Programming Research Group

A NOTE ON PROBABILISTIC INTEGER SORTING

Alexandros V. Gerbessiotis and Constantinos J. Siniolakis

PRG-TR-14-97

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

A Note on Probabilistic Integer Sorting*

Alexandros V. Gerbessiotis and Constantinos J. Siniolakis

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

April 1997

Abstract

We present a new probabilistic sequential algorithm for stable sorting n uniformly distributed keys in an arbitrary range. The algorithm runs in linear time with very-high probability $1 - 2^{-\Omega(n)}$ (the best previously known probability bound has been $1 - 2^{-\Omega(n/(\lg n \lg\lg n))}$). We also describe an EREW PRAM version of the algorithm that sorts in $O((n/p + \lg p)\lg n / \lg(n/p + \lg n))$ time using $p \leq n$ processors within the same probability bound. Additionally, we present experimental results for the sequential algorithm that establish the practicality of our algorithm.

Keywords: Integer sorting - parallel integer sorting - probabilistic analysis of algorithms.

*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.

1 Introduction

In this work we focus our attention on probabilistic sequential and parallel algorithms for stable sorting uniformly distributed integers in an arbitrary range. This is an important special case of integer sorting with a significant number of applications [8, 9]. The parallel algorithm we describe works on the EREW PRAM [5].

2 Related Work

Several probabilistic algorithms have been proposed in the past for the problem of sorting uniformly distributed integer keys. In [9], an algorithm is presented that sorts in $O(n)$ expected-time, and the same time bound is obtained in [10]. An improvement of the result in [10] is presented in [7], that yields an integer sorting algorithm that runs in $O(n)$ time with probability at least $1 - c^{-\sqrt{n}}$, for any constant $c > 0$. A PRAM CRCW version of this algorithm is also described that sorts in $O(\lg n)$ time using $n/\lg n$ processors with the same probability bound. In an independent work, Chlebus [3], by using a more complicated algorithm realizes the same time bound as in [7], but achieves the probability bound of $1 - 2^{-cn/\lg^3 n}$. Finally, in [12] a CRCW randomized version of [9] is presented that sorts in $O(\lg n)$ time using $n/\lg n$ processors with probability at least $1 - n^{-c}$. The sequential version of the latter algorithm improves, however, the probability bounds of previous approaches to $1 - 2^{-cn/(\lg n \lg\lg n)}$. We note that the parallel versions of the algorithms in [3, 7] are non-stable.

In this paper we present a sequential integer stable sorting algorithm that achieves the improved probability bound $1 - 2^{-cn}$, for any constant $c > 0$. An immediate extension of this algorithm to the EREW PRAM model sorts n keys with p processors on an EREW PRAM in time $O((n/p + \lg p)\lg n/\lg(n/p + \lg p))$ with the same probability bound. We also present experimental results that compare the performance of our algorithm to quicksort, radix-sort, and hybrid-sort [7, 8, 10].

3 A Combinatorial Lemma

This section is devoted to the proof of a technical combinatorial lemma. We start by presenting some well-known probabilistic bounds. Let $X_1, \ldots, X_n$ be a sequence of n independent Bernoulli random variables each with mean P, and let $\log x$, $\lg x$ be the logarithm of x to the base e and 2 respectively. Then, the random variable $X_{n,P} = \sum_{i=1}^{n} X_i$ follows a binomial distribution. Let $H(x)$, $0 < x < 1$, be the weighted entropy function:

$$H(uP) = uP \log \frac{1}{u} + (1 - uP) \log \frac{1 - P}{1 - uP}.$$

Then, for any $u \geq 1$, the following holds [1, page 11].

