Fast Parallel Space Allocation, Estimation and Integer Sorting

Torben Hagerup

MPI–I–91–106 June 1991

Fast Parallel Space Allocation, Estimation and Integer Sorting

TORBEN HAGERUP *

Max-Planck-Institut für Informatik
W–6600 Saarbrücken, Germany

1 Introduction

We show that each of the following problems can be solved fast and with optimal speedup with high probability on a randomized CRCW PRAM using $O(n)$ space:

(1) Space allocation: Given n nonnegative integers $x_1, \ldots, x_n$, allocate n blocks of consecutive memory cells of sizes $x_1, \ldots, x_n$ from a base segment of $O(\sum_{i=1}^{n} x_i)$ consecutive memory cells;
(2) Estimation: Given n integers in the range $1 \,..\, n$, compute "good" estimates of the number of occurrences of each value in the range $1 \,..\, n$;
(3) Integer chain-sorting: Given n integers $x_1, \ldots, x_n$ in the range $1 \,..\, n$, construct a linked list containing the integers $1, \ldots, n$ such that for all $i, j \in \{1, \ldots, n\}$, if i precedes j in the list, then $x_i \leq x_j$.

The running times achieved are $O(\log^* n)$ for problem (1) and $O((\log^* n)^2)$ for problems (2) and (3). Moreover, given slightly superlinear processor and space bounds, these problems or variations of them can be solved in constant expected time.

Our algorithm for space allocation applies equally well to the allocation of consecutively numbered processors, or of any other resource that comes in consecutively numbered units. It is used as a crucial subroutine in our algorithms for problems (2) and (3), and we expect it to have numerous other applications. Variants of the estimation problem also crop up quite frequently. Bast and Hagerup (1991) apply an algorithm similar to ours to the problem of parallel hashing. Chain-sorting is simply standard sorting with a nonstandard output convention, which was used previously by Gavril (1975) in the context of merging. It may be viewed as standard sorting "minus" list ranking, since chain-sorting followed by list ranking of the resulting list is equivalent to standard sorting. The known lower bounds on parallel integer sorting apply to list ranking, but not to chain-sorting, which is one reason for considering the latter problem: If chain-sorting constitutes the "essence" of sorting, as indeed we feel, what happens if it is isolated from list ranking? In addition to the results mentioned above, we also describe a simple algorithm for chain-sorting in constant expected

* Supported in part by the Deutsche Forschungsgemeinschaft, SFB 124, TP B2, VLSI Entwurfsmethoden und Parallelität, and in part by the ESPRIT II Basic Research Actions Program of the EC under contract No. 3075 (project ALCOM).

time with $O(n \log n / \log\log n)$ processors. As a rather trivial by-product of our fast chain-sorting algorithms, we are able to improve the best previous result on (standard) randomized sorting of n integers in the range $1 \,..\, n$.

From a different point of view, the present paper explores the power flowing from a combination of three new techniques in algorithm design and analysis: First, the "$\log^* n$" technique introduced by Raman (1990) and developed in its full potential by Matias and Vishkin (1991). Second, randomized "scattering" procedures for estimating various quantities crudely, but rapidly. And third, the analysis of randomized algorithms using martingale theory, which is not new, but has in the past not been used as often as it deserves.

The structure of the paper is as follows: After some preliminaries in Section 2, Section 3 introduces various concepts under the general heading of "scattering" and lists some of their basic properties. Section 4 defines and solves the ***interval allocation problem***, our formalization of problem (1) above, while Section 5 describes our best algorithm for problem (2). Section 6 is devoted to the chain-sorting problem (3), and Section 7 studies the consequences of allowing slightly superlinear processor and space bounds.

2 Preliminaries

The present paper employs different variants of the CRCW PRAM, definitions of most of which can be found in (Chlebus *et al.*, 1989). While we strive to implement each algorithm in the weakest possible model, we expect the distinction between different variants to be of little concern to most readers, and we rarely go through the not in all cases trivial arguments needed to show that a particular algorithm can be implemented in a particular model. In all cases it will be obvious that the standard ARBITRARY PRAM can be used. The word "processor" is often used not in the sense of a physical processor, but rather to designate a logically distinct ***task***, of which several may be executed by a single physical processor. We speak of ***virtual*** processors when we want to emphasize this point of view.

In order to make many proofs more readable, we make extensive use of the notion of a ***negligible probability***. What constitutes a negligible probability is different from case to case. Is the goal, e.g., to show that some event occurs with probability $2^{-n^{\Omega(1)}}$, then in the proof all probabilities of the form $2^{-n^{\Omega(1)}}$ can be ignored. An event that occurs ***with high probability*** (w.h.p.) is the complement of an event of negligible probability. We often tacitly assume that such events always occur.

Lemma 2.1 states various inequalities commonly known as Chernoff bounds. For proofs see (Hagerup and Rüb, 1990). Lemma 2.2 is folklore.

Lemma 2.1: For every binomially distributed random variable S,

(a) For all ϵ with $0 \le \epsilon \le 1$, $\Pr(S \ge (1+\epsilon)E(S)) \le e^{-\epsilon^2 E(S)/3}$. In particular, $\Pr(S \ge 2E(S)) \le e^{-E(S)/3}$;

(b) For all ϵ with $0 \le \epsilon \le 1$, $\Pr(S \le (1-\epsilon)E(S)) \le e^{-\epsilon^2 E(S)/2}$. In particular, $\Pr(S \le E(S)/2) \le e^{-E(S)/8}$;

(c) For every $z > 0$, $\Pr(S \ge z) \le \left(\frac{eE(S)}{z}\right)^z$.

