 p)))$, $m = O(\lg n/\omega_n)$, $\omega_n^2\, m^2\, p^{1/\bar{m}} = O(n/p)$ *and* $\omega_n^2\, m^2\, p^{1/\bar{m}}\, \lg^2 p = O((n/p)\ \lg n)$ *are satisfied.*

Proof: Let $p = n^{1-t}$, for any $0 < t < 1$. We first consider the computation time of the sorting algorithm. We employ for the broadcasting and parallel prefix operations the algorithms implied in lemmas 1 and 2. Substituting in proposition 1 $m = O(\lg n/\omega_n)$, $\omega_n^2\, m^2\, p^{1/\bar{m}} = O(n/p)$, $\omega_n^2\, m^2\, p^{1/\bar{m}}\, \lg^2 p = O((n/p)\ \lg n)$ and $n_{max} = (1 + O(1/\omega_n))\,(n/p)$ we obtain the following bound on the computation time.

$$c_s\ (1 + (m-r))\ \frac{n}{p}\,\lg\frac{n}{p} + c_r\left(\frac{r\,(\bar{m}-m)}{\bar{m}} + \frac{r\,(r+1)}{2\bar{m}}\right)\frac{n}{p}\,\lg p + O\left(\frac{n \lg n}{\omega_n p}\right).$$

The desired expression for π in the statement of the theorem 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, r, $\bar{m}$ and ω_n with regard to the actual values of n, p, L and g. By making, however, more generic selections for the underlying parameters and worst-case bounds for p, L and g, more general results in terms of n can be obtained. Our main claims assume that $c_s = c_r$ (the claims can be easily extended to accommodate the case $c_s > c_r$). The case $c_s < c_r$ is treated separately in the remark following corollary 3.

Corollary 1 *Let ω_n be any function of n such that $\omega_n = \Omega(1)$ and $\omega_n = O(\lg n)$. Then, for any constant ϵ, $1/2 < \epsilon < 1$, there exists a p-processor BSP algorithm for sorting n keys that for all integers n and p such that $p = n^{1-\epsilon}$, has $\pi = 1 + O(1/\omega_n)$, and if $L = O((n/p)/(\omega_n\ \lg n))$ and $g = O(\lg n/\omega_n)$, then $\mu = O(1/\omega_n)$.*

Proof: In theorem 1 let $t = \epsilon$ and $r = m = \bar{m} = 1$. Then, the conditions of theorem 1 are satisfied and we obtain the following bound on π, for $c_s = c_r$.

$$\pi = t + (1-t) + O\left(\frac{1}{\omega_n}\right) = 1 + O\left(\frac{1}{\omega_n}\right).$$

For the communication time, if $L = O((n/p)/(\omega_n\ \lg n))$ and $g = O(\lg n/\omega_n)$, then we also have $\mu = O(1/\omega_n)$. ■

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

Corollary 2 *Let ω_n be any function of n such that $\omega_n = \Omega(1)$ and $\omega_n = O(\lg n/\lg\lg n)$. Then, for any arbitrarily small constant ζ such that $0 < \zeta < 1/2$ and any constant ϵ such that $0 < \epsilon \leq 1/2 + \zeta$, there exists a p-processor BSP algorithm for sorting n keys that for all integers n and p such that $p = n^{1-\epsilon}$, has $\pi = 1 + O(1/\omega_n)$, and if $L = O((n/p)/(\omega_n\ \lg n))$ and $g = O(\lg n/\omega_n)$, then $\mu = O(1/\omega_n)$.*

Proof: In theorem 1 let $t = \epsilon$, $\bar{m} = (1-t)/(t-\delta)$, $m \leq \lceil \bar{m} \rceil$ and $r = 1$, where $\delta = (\lg\lg n + 2\lg\omega_n)/\lg n$. Then, the conditions of theorem 1 are satisfied and we obtain the following bound on π, for $c_s = c_r$.

$$\begin{aligned}
\pi &= m\,t + \left(1 - \frac{(m-1)\,(t-\delta)}{1-t}\right)(1-t) + O\left(\frac{1}{\omega_n}\right) \\
&= 1 + (m-1)\,\delta + O\left(\frac{1}{\omega_n}\right) \\
&= 1 + O\left(\frac{1}{\omega_n}\right).
\end{aligned}$$

For the communication time, if $L = O((n/p)/(\omega_n \lg n))$ and $g = O(\lg n/\omega_n)$, then we also have $\mu = O(1/\omega_n)$. ∎

When only poly-logarithmic slack is available the following corollary can be derived.