$$\mathrm{Prob}[X_{n,P} \geq unP] \leq e^{nH(uP)}.$$

We therefore obtain the following bound for the probability on the tail of the binomial distribution [2]. For any $u \geq 1$ and $0 < uP < 1$,

$$\begin{aligned}
\mathrm{Prob}[X_{n,P} \geq unP] &\leq e^{nH(uP)} \\
&= u^{-unP}\left(\frac{1-P}{1-uP}\right)^{n(1-uP)} \\
&= u^{-unP}\left(1+\frac{(u-1)P}{1-uP}\right)^{n(1-uP)} \\
&\leq u^{-unP}e^{(u-1)Pn} \\
&= \left(\frac{e}{u}\right)^{unP}e^{-nP} \\
&< \left(\frac{e}{u}\right)^{unP}. \qquad (1)
\end{aligned}$$

We also use the following result that derives from Stirling's formula [1].

$$\binom{n}{m} \leq \left(\frac{ne}{m}\right)^m, \qquad (2)$$

for all integers n, m such that $n \geq m$.

We are now ready to present the main result of this section.

Lemma 3.1 *For any positive constants d and β, if k balls are independently placed into $k^{1+\beta}$ bins with probability at most $d/k^{1+\beta}$ of a ball being placed in a fixed bin, then there are at most $Bk/\lg k$ balls into bins of size greater than or equal to D with probability at least $1-2^{-ck+1}$, for any positive constants B, c and D such that $0 < c < \beta(B - 2B/D - B/(\beta D))$.*

Remark. We assume that k is large enough to ensure that $(eD/B)(ed/D)^D \log k \leq k^\beta$ and $d < k^\beta$.

Proof : For any positive constant B, denote by $\mathcal{E}$ the event that there are at least $\hat{B} = Bk/(D\lg k)$ bins of size greater than or equal to D. By definition, the probability of a ball being placed in a fixed bin is at most $d/k^{1+\beta}$. Then, by a combinatorial argument [7], we bound $\mathrm{Prob}[\mathcal{E}]$ as follows.

$$\begin{aligned}
\mathrm{Prob}[\mathcal{E}] &\leq \binom{k^{1+\beta}}{\hat{B}} \sum_{a_1\geq D, a_2\geq D, \ldots, a_{\hat{B}}\geq D} \binom{k}{a_1}\cdots\binom{k-\sum_{j=1}^{\hat{B}-1}a_j}{a_{\hat{B}}}\left(\frac{d}{k^{1+\beta}}\right)^{a_1+\ldots+a_{\hat{B}}} \\
&\leq \binom{k^{1+\beta}}{\hat{B}} \sum_{a_1\geq D, a_2\geq D, \ldots, a_{\hat{B}}\geq D} \binom{k}{a_1}\cdots\binom{k}{a_{\hat{B}}}\left(\frac{d}{k^{1+\beta}}\right)^{a_1+\ldots+a_{\hat{B}}} \\
&\leq \binom{k^{1+\beta}}{\hat{B}} \sum_{a_1\geq D, a_2\geq D, \ldots, a_{\hat{B}}\geq D} \left(\frac{ek}{D}\,\frac{d}{k^{1+\beta}}\right)^{D\hat{B}}
\end{aligned}$$

$$
\begin{aligned}
&\leq \binom{k^{1+\beta}}{\hat{B}} k^{\hat{B}} \left(\frac{ek}{D}\frac{d}{k^{1+\beta}}\right)^{D\hat{B}} \\
&\leq \left(\frac{eD}{B} k^{1+\beta} \lg k \left(\frac{ek}{D}\frac{d}{k^{1+\beta}}\right)^{D}\right)^{\hat{B}} \\
&\leq \left(\frac{eD}{B} k^{1+\beta} \lg k \left(\frac{ed}{Dk^{\beta}}\right)^{D}\right)^{\hat{B}}.
\end{aligned}
$$

By assumption, k is large enough to ensure that $(eD/B)(ed/D)^D \log k \leq k^\beta$. Therefore, for any $0 < c < \beta(B - 2B/D - B/(\beta D))$ the following bound is derived.