Lemma 2.2: Let $m \in \mathbb{N}$ and $0 \le p_1, \ldots, p_m \le 1$, and let $X_1, \ldots, X_m$ be 0-1 random variables with a joint distribution. Suppose that

$$\Pr(X_i = 1 \mid X_1 = x_1, \ldots, X_{i-1} = x_{i-1}) \le p_i,$$

for $i = 1, \ldots, m$ and for all $x_1, \ldots, x_{i-1}$ for which the conditional probability is defined. Then for all $z \in \mathbb{R}$, $\Pr(\sum_{i=1}^m X_i \ge z) \le \Pr(\sum_{i=1}^m Y_i \ge z)$, where $Y_1, \ldots, Y_m$ are independent 0-1 random variables with $\Pr(Y_i = 1) = p_i$, for $i = 1, \ldots, m$.

The following fact is implied by Azuma's inequality (Bollobás, 1987). In later applications, we write "by a Chernoff bound" instead of "by Lemma 2.1", and "by a martingale argument" rather than "by Lemma 2.3".

Lemma 2.3: Let $n \in \mathbb{N}$, let $Z_1, \ldots, Z_n$ be independent random variables with finite ranges, and let S be an arbitrary real function of $Z_1, \ldots, Z_n$ with $E(S) \ge 0$. Assume that S changes by at most c in response to an arbitrary change in a single Z_i. Then for every $z \ge 2E(S)$,

$$\Pr(S \ge z) \le e^{-z^2/(8c^2 n)}.$$

Lemma 2.4: For every fixed $\delta > 0$ and for $p \ge n$, the prefix sums of n numbers, each of absolute size at most $2^{(\log p)^{1-\delta}}$, can be computed in $O(\log n / \log\log p)$ time on a p-processor COMMON or TOLERANT PRAM.

Proof: This is a generalization of a result by Ragde (1990, p. 747), proved in essentially the same way. ∎

Corollary 2.5: For every fixed $\delta > 0$ and $m = (\log n)^{O(1)}$, given m nonnegative integers $x_1, \ldots, x_m$, it is possible, on a COMMON or TOLERANT PRAM using constant time, $O(n^\delta)$ processors and $O(n^\delta)$ space, to compute m nonnegative integers $y_1, \ldots, y_m$ such that

(1) For $i = 1, \ldots, m-1$, $y_{i+1} - y_i \ge x_i$;

(2) $y_m \le 4\sum_{i=1}^m x_i$.

Proof: Compute $R = \max\{x_i : 1 \le i \le m\}$ and for $i = 1, \ldots, m$, let x_i' be the smallest multiple of $\lceil R/m \rceil$ no smaller than x_i. Apply Lemma 2.4 to $x_1', \ldots, x_m'$ after observing that $\sum_{i=1}^m x_i' \le 4\sum_{i=1}^m x_i$ and that each x_i' can be expressed in $O(\log m)$ bits (taking $\lceil R/m \rceil$ as a unit). ∎

Lemma 2.6 (Eppstein and Galil (1988, Theorem 4(c))): For every τ with $1 \le \tau \le n$, the maximum of n integers, each of absolute size $n^{O(1)}$, can be computed on a COMMON or TOLERANT PRAM using $O(\tau)$ time, $O(n/\tau)$ processors and $O(n)$ space.

Corollary 2.7: For every τ with $1 \le \tau \le n$, the first 1 in a bit vector of size n can be computed on a COMMON or TOLERANT PRAM using $O(\tau)$ time, $O(n/\tau)$ processors and $O(n)$ space.

The lemma below was essentially shown by Cole and Vishkin (1989, Section 3.2).

Lemma 2.8: For every $m \in \mathbb{N}$ and every τ with $\log n / \log\log n + m \le \tau \le n$, n integers in the range $1\,..\,m$ can be sorted on a COMMON or TOLERANT PRAM using $O(\tau)$ time, $O(n/\tau)$ processors and $O(n)$ space.

3 Scattering

The fundamental intuitive meaning of a ***scattering*** is that each of a number of objects is placed randomly and independently of other objects in one of a number of cells placed in a row. In this paper we are interested in the resulting ***fullness*** of the row, i.e., in the ratio of occupied cells to the total number of cells. Since this is clearly a random variable that tends to take on larger values if more objects are scattered, it provides a (very crude) basis for estimating the number of scattered objects. By letting each object participate in the scattering with some suitable probability instead of with probability 1 as above (a ***conditional scattering***), we can adjust the "region of sensitivity" of a scattering according to need. A ***graduated conditional scattering*** (GCS) takes this idea one step further by providing a whole array of conditional scatterings, each with a different associated probability of participation, which gives us a way to make more substantial statements about the number of scattered objects. Graduated conditional scatterings were introduced in (Hagerup and Radzik, 1990), although not for the purpose of estimation.

Our analysis of the outcome of a GCS is mostly limited to determining the last (i.e., lowest-probability) scattering to satisfy some property. Two properties are relevant to us: The row of a scattering being ***full*** (all cells are occupied); and the row being at least half full. It turns out that testing according to full rows is computationally easier, but leads to less accurate estimates. This is expressed more precisely in the lemmas below.