Corollary 3 *Let ω_n be any function of n such that $\omega_n = \Omega(1)$ and $\omega_n = O((\alpha \lg\lg n)/\lg(1+\alpha))$. For any α, $\alpha = \Omega(1)$ and $\alpha = o(\lg n/\lg\lg n)$, there exists a p-processor BSP algorithm for sorting n keys that for all integers n and p such that $p = n/\lg^{3+\alpha} n$, has $\pi = 1 + 3/\alpha + O(1/\omega_n)$, and if $L = O((n/p)\,(\alpha \lg\lg n)/(\omega_n \lg^2 n))$ and $g = O((\alpha \lg\lg n)/\omega_n)$, then $\mu = O(1/\omega_n)$.*

Proof: In theorem 1 let $t = (3+\alpha)\,(\lg\lg n/\lg n)$, $\bar{m} = (1-t)/(t-\delta)$, $m \leq \lceil \bar{m} \rceil$ and $r = 1$, where $\delta = (3\lg\lg n + 2\lg(\omega_n/(\alpha \lg\lg n)))/\lg n$. Then, the conditions of theorem 1 are satisfied and we obtain the following bound on π, for $c_s = c_r$.

$$\begin{aligned}
\pi &= m\,t + \left(1 - \frac{(m-1)\,(t-\delta)}{1-t}\right)(1-t) + O\left(\frac{1}{\omega_n}\right) \\
&= 1 + (m-1)\,\delta + O\left(\frac{1}{\omega_n}\right) \\
&= 1 + O\left(\frac{1}{\omega_n}\right).
\end{aligned}$$

For the communication time, if $L = O((n/p)\,((\alpha \lg\lg n)/(\omega_n \lg^2 n)))$ and $g = O((\alpha \lg\lg n)/\omega_n)$, then we also have $\mu = O(1/\omega_n)$. ∎

Remark In the aforementioned analysis, if we substitute $r = 0$, then we obtain $\pi = 1 + 3/\alpha + O((\alpha \lg\lg n)/\lg n + 1/\omega_n)$ and $\mu = O(1/\omega_n)$ for *any* constant c_r, and thus, corollaries 1– 3 can be extended in a straightforward way to handle the case $c_r > c_s$.

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 can be performed by either using our algorithm recursively or by some completely different algorithm. In particular, since the set of splitters per processor is already sorted we may use a merging algorithm for sorting the splitters, i.e., q sorted sequences, each of size s and one per processor, are merged. We perform this merging by using a binary tree execution sequence of depth $\lceil \lg q \rceil$, where in the i-th iteration pairs of subsequences 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 [10]. The latter algorithm derives from [20] and uses the idea of deterministic oversampling. The algorithm runs in time $O(s) + O(C^1_{ppf}(q))$ and $O(\max\{L, g\,O(s)\}) + O(M^1_{ppf}(q)) + O(M^s_{brd}(q))$ for computation and communication respectively. For the case $s = \Omega(\lg q)$, the computation and communication time (with respect to g) of stage 2 of RECURSIVE_SORT is reduced by a multiplicative factor of $O(\lg p)$. Therefore, the condition $\omega_n^2\, m^2\, p^{1/\bar{m}}\, \lg^2 p = O((n/p)\, \lg n)$ of theorem 1 is relaxed to $\omega_n^2\, m^2\, p^{1/\bar{m}}\, \lg p = O((n/p)\, \lg n)$. We thus obtain the following analogue of corollary 3.

Corollary 4 *Let ω_n be any function of n such that $\omega_n = \Omega(1)$ and $\omega_n = O((\alpha\, \lg\lg n)/\lg(1+\alpha))$. For any α, $\alpha = \Omega(1)$ and $\alpha = o(\lg n/\lg\lg n)$, there exists a p-processor BSP algorithm for sorting n keys that for all integers n and p such that $p = n/\lg^{2+\alpha} n$, has $\pi = 1 + 2/\alpha + O(1/\omega_n)$, and if $L = O((n/p)\,(\alpha\, \lg\lg n)/(\omega_n\, \lg^2 n))$ and $g = O((\alpha\, \lg\lg n)/\omega_n)$, then $\mu = O(1/\omega_n)$.*