$$
\begin{aligned}
\text{Prob}[\mathcal{E}] &\leq \left(k^{1+2\beta}\frac{1}{k^{\beta D}}\right)^{\hat{B}} \\
&\leq \left(k^{1+2\beta-\beta D}\right)^{\hat{B}} \\
&\leq 2^{(1+2\beta-\beta D)Bk/D} \\
&\leq 2^{-ck}.
\end{aligned}
$$

We have therefore established that the probability that there are at least $\hat{B}$ bins of size greater than or equal to D is at most 2^{-ck} for some appropriately bounded positive constant c. Denote by $\mathcal{Z}$ the event that the total number of balls falling into these $\hat{B}$ bins is at least $Bk/\lg k$. Consider any fixed set of $\hat{B}$ bins out of the total $k^{1+\beta}$ bins. By definition, placing a ball into a bin is independent of all other placements. Therefore, the total number of balls falling into these bins is stochastically dominated by the binomially distributed random variable $X_{n,P} = X_{k,dB/(Dk^\beta \lg k)}$. By employing the probability bound of equation 1 on the tail of the binomial distribution the following upper bound on Prob$[\mathcal{Z}]$ is derived.

$$
\begin{aligned}
\text{Prob}[\mathcal{Z}] &< \binom{k^{1+\beta}}{\hat{B}} \left(\frac{ed}{Dk^{\beta}}\right)^{Bk/\lg k} \\
&\leq \left(\frac{eD}{B}\left(\frac{ed}{Dk^{\beta}}\right)^{D} k^{\beta} \lg k\right)^{Bk/(D\lg k)}.
\end{aligned}
$$

By assumption, k is large enough to ensure that $(eD/B)(ed/D)^D \log k \leq k^\beta$. Therefore, the following bound holds for any $0 < \hat{c} < \beta(B - 2B/D)$.

$$
\begin{aligned}
\text{Prob}[\mathcal{Z}] &\leq k^{-\beta(D-2)(Bk/(D\lg k))} \\
&\leq 2^{-\hat{c}k}.
\end{aligned}
$$

We have established that the probability that at least $Bk/\lg k$ balls fall into some fixed set of $Bk/(D\lg k)$ bins is at most $2^{-\hat{c}k}$. We have also shown that the probability that there are at least $Bk/(D\lg k)$ bins of size greater than or equal to D is at most 2^{-ck}. Therefore, for any $0 < \hat{c} = c < \beta(B - 2B/D - B/(\beta D))$ and $D > (1+2\beta)/\beta$ there are at most $Bk/\lg k$ balls into bins of size greater than or equal to D with probability at least $1 - 2^{-ck+1}$, and the lemma follows. ■

4 An Improved Analysis of R-S-Sort

In this section we show how to modify the $O(n)$ expected-time Radix-plus-Sifting (R-S-Sort) algorithm of [9] to obtain a run-time of $O(n)$ with probability at least $1-2^{-cn+1}$, for any constant $c > 0$, under the assumption that the n input keys are uniformly distributed integers in an arbitrary range. We address in fact a slightly more general problem; its solution will provide us with an integer sorting algorithm. The main algorithmic techniques utilized in the design of our algorithm derive from the methods of [7, 9, 10, 12].

The basic idea behind the algorithm in [9] is to sort the keys with respect to some reduced number of Most Significant Bits (MSBs). As noted in [12] this nearly sorts the input. Suppose for example that we have sorted the n integer keys with respect to their $\lceil \lg n \rceil$ MSBs. For $i \in \{1, \ldots, 2^{\lceil \lg n \rceil}\}$, let S_i be the (possibly empty) sequence of keys with value i for their $\lceil \lg n \rceil$ MSBs. Then, the original sorting problem reduces to $2^{\lceil \lg n \rceil}$ disjoint subproblems, namely the problems of sorting $S_1, \ldots, S_{2^{\lceil \lg n \rceil}}$. In particular, the required sorted sequence is $S_1^* \circ \ldots \circ S_{2^{\lceil \lg n \rceil}}^*$, where S_i^* is the sequence of keys in S_i in sorted order and $\circ$ denotes concatenation. Therefore, the total time required for the algorithm is $O(n) + \sum_{i \in \{1,\ldots,2^{\lceil \lg n \rceil}\}} O(|S_i| \lg |S_i|)$, which is known to be $O(n)$ on the average [9].