Definition: For $s \in \mathbb{N}$ and $0 \le p \le 1$, a ***conditional scattering*** with probability p and range s is a random experiment carried out by a set U of elements as follows: Each element $u \in U$, independently of other elements, chooses a random number X_u with $\Pr(X_u = 0) = 1 - p$ and $\Pr(X_u = i) = p/s$, for $i = 1, \ldots, s$. An element $u \in U$ is said to ***occupy*** the value i if $X_u = i$, for $i = 1, \ldots, s$, and the ***fullness*** of the scattering is k/s, where $k = |\{X_u : u \in U\} \setminus \{0\}| = |\{i : 1 \le i \le s \text{ and } i \text{ is occupied by at least one element of } U\}|$.

Lemma 3.1: Let $m, s \in \mathbb{N}$ and $0 \le p \le 1$, and let N be the number of occupied values in a conditional scattering with probability p and range s carried out by a set of m elements. Then for every integer k with $0 \le k \le s$,

(a) $\Pr(N \le k) \le \binom{s}{k} 2^{mp(k/s-1)}$;

(b) $\Pr(N \le s/2) \le 2^{s-mp/2}$;

(c) $\Pr(N < s) \le s \cdot 2^{-mp/s}$;

(d) If $p = 1$, then $\Pr(N \le \min\{s/(4e), m/2\}) \le 2^{-m}$;

(e) $\Pr(N \ge k) \le (mpe/k)^k$.

Proof: By elementary combinatorics. ∎

Definition: For $r, s \in \mathbb{N}$, a ***graduated conditional scattering*** (GCS) with range $r \times s$ is a collection $\mathcal{S} = \{\mathcal{S}_1, \ldots, \mathcal{S}_r\}$, where $\mathcal{S}_i$, called the ith ***row*** of $\mathcal{S}$, is a conditional scattering with probability 2^{-i} and range s, for $i = 1, \ldots, r$. For $0 \le f \le 1$, define the ***last f-row*** of $\mathcal{S}$ as 0 if none of $\mathcal{S}_1, \ldots, \mathcal{S}_r$ has fullness $\ge f$, and otherwise as $\max\{i : 1 \le i \le r \text{ and } \mathcal{S}_i \text{ has fullness} \ge f\}$.

Lemma 3.2: Let $m, r, s \in \mathbb{N}$ and let L be the last 1/2-row of a GCS of m elements with range $r \times s$. Let $M = 2^L s$ and put $c_1 = 1/(2^7 e)$ and $c_2 = 12$. Then

(a) If $m \le c_1 s$, then $\Pr(L > 0) \le 2^{-s}$;

(b) If $m > c_1 s$, then $\Pr(m \le c_1 M) \le 2^{-s}$;

(c) $\Pr(L > 0 \text{ and } m \le c_1 M) \le 2^{-s}$;

(d) If $r \ge \lfloor \log m \rfloor$, then $\Pr(m \ge c_2 M) \le 2^{-s}$.

Proof: If $L > 0$, the fullness of row L is at least 1/2. Hence by Lemma 3.1(e), for every $l > 0$,

$$\Pr(L \ge l) \le \sum_{i=\lceil l \rceil}^{\infty} \left(\frac{2em \cdot 2^{-i}}{s} \right)^{s/2} \le \left(\frac{2^{5-l} em}{s} \right)^{s/2}. \qquad (*)$$

Likewise, if $L < r$, the fullness of row $L+1$ is less than 1/2. Hence by Lemma 3.1(b), for every $l < r$,

$$\Pr(L \le l) \le \min\{1, \sum_{i=-\infty}^{\lfloor l \rfloor} 2^{s - m \cdot 2^{-i-2}}\} \le 2^{s+1-m \cdot 2^{-l-2}}. \qquad (**)$$

By $(*)$, $\Pr(L \ge 1) \le (2^4 em/s)^{s/2}$, which for $m \le c_1 s$ is at most 2^{-s}. This shows (a). To verify (b), apply $(*)$ with $l = \log(m/(c_1 s)) > 0$ to obtain

$$\Pr(m \le c_1 M) = \Pr(L \ge l) \le \left(\frac{2^{5-l} em}{s} \right)^{s/2} = (2^5 c_1 e)^{s/2} = 2^{-s}.$$

(c) follows immediately from (a) and (b). Finally apply $(**)$ with $l = \log(m/(c_2 s)) < r$ to obtain

$$\Pr(m \ge c_2 M) = \Pr(L \le l) \le 2^{s+1-m \cdot 2^{-l-2}} = 2^{s+1-c_2 s/4} = 2^{1-2s} \le 2^{-s},$$

which shows (d). ∎

In the following we shall consistently use c_1 and c_2 with the same meaning as in Lemma 3.2.

Lemma 3.3: Let $m, r, s \in \mathbb{N}$ and $a > 0$ and let L be the last 1-row of a GCS of m elements with range $r \times s$. Then if $M = 2^L s$,

(a) $\Pr(L = r) \le (2^{-r} em/s)^s$;

(b) $\Pr(M > \max\{s, am\}) \le (2e/a)^s$;

(c) $\Pr(L < r \text{ and } m > aM) \le s \cdot 2^{1-a/2}$.