Proof:(of lemma 3) It is a combination of corollaries 1, 2 and 4. ■

6 Duplicate Keys

The algorithm, as described, does not handle duplicate keys. One way to allow duplicates is by making keys distinct. This can be achieved by attaching to each key the address of the memory location it is stored in. For data types whose bit or byte length is comparable to that of their memory location, such a transformation leads – 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 outlined in the following discussion. Procedure SEQUENTIAL_SORT is implemented by means of some stable sequential sorting algorithm. In procedure RECURSIVE_SORT all keys are appended the address of their memory location. The attached information is only used in stages 2–4 (note that these stages require distinct keys to achieve stability and load-balance). The computational and communication cost of these stages is within the lower order terms of the analysis of corollaries 1–4, and therefore, the optimality claims still hold.

7 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.

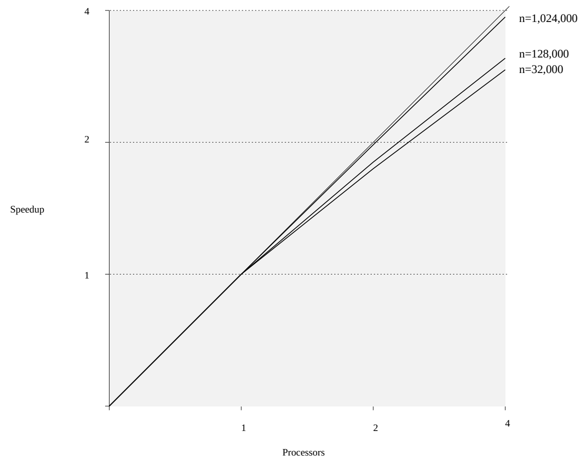

Figure 1: Speedup achieved by deterministic sorting on a SGI Power Challenge system.

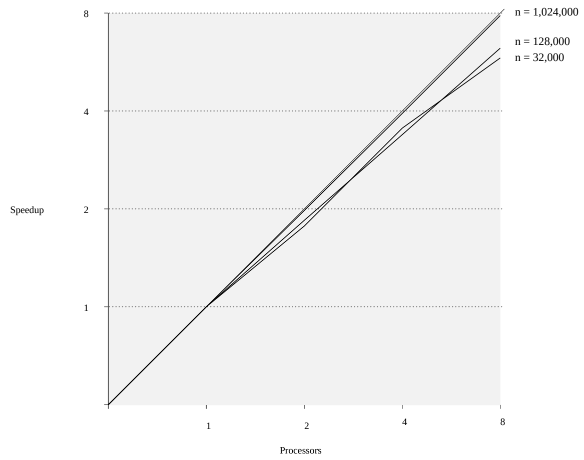

Figure 2: Speedup achieved by deterministic sorting on an IBM 9076 SP1 system.

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] M. Ajtai, J. Komlós, and E. Szemerédi. Sorting in $c \log n$ steps. *Combinatorica*, 3:1-19, 1983.

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

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

[5] G. Bilardi, K. T. Herley, A. Pietracaprina, G. Pucci, and P. Spirakis. BSP vs. LogP. In *Proceedings of the 8-th ACM Symposium on Parallel Algorithms and Architectures*, pp. 25-32, 1996.

[6] R. Cole. Parallel merge sort. *SIAM Journal on Computing,* 17(4):770-785, 1988.

[7] R. Cypher and J. Sanz. Cubesort: A parallel algorithm for sorting n-data items with s-sorters. *Journal of Algorithms*, 13;211-234, 1992.

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

[9] 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*, pp. 223-232, 1996.

[10] A. V. Gerbessiotis and C. J. Siniolakis. Primitive operations on the BSP model. Technical Report PRG-TR-23-96, Computing Laboratory, Oxford University, 1996.

[11] A. V. Gerbessiotis and C. J. Siniolakis. An experimental study of sorting algorithms on the BSP model: Theory, practice and experience. Technical Report PRG-TR-06-97, Computing Laboratory, Oxford University, 1997.

[12] A. V. Gerbessiotis and C. J. Siniolakis. A randomized sorting algorithm on the BSP model. In Proceedings of IPPS, 1997.

[13] M. T. Goodrich. Communication-efficient parallel sorting. *Proceedings 26th ACM STOC*, 1996.

[14] 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.

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

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

[17] W. F. McColl. General purpose parallel computing. In *Lectures on Parallel Computation,* (A. Gibbons, and P. Spirakis, eds.), Cambridge University Press, 1993.

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

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

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

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

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