Our algorithms work for a universe U equipped with a total order $<$ that can be evaluated in constant time by one processor for any given pair of arguments. Let U^* be the set of ordered sequences $x_1, \ldots, x_m \in U$ with $x_1 < \ldots < x_m$. We assume keys of U and U^* to be represented by means of arrays in the obvious way. Given $X = x_1, \ldots, x_m \in U^*$ and $y \in U$, we define the *rank* of y in X, denoted by $\text{rank}(y : X)$, to be the cardinality of the set $\{x_i \mid i \in \{1, \ldots, m\} \text{ and } x_i \leq y\}$. Similarly, given $X = x_1, \ldots, x_m \in U$, we define the *position* of x_i in X, denoted by $\text{pos}(x_i : X)$, to be i. The remainder of this section is devoted to a proof of the following theorem.

Theorem 4.1 *Let U be a universe equipped with a total order $<$ that can be evaluated in constant time for any given pair of arguments. For any constant $\beta > 0$, let $f : U \to \{1, \ldots, \lceil n^{1+\beta} \rceil\}$ be a non-decreasing function that can be evaluated in constant time, and suppose that $X_1, \ldots, X_n$ are random variables such that for some positive constant d,*

$$Prob[f(X_i) = k] \leq \frac{d}{\lceil n^{1+\beta} \rceil}, \tag{3}$$

for all $i \in \{1, \ldots, n\}$ and $k \in \{1, \ldots, \lceil n^{1+\beta} \rceil\}$. Then, for any constant $c > 0$, there exists a sequential algorithm for stable sorting the sequence (of values) $X_1, \ldots, X_n$ in time $O(n)$ with probability at least $1 - 2^{-cn+1}$.

Remark. Suppose for example that $X_1, \ldots, X_n$ are independent and uniformly distributed over $[1, \ldots, \lceil n^\alpha \rceil]$, $\alpha \geq 1 + \beta$. Then, as noted in [7], inequality 3 is satisfied with:

$$f(x) = \begin{cases} 1 & \text{if } x < 1, \\ \left\lfloor (x-1) \left\lceil \frac{\lceil n^{\alpha} \rceil}{\lceil n^{1+\beta} \rceil} \right\rceil^{-1} \right\rfloor + 1 & \text{if } 1 \le x \le \lceil n^{\alpha} \rceil, \\ n & \text{if } x > \lceil n^{\alpha} \rceil, \end{cases}$$

and $\mathrm{Prob}[f(X_i) = k] \le 2/\lceil n^{1+\beta} \rceil$ for all $i \in \{1, \ldots, n\}$ and $k \in \{1, \ldots, \lceil n^{1+\beta} \rceil\}$. Notice that the above function can be efficiently implemented by simply "extracting" from the x_i's an appropriate number of MSBs.

Proof of theorem 4.1: We show how to modify the R-S-Sort algorithm in order to obtain $O(n)$ run-time with probability at least $1 - 2^{-cn+1}$, for any constant $c > 0$. The input is initially stored in array $S = \langle x_1, x_2, \ldots, x_n \rangle$. After termination of the algorithm the sorted sequence S is assumed to reside in an auxiliary array S^*.

Program R-S-Sort

begin
Stage 1 :
For all $i \in \{1, \ldots, n\}$, compute $y_i = f(x_i)$, while retaining for each key y_i the information pertaining to key x_i.
[**Analysis of Stage 1:** Since by definition, $f(x_i)$, for any $x_i \in U$, can be evaluated in constant time, this stage requires $O(n)$ time.]