Proof: Similar to that of Lemma 3.2. ∎

Lemma 3.4: Let $r, s \in \mathbb{N}$ and suppose that one (virtual) processor is associated with each element of some set U. Then the last 1-row of a GCS of U with range $r \times s$ can be determined in constant time

(a) on a COLLISION$^+$ PRAM using r additional processors and $O(rs)$ space;

(b) on a COLLISION PRAM using r additional processors and $O(rs)$ initialized space.

Proof: Associate an $r \times s$ array A and an $r \times 1$ *test vector* w with the GCS. Each processor associated with an element of U chooses a random cell $A[I, J]$, where $\Pr(I = i, J = j) = 2^{-i}/s$, for $i = 1, \dots, r$ and $j = 1, \dots, s$, and distinct processors act independently (with whatever probability is left over, processors do nothing). In other words, each processor chooses a random row, row i being chosen with probability 2^{-i}, and then picks a cell at random from the chosen row. Next $w[i]$ is set to 1 exactly if some processor chose a cell in row i, for $i = 1, \ldots, r$. Each processor then checks that some processor chose the right neighbor of the cell that it chose itself (view the leftmost cell as the right neighbor of the rightmost cell); if not, it clears the relevant test cell, indicating that the row is not full. For part (b), this is trivial. For part (a), since initializing A might be too expensive, the check is carried out by letting each processor inspect the right neighbor of its chosen cell before and after adding 1 (say) to its own cell. Finally Corollary 2.7 is used to determine the last 1 in the test vector. ∎

Lemma 3.5: Suppose that one (virtual) processor is associated with each element of some set U. Then for all constants $K, \delta > 0$, the last 1/2-row of a GCS of U with range $r \times s$, where $r, s \leq (\log n)^K$, can be determined on a TOLERANT PRAM using constant time, $O(n^\delta)$ additional processors and $O(n^\delta)$ space.

Proof: By Lemma 2.4 and Corollary 2.7. ∎

4 Interval allocation

Definition: For $n \in \mathbb{N}$ and $d_1, d_2, s \in \mathbb{R}$, the ***incomplete compaction problem*** of size n and with parameters $d_1 \xrightarrow[s]{} d_2$ is the following: Given n bits $x_1, \ldots, x_n$ with $\sum_{j=1}^{n} x_j \leq d_1$, compute n nonnegative integers $y_1, \ldots, y_n$ such that

(1) $|\{j : 1 \leq j \leq n,\ x_j = 1 \text{ and } y_j = 0\}| \leq d_2$;

(2) For $1 \leq i < j \leq n$, if $y_i \neq 0$, then $y_i \neq y_j$;

(3) $\max\{y_j : 1 \leq j \leq n\} = O(s)$.

A natural interpretation of the incomplete compaction problem is that $(x_1, \ldots, x_n)$ is a bit vector representation of a set of at most d_1 *active elements*, scattered over an array of size n, the task being to place all except at most d_2 of these in an array of size $O(s)$. We choose our terminology accordingly.

Lemma 4.1: For all $d \geq 0$ and $1 \leq \tau \leq n$, incomplete compaction problems of size n and with parameters $d^{1/5} \xrightarrow[d]{} 0$ can be solved on a (deterministic) TOLERANT PRAM using $O(\tau)$ time, $O(n/\tau)$ processors and $O(n)$ space.

Proof: The result was proved by Ragde with 1/4 instead of 1/5 for the stronger ARBITRARY PRAM (Ragde, 1990, Theorem 1). Our proof is very similar, but uses a simulation of ARBITRARY PRAMs by TOLERANT PRAMs with more processors, as described in (Chlebus *et al.*, 1989). ∎

Matias and Vishkin (1991) gave an algorithm for incomplete compaction of greater applicability than Ragde's. We describe an algorithm essentially identical to theirs together with a strengthened analysis that achieves a better probability bound. Consider the following experiment, which occurs as a crucial subroutine of the algorithm: v copies of each of m elements scatter over an array of size

s. An element *succeeds* if at least one of its copies does not collide with other copies. The following lemma provides useful bounds on the number of unsuccessful elements.

Lemma 4.2: Let $m, v, s \in \mathbb{N}$ and let $Z_{1,1}, \dots, Z_{m,v}$ be independent and uniformly distributed over $\{1, \dots, s\}$. For $i = 1, \dots, m$ and $j = 1, \dots, v$, let

$$X_{i,j} = \begin{cases} 1, & \text{if } Z_{i',j'} = Z_{i,j} \text{ for some } (i',j') \neq (i,j); \\ 0, & \text{otherwise.} \end{cases}$$

Finally let $S = \sum_{i=1}^{m} \prod_{j=1}^{v} X_{i,j}$. Then

(a) $E(S) \le m \left(\frac{2emv}{s} \right)^{v/2}$;

(b) For every $z \ge 2E(S)$, $\Pr(S \ge z) \le e^{-z^2/(32mv)}$.

Proof: (a) For $j = 1, \dots, v$, define

$$X'_j = \begin{cases} 1, & \text{if } Z_{m,j} = Z_{i',j'} \text{ for some } (i',j') \\ & \quad \text{with either } i' < m \text{ or } (i' = m \wedge j' < j); \\ 0, & \text{otherwise.} \end{cases}$$

In other words, $X'_j = 1$ iff $Z_{m,j}$ has the same value as a lexicographically lower-numbered variable. It is not difficult to see that $\sum_{j=1}^{v} X_{m,j} \le 2 \sum_{j=1}^{v} X'_j$. But

$$\Pr(X'_j = 1 \mid X'_1 = x_1, \dots, X'_{j-1} = x_{j-1}) \le \frac{mv}{s}$$

for $j = 1, \dots, v$ and for all $(x_1, \dots, x_{j-1}) \in \{0,1\}^{j-1}$ with $\Pr(X'_1 = x_1, \dots, X'_{j-1} = x_{j-1}) > 0$. Hence by Lemma 2.2 and Chernoff bound (c),

$$\Pr\left(\sum_{j=1}^{v} X'_j \ge v/2 \right) \le \left(\frac{2emv}{s} \right)^{v/2}$$

and

$$\begin{aligned} E(S) &= m \cdot E\left(\prod_{j=1}^{v} X_{m,j} \right) = m \cdot \Pr\left(\sum_{j=1}^{v} X_{m,j} = v \right) \\ &\le m \cdot \Pr\left(\sum_{j=1}^{v} X'_j \ge v/2 \right) \le m \left(\frac{2emv}{s} \right)^{v/2}. \end{aligned}$$

(b) S is easily seen to satisfy the conditions of Lemma 2.3 with $n = mv$ and $c = 2$, which shows (b).
∎

Theorem 4.3: There is a constant $\epsilon > 0$ such that for all $d, v, \tau \in \mathbb{R}$ with $1 \le d, \tau \le n$ and $1 \le v \le \log d$, incomplete compaction problems of size n and with parameters

$$\frac{d}{v^6} \xrightarrow[d/v^5]{} \frac{d}{2^{6v}}$$

can be solved on a TOLERANT PRAM using $O(\tau)$ time, $O(n/\tau)$ processors and $O(n)$ space with probability at least $1 - 2^{-n^{\epsilon}}$.

Proof (adapted from Raman (1990)): Observe first that it suffices to describe a *basic algorithm* whose failure probability is bounded by $2^{-d^{\epsilon'}}$, for some fixed $\epsilon' > 0$. To see this, note that if $d \geq n^{1/10}$, then the failure probability incurred by the basic algorithm is sufficiently small, whereas if $d < n^{1/10}$, then the active elements can be moved to an array of size at most $\sqrt{n}$ using Lemma 4.1, after which the basic algorithm can be applied independently $\lfloor\sqrt{n}\rfloor$ times, each trial failing with probability at most $1/2$. The basic algorithm is described below.

Without loss of generality assume that v is an integer (otherwise replace v by $\lfloor v \rfloor$ and apply the algorithm twice). Also assume first that $v \leq \frac{1}{15}\log d$.

Step 1: Scatter the active elements over an array of size $\lceil Kd/v \rceil$, where $K = \lceil 2^{17}e \rceil$. Let D be the set of colliding elements.

Step 2: Divide $\{1,\ldots,n\}$ into $\lceil n/v^6 \rceil$ *clusters* $C_1,\ldots,C_{\lceil n/v^6 \rceil}$ of size v^6 each, except for one cluster, which may be smaller. For $i = 1,\ldots,\lceil n/v^6 \rceil$ and using the algorithm of Lemma 4.1, attempt to compact the elements of $C_i \cap D$ into an array of size v^5 (this will succeed if $|C_i \cap D| \leq v$). Let F_1 be the set of active elements in clusters for which the compaction fails.

Step 3: Associate v processors with each active element not in F_1 (by construction, this is trivial) and scatter these processors over an array of size $\lceil Kd/v^5 \rceil$. Place each element with at least one noncolliding associated processor in the cell corresponding to one such processor.

By an argument already used in the proof of Lemma 4.2, for each fixed $i \in \{1,\ldots,\lceil n/v^6 \rceil\}$,

$$\Pr(|C_i \cap D| \geq v) \leq \left(e \cdot v^6 \cdot \frac{d/v^6}{Kd/v} \cdot \frac{1}{v/2}\right)^{v/2} = \left(\frac{2e}{K}\right)^{v/2} \leq 2^{-8v}.$$

It follows that

$$E(|F_1|) \leq d \cdot 2^{-8v}.$$

Since the random choice made by a single active element in Step 1 can affect $|F_1|$ by at most $2v^6$, a martingale argument gives that $|F_1| \leq d \cdot 2^{-7v}$, except with negligible probability. By Lemma 4.2, the set F_2 of active elements that are neither in F_1 nor placed in Step 3 (i.e., those without noncolliding associated processors), satisfies

$$E(|F_2|) \leq d\left(2e \cdot \frac{d}{v^6} \cdot v \cdot \frac{1}{Kd/v^5}\right)^{v/2} \leq d\left(\frac{2e}{K}\right)^{v/2} = d \cdot 2^{-8v},$$

and the actual number of active elements in F_2 after Step 3 is bounded by $2d \cdot 2^{-8v} \leq d \cdot 2^{-7v}$, except with negligible probability. Adding to these the elements in F_1, we may conclude that the number of elements remaining active after Step 3 is at most $2d \cdot 2^{-7v} \leq d \cdot 2^{-6v}$, except with negligible probability. This proves the theorem in the case $v \leq \frac{1}{15}\log d$. If $v > \frac{1}{15}\log d$, the same algorithm and the same argument show that for some constant $\alpha > 0$, the number of active elements is reduced to at most $d^{1-\alpha}$. Now apply the algorithm again, but with $v = \lfloor d^{\alpha/6} \rfloor$, and note that $\Pr(F_1 \cup F_2 \neq \emptyset) \leq E(|F_1| + |F_2|) \leq d \cdot 2^{-7v}$. The latter probability is negligible. In other words, no elements remain active. ∎

Corollary 4.4: For all fixed $k \in \mathbb{N}$, there is a constant $\epsilon > 0$ such that for all τ with $1 \leq \tau \leq n$, incomplete compaction problems of size n and with parameters $d \underset{s}{\longrightarrow} 0$, where $s = d \cdot \max\{\log^{(k)} n, 1\}$, can be solved on a TOLERANT PRAM using $O(\tau)$ time, $O(n/\tau)$ processors and $O(n)$ space with probability at least $1-2^{-n^\epsilon}$.