Stage 2 :
Sort the sequence (array) $\langle y_1, \ldots, y_n \rangle$ using a stable radix-sort algorithm, where $y_i \in \{1, \ldots, \lceil n^{1+\beta} \rceil\}$, for any i such that $1 \le i \le n$, is viewed as a $(1+\beta)$-digit number with digits in the range $[1, \ldots, n]$.
[**Analysis of Stage 2:** A stable radix-sort algorithm can be found in [8]. When this stage is executed, y_i, for any i such that $1 \le i \le n$, belongs to the range $[1, \ldots, \lceil n^{1+\beta} \rceil]$, and, therefore, this stage requires, by way of [8], time $O((\lg n^{1+\beta} / \lg n) n) = O((1+\beta)n)$.]

Remark. For all $i \in \{1, \ldots, \lceil n^{1+\beta} \rceil\}$, let S_i be the (possibly empty) sequence $S_i = \langle x_j \mid j \in \{1, \ldots, n\} \text{ and } f(x_j) = i \rangle$, and let $S = S_1 \circ \ldots \circ S_{\lceil n^{1+\beta} \rceil} = \langle x_{w_1}, \ldots, x_{w_n} \rangle$ be the concatenation of the S_i's. Then, the original sorting problem reduces to solving $l \le n$ disjoint and non-empty subproblems, namely, the problems of sorting $S_{u_1}, \ldots, S_{u_l}$ such that S_{u_i} is non-empty, where $u_i \in \{1, \ldots, n^{1+\beta}\}$, for any i such that $1 \le i \le l$. In particular, the required sorted sequence S^* is $S^* = S^*_{u_1} \circ \ldots \circ S^*_{u_l}$, where $S^*_{u_i}$ is the sequence of keys in the non-empty S_{u_i} in sorted order.

Stage 3 :
Let D be some positive constant to be specified later. Determine the $l_1 \le l$ non-empty subsequences S_{v_i} satisfying $|S_{v_i}| < D$, with $v_i \in \{1, \ldots, \lceil n^{1+\beta} \rceil\}$ for any i such that $1 \le i \le l_1$, and sort them by some deterministic stable sorter, while retaining for each key $x_j \in S$, for any j such that $1 \le j \le n$, its position in both S and S_{v_i}, i.e., $\mathrm{pos}(x_j : S)$ and $\mathrm{pos}(x_j : S_{v_i})$ respectively.

[**Analysis of Stage 3:** If D is sufficiently small we can use specially tailored sorters to sort the S_{v_i}'s. Otherwise, we may employ any deterministic $O(n \lg n)$ running-time sorter. Both kinds of sorters can be found in [8]. Hence, since the number of non-empty sequences S_{v_i} is $l_1 \leq l$ this stage requires at most $\sum_{i=1}^{l_1} O(|S_{v_i}| \lg |S_{v_i}|) = l_1 \, O(D \lg D) = O(n)$ time, as D is a constant.]

Stage 4 :
For every non-empty subsequence S_{v_i}, for any i such that $1 \leq i \leq l_1$, formed in stage 3, place each key $x_j \in S_{v_i}$ in the output array S^*, at position $\mathrm{pos}(x_j : S) + (\mathrm{rank}(x_j : S_{v_i}) - \mathrm{pos}(x_j : S_{v_i}))$.
[**Analysis of Stage 4:** Since sequence S_{v_i} is in sorted order, the rank of x_j in S_{v_i} can be computed in constant time, and therefore, this stage takes time $O(n)$.]