Proof: Apply Theorem 4.3 a constant number of times. ∎

Definition: For $n \in \mathbb{N}$ and $d \in \mathbb{R}$, the ***complete linear compaction*** problem of size n and with limit d is the incomplete compaction problem of size n and with parameters $d \underset{d}{\longrightarrow} 0$.

Corollary 4.5 (to Theorem 4.3): There is a constant $\epsilon > 0$ such that for all τ with $\log^* d \le \tau \le n$, complete linear compaction problems of size n and with limit d can be solved on a TOLERANT PRAM using $O(\tau)$ time, $O(n \log^* d/\tau)$ processors and $O(n)$ space with probability at least $1-2^{-n^\epsilon}$.

Proof: Apply Theorem 4.3 at most $\log^* d$ times, starting with $v = 1$. ∎

A weaker form of Corollary 4.5 was first proved by Matias and Vishkin (1991), who also noted that it has applications to processor scheduling as per Brent's principle. We next describe an improved algorithm that achieves optimality.

Theorem 4.6: There is a constant $\epsilon > 0$ such that for all τ with $\log^* d \le \tau \le n$, complete linear compaction problems of size n and with limit $d \le n$ can be solved on a TOLERANT PRAM using $O(\tau)$ time, $O(n/\tau)$ processors and $O(n)$ space with probability at least $1-2^{-n^\epsilon}$.

Proof: Assume that $d \ge n^{1/6}$, since otherwise the claim follows from Lemma 4.1 and Corollary 4.5. We describe a preprocessing stage that reduces the problem size from n to $O(n/\log^* d)$. Divide the set $\{1,\ldots,n\}$ into $O(n/\log^* d)$ ***ranges*** of size at most $\log^* d$ each and associate a (virtual) processor with each range. Using a global array A of size $2d$, the processors now execute $4\log^* d$ ***rounds***. In each round, each processor chooses an active element in its range, if any are left, and attempts to place the chosen element in a random cell of A. If the cell is not already occupied and there is no collision, the element becomes inactive. Since each such trial succeeds with probability at least $1/2$ independently of previous trials, Lemma 2.2 and Chernoff bound (b) imply that the probability that a fixed processor has any active elements left after $4\log^* d$ rounds (call such a processor ***busy***) is at most $e^{-\log^* d/4}$. The expected total number of busy processors is therefore

$$O\left(\frac{n}{\log^* d \cdot e^{-\log^* d/4}}\right) = O\left(\frac{n}{(\log^* d)^2}\right),$$

and by a martingale argument, the actual number of busy processors is also $O(n/(\log^* d)^2)$, except with negligible probability. Use Corollary 4.5 to place the busy processors in an array of size $O(n/(\log^* d)^2)$. This implicitly places the remaining active elements in an array of size $O(n/\log^* d)$, and the compaction can be completed using Corollary 4.5. ∎

The ***multi-compaction*** problem defined below generalizes the compaction problem by allowing elements to belong to different ***classes***. Each class is to be compacted into a separate array.

Definition: For $n, m \in \mathbb{N}$ and $d_1,\ldots,d_m \in \mathbb{R}$, the (complete) ***multi-compaction*** problem of size n and with parameters $d_1,\ldots,d_m$ is the following: Given n integers $x_1,\ldots,x_n$ in the range $0\,..\,m$ such that for $i = 1, \ldots, m$, $|\{j : 1 \le j \le n \text{ and } x_j = i\}| \le d_i$, compute n nonnegative integers $y_1, \ldots, y_n$ such that

(1) For $j = 1, \ldots, n$, $x_j = 0 \Leftrightarrow y_j = 0$;

(2) For $1 \le i < j \le n$, if $x_i = x_j \neq 0$, then $y_i \neq y_j$;

(3) For $i = 1, \ldots, n$, $\max\{y_j : 1 \le j \le n \text{ and } x_j = i\} = O(d_i)$.

Theorem 4.7: There is a constant $\epsilon > 0$ such that for $m = (\log n)^{O(1)}$ and for all τ with $\log^* n \le \tau \le n$, multi-compaction problems of size n and with parameters $d_1, \ldots, d_m$ can be solved on a TOLERANT PRAM using $O(\tau)$ time, $O(n/\tau)$ processors and $O(n)$ space with probability at least $1 - 2^{-n^\epsilon}$.

Proof: For $i = 1, \ldots, m$, let *Class i* be the set $\{j : 1 \le j \le n \text{ and } x_j = i\}$. Essentially apply the algorithm of Theorem 4.3 $\log^* n$ times, one difference being that for each $i \in \{1, \ldots, m\}$, the elements in Class i scatter in Step 3 over a separate array of size $\lceil K d_i / v^5 \rceil$, space for which can be allocated using Corollary 2.5. The analysis of Theorem 4.3 carries over to this more general situation and shows that each class is compacted correctly, except with negligible probability. Optimality can be achieved as in Theorem 4.6. ∎

Prior to studying the interval allocation problem, we describe a first simple estimation algorithm. In giving a precise definition of the estimation problem, part of the challenge is to find the right characterization of a good estimate. Estimates that are correct up to a constant factor (a *fine-estimator*, in the terminology introduced below) are certainly sufficient for our purposes. Unfortunately, however, such estimates are hard to come by, and we are forced to relax our requirements to those defining a *coarse-estimator*.

Definition: Let $n, m \in \mathbb{N}$, and let $x_1, \ldots, x_n$ be n integers in the range $1\,..\,m$. For $i = 1, \ldots, m$, let $b_i = |\{j : 1 \le j \le m \text{ and } x_j = i\}|$. A *fine-estimator* for $x_1, \ldots, x_n$ of width m is a sequence of m nonnegative integers $\hat{b}_1, \ldots, \hat{b}_m$ such that $b_i \le \hat{b}_i \le K b_i$, for $i = 1, \ldots, m$ and for some constant K. A *coarse-estimator* for $x_1, \ldots, x_n$ of width m is a sequence of m independent nonnegative integer random variables $\hat{b}_1, \ldots, \hat{b}_m$ such that

(A) $\sum_{i=1}^m \hat{b}_i = O(n)$;

(B) For $i = 1, \ldots, m$ and for all $a \ge 1$, $\Pr(b_i > a\hat{b}_i) \le 2^{-a}$.

For $n, m \in \mathbb{N}$ and $0 \le p \le 1$, an algorithm solves the fine-estimation (coarse-estimation) problem of size n and width m with probability p if it inputs n integers $x_1, \ldots, x_n$ in the range $1\,..\,m$ and, conditionally on an event of probability p, outputs a fine-estimator (coarse-estimator) for $x_1, \ldots, x_n$ of width m.

A statement quite similar to Lemma 4.8 below can be derived by combining results of (Stockmeyer, 1983) and (Ajtai and Ben-Or, 1984) with the standard simulation of unbounded fan-in circuits by CRCW PRAMs (Stockmeyer and Vishkin, 1984). We give a different proof, which in the context of PRAMs seems more direct.

Lemma 4.8: For every fixed $\delta > 0$, there is a constant $\epsilon > 0$ such that the following problem can be solved on a TOLERANT PRAM using constant time, $O(n^{1+\delta})$ processors and $O(n^{1+\delta})$ space with probability at least $1 - 2^{-n^\epsilon}$: Given n bits $x_1, \ldots, x_n$, compute a bit y such that

(1) $\sum_{j=1}^n x_j \geq n/2 \Rightarrow y = 1$;

(2) $\sum_{j=1}^n x_j \leq n/8 \Rightarrow y = 0$.

Proof: The idea of the proof, which the reader may appreciate better after the first reading, is to "amplify" a constant-factor difference to a "polynomial" difference, which can then easily be detected using Ragde's lemma (Lemma 4.1).

Assume $\delta \leq 1$, take $q = \lceil n^{\delta/2} \rceil$ and $t = \lfloor \frac{n}{32\lceil \log n \rceil} \rfloor$ and let A be an array whose size is a multiple of t between $2nq$ and $4nq$. As usual, we consider the ones in the input as (active) elements. Scatter q numbered copies of each element over A. W.h.p., each element will have at least one noncolliding copy; move it to the cell in A occupied by the lowest-numbered such copy and observe that all distributions of the elements in A have the same probability.

Divide A into t subarrays of the same size and define a subarray to be ***heavy*** if it contains more than $8\lceil \log n \rceil$ elements. Associate q processors with each cell of A and use Lemmas 4.1 and 2.4 to determine the set of heavy subarrays. Associate a representative with each heavy subarray and attempt, using Lemma 4.1, to move the set of representatives to an array of size $\lfloor t/5 \rfloor$. Set $y = 1$ if and only if this fails.

Let $d = \sum_{j=1}^n x_j$ be the total number of elements and note that the expected number of elements in a fixed subarray is d/t. This quantity is $\geq 16\lceil \log n \rceil$ if $d \geq n/2$, while for sufficiently large values of n it is $\leq 4\lceil \log n \rceil + 1$ if $d \leq n/8$. Using Lemmas 2.1 and 2.2, one can show that for sufficiently large values of n, the probability that a fixed subarray is heavy is at least $1 - n^{-1}$ if $d \geq n/2$, while it is at most n^{-1} if $d \leq n/8$. Hence the expected number of heavy subarrays is $\geq t/2$ if $d \geq n/2$, while it is ≤ 1 if $d \leq n/8$. By a martingale argument, w.h.p. the actual number of heavy subarrays is $\geq t/4$ if $d \geq n/2$, while it is $\leq \lfloor t/5 \rfloor^{1/5}$ if $d \leq n/8$. In the first case, the compaction using Lemma 4.1 will surely fail, while in the second case it will succeed. In either case y receives the correct value. ∎

When using the algorithm of Lemma 4.8 to analyze the outcome of a GCS $\mathcal{S} = \{\mathcal{S}_1, \ldots, \mathcal{S}_r\}$ below, we apply the algorithm separately to each row of $\mathcal{S}$ and define a row to be ***almost-full*** if the algorithm assigns the value 1 to the bit y associated with the row. The ***last almost-full row*** of $\mathcal{S}$ is 0 if none of $\mathcal{S}_1, \ldots, \mathcal{S}_r$ is almost-full, and otherwise is $\max\{i : 1 \leq i \leq r \text{ and } \mathcal{S}_i \text{ is almost-full}\}$.