Stage 5 :
Determine the $l_2 = l - l_1$ non-empty subsequences S_{w_i}, with $w_i \in \{1, \ldots, \lceil n^{1+\beta} \rceil\}$ for any i such that $1 \leq i \leq l_2$, that satisfy $|S_{w_i}| \geq D$, and subsequently place their keys in a contiguous block of memory (array), say, $R = \langle r_1, \ldots, r_{|R|} \rangle$. Sort R by any deterministic $O(n \lg n)$ running-time stable comparison-sort algorithm, while retaining for each key $r_i \in R$ its position in both S and R, i.e., $\mathrm{pos}(r_i : S)$ and $\mathrm{pos}(r_i : R)$ respectively.
[**Analysis of Stage 5:** This stage requires $O(n + |R| \lg |R|)$ time.]

Stage 6 :
For all $i \in \{1, \ldots, |R|\}$, place r_i in the output array S^* at position $\mathrm{pos}(r_i : S) + (\mathrm{rank}(r_i : R) - \mathrm{pos}(r_i : R))$.
[**Analysis of Stage 6:** Since the sequence R is in sorted order, the rank of r_i in R can be computed in constant time, and, therefore, this stage takes time $O(|R|)$.]
end

The overall time complexity of the algorithm can be verified to be $O((1+\beta)n + |R| \lg |R|)$. Therefore, our goal is to show that for any positive constants c and β, there is some positive constant D such that $|R| = O(n / \lg n)$. Substituting in lemma 3.1 for $k = n$ the following is derived for any positive constants B, c and D that satisfy the conditions of lemma 3.1.

$$\mathrm{Prob}\left[|R| \geq B \frac{n}{\lg n}\right] \leq 1 - 2^{-cn+1}.$$

In particular, for $\beta = 3/2$ and $D = 4$ we get the following bound.

$$\mathrm{Prob}\left[|R| \geq 2 \frac{n}{\lg n}\right] \leq 1 - 2^{-n+1}.$$

The theorem follows. ■

5 EREW PRAM *R-S*-Sort

In this section we describe an EREW PRAM version of the R-S-Sort algorithm. The algorithm sorts n keys on a p-processor EREW PRAM, where $p \leq n$, in time $O((n/p +$

$\lg p) \lg n / \lg (n/p + \lg p))$ with probability at least $1 - 2^{-cn+1}$, for any constant $c > 0$. Our analysis makes use of the following EREW adaptation of the stable radix-sort BSP algorithm presented in [6], which derives from – and augments – the PRAM algorithm of [11].

Lemma 5.1 *There exists an EREW PRAM algorithm for stable sorting n integers in the range $[1, \ldots, \lceil n^{\alpha} \rceil]$, for any constant $\alpha \geq 1$, that requires $O((\lg n^{\alpha} / \lg (n/p + \lg p))(n/p + \lg p))$ time.*

We can thus verify that stages 1, 2, 3, 4, and 6 of the sequential R-S-Sort algorithm have corresponding steps in the parallel algorithm. Stage 5 can be efficiently realized by using Cole's merge-sort algorithm [4] which sorts n keys with $p \leq n$ processors in time $O(n \lg n / p)$. We have therefore shown for the EREW PRAM model the following analogue of theorem 4.1.

Theorem 5.1 *Let U be a universe equipped with a total order $<$ that can be evaluated in constant time for any given pair of arguments. For any constant $\beta > 0$, let $f : U \to \{1, \ldots, \lceil n^{1+\beta} \rceil\}$ be a non-decreasing function that can be evaluated in constant time, and suppose that $X_1, \ldots, X_n$ are random variables such that for some positive constant d,*

$$Prob[f(X_i) = k] \leq \frac{d}{\lceil n^{1+\beta} \rceil},$$

for all $i \in \{1, \ldots, n\}$, $k \in \{1, \ldots, \lceil n^{1+\beta} \rceil\}$. Then, for any constant $c > 0$ there exists an EREW PRAM algorithm for stable sorting the sequence (of values) $X_1, \ldots, X_n$ that for all integers n and p, satisfying $p \leq n$ takes, with probability at least $1 - 2^{-cn+1}$, time $O((\lg n / \lg (n/p + \lg p))(n/p + \lg p))$.

Proof : We can verify that stages 1, 3, 4, and 6 of the sequential R-S-Sort algorithm have corresponding implementations in the parallel algorithm, and each such parallel stage requires time $O(n/p + \lg p)$. Note that parallel prefix operations on an EREW PRAM take time $O(n/p + \lg p)$. By way of lemma 5.1, stage 2 requires time $O((\lg n^{1+\beta} / \lg (n/p + \lg p))(n/p + \lg p))$. Stage 5 can be efficiently realized by using Cole's merge-sort algorithm [4] in time $O(n/p + \lg p)$, and the theorem follows. ■

6 Experimental Results

We present indicative experimental results for the sequential algorithm. We ran our experiments on an SGI Power Challenge 64-bit Streaming Superscalar TFP processor with 512MBytes RAM. All software has been written in C – by the authors – and the source code has been compiled using the supplied `cc` compiler with option `-O3` under the operating system `IRIX 6.0.1`. For each of three test problem instances, we performed a series of experiments, and the running times indicated are the average of those experiments. Table 1 compares the performance of quicksort, radix-sort, hybrid-sort and our newly

	w_1			w_2			w_3		
Problem size	2^6	2^{12}	2^{18}	2^6	2^{12}	2^{18}	2^6	2^{12}	2^{18}
Radix-Sort	1.00	27.64	2028	1.79	61.44	5241	3.17	123.32	12202
Hybridsort	0.47	15.62	1361	0.51	17.11	1458	0.50	19.72	1699
Quicksort	0.21	17.92	1801	0.24	24.45	2452	0.29	30.93	3011
Our Algorithm	0.37	9.22	794	0.37	10.41	1037	0.41	13.20	1431

Table 1: Experimental results.

proposed algorithm, where in the proof of theorem 4.1 we fix $D = 5$ and range β from 1 to 3. The input consisted of strings uniformly distributed over an alphabet of size 64. We used strings of length $w_1 = 10$, $w_2 = 20$ and $w_3 = 40$. The results are summarized below. All measurements are normalized with respect to the performance of radix-sort on 64 keys, each of length w_1.

As indicated in table 1, our algorithm outperforms all other algorithms in the problem instances tested, except in the case of small problem instances (case of 64 keys), where quicksort seems to be faster.

7 Conclusion

We have presented new probabilistic sequential and EREW PRAM algorithms for stable sorting n uniformly distributed integer keys in an arbitrary range. Our sequential algorithm achieves the improved probability bound of $1 - 2^{-cn+1}$, for any constant $c > 0$. We have also described an EREW PRAM version of the algorithm that sorts in time $O((\lg n / \lg(n/p + \lg p))(n/p + \lg p))$ using $p \leq n$ processors with the same probability.

References

[1] Béla Bollobás. *Random Graphs.* Academic Press, New York, 1984.

[2] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Ann. Math. Statistics,* 23:493-507, 1952.

[3] B. S. Chlebus. Parallel iterated bucket sort. *Information Processing Letters,* 31:181-183, 1989.

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

[5] S. Fortune and J. Wyllie. Parallelism in random access machines. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing,* pp. 114-118, 1978.

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

[7] T. Hagerup. Hybridsort revisited and parallelized. *Information Processing Letters,* 32:35-39, 1989.

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

[9] M. D. MacLaren. Internal sorting by radix plus sifting. *Journal of the ACM,* 13:404-411, 1966.

[10] H. Meijer and S. G. Akl. The design and analysis of a new hybrid sort algorithm. *Information Processing Letters,* 10:213-218, 1980.

[11] S. Rajasekaran and J. H. Reif. Optimal and sublogarithmic time randomized parallel sorting algorithms. *SIAM Journal on Computing,* 18:594-607, 1989.

[12] S. Rajasekaran and S. Sen. On parallel integer sorting. *Acta Informatica,* 29:1-15, 1992.