In the remainder of the paper, whenever dealing with n integers $x_1, \ldots, x_n$ in the range $1\,..\,m$, define $B_i = \{j : 1 \leq j \leq n \text{ and } x_j = i\}$ and $b_i = |B_i|$, for $i = 1, \ldots, m$, and call i a ***color***, B_i a ***color class*** and b_i the ***multiplicity*** of i.

Theorem 4.9: For every fixed $\delta > 0$, there is a constant $\epsilon > 0$ such that for all τ with $1 \leq \tau \leq n^\epsilon$, fine-estimation problems of size n and width $m \leq n^{1-\delta}$ can be solved on a TOLERANT PRAM using $O(\tau)$ time, $O(n/\tau)$ processors and $O(n)$ space with probability at least $1 - 2^{-n^\epsilon}$.

Proof: Let $s = \lceil n^{\delta/3} \rceil$. For $i = 1, \ldots, m$, set $\hat{b}_i = 0$ if $b_i = 0$, and otherwise carry out the following procedure:

Step 1: Using Lemma 4.8, execute a GCS $\mathcal{S}_i$ of B_i with range $\lfloor \log n \rfloor \times s$ and let l_i be the last almost-full row of $\mathcal{S}_i$.

Step 2: If $l_i > 0$, take $\hat{b}_i = c_2 \cdot 2^{l_i} s$. Otherwise use Corollary 4.4 to allocate s processors to each element of B_i, let these processors execute a GCS S'_i with range $\lfloor \log(sn) \rfloor \times s$ and take $\hat{b}_i = c_2 \cdot 2^{l'_i}$, where l'_i is the last almost-full row of S'_i.

It is easy to see that the space needed by the algorithm is $O(n)$. To see that the same holds for the number of operations, note that by Lemma 3.2(d) and condition (1) of Lemma 4.8, no processors are allocated to a fixed color class B_i with $b_i \geq c_2 s$, except with negligible probability.

Arguing as in its proof, one easily shows that the assertions of Lemma 3.2 continue to hold, except with negligible probability, if L is taken to be the last almost-full row and c_1 is replaced by a suitable positive constant $c'_1 < 1$. Now fix $i \in \{1, \ldots, m\}$ and use Lemma 3.2, modified in this way, to conclude that the following happens w.h.p.: If $l_i > 0$, we have $b_i \leq \hat{b}_i \leq (c_2/c'_1) b_i$. If $l_i = 0$, then $sb_i \leq c_2 \cdot 2^{l'_i} s \leq (c_2/c'_1) sb_i$, i.e., $b_i \leq \hat{b}_i \leq (c_2/c'_1) b_i$. ∎

Definition: For $n \in \mathbb{N}$, the (complete) ***interval allocation*** problem of size n is the following: Given n nonnegative integers $x_1, \ldots, x_n$, compute n nonnegative integers $y_1, \ldots, y_n$ such that

(1) For $j = 1, \ldots, n$, $x_j = 0 \Leftrightarrow y_j = 0$;

(2) For $1 \leq i < j \leq n$, if $0 \notin \{x_i, x_j\}$, then $\{y_i, \ldots, y_i + x_i - 1\} \cap \{y_j, \ldots, y_j + x_j - 1\} = \emptyset$;

(3) $\max\{y_j : 1 \leq j \leq n\} = O(\sum_{j=1}^{n} x_j)$.

While the compaction problem asks that unit intervals be placed in a base segment, the interval allocation problem specifies intervals of varying length to be placed. Viewed another way, each interval is a request for a ***block*** of consecutive indices of a certain size. Informally, condition (2) means that blocks do not overlap, and (3) means that the allocated blocks are optimally packed, except for a constant factor.

Theorem 4.10: There is a constant $\epsilon > 0$ such that for all τ with $\log^* n \leq \tau \leq n$, interval allocation problems of size n can be solved on a TOLERANT PRAM using $O(\tau)$ time, $O(n/\tau)$ processors and $O(n)$ space with probability at least $1 - 2^{-n^{\epsilon}}$.

Proof: Let the input be $x_1, \ldots, x_n$, let $W = \sum_{i=j}^{n} x_j$ and begin by computing $R = \max\{x_j : 1 \leq j \leq n\}$. This can be done through a simple combination of an algorithm by Reischuk (1985) with Lemma 4.1, the details of which are described elsewhere (Hagerup, 1991). Actually, for the applications of Theorem 4.10 in the present paper, we will always have $R = O(n)$, and it is not necessary to actually compute R. Let $u = \lceil 2R/n \rceil$ and replace x_j by the smallest number x'_j in the set $\{0\} \cup \{2^i u : 1 \leq i \leq \lfloor \log n \rfloor\}$ no smaller than x_j, for $j = 1, \ldots, n$. Then $W \leq \sum_{j=1}^{n} x'_j \leq 6W$, and as a result of this transformation, we can and will in the following assume that each nonzero input number x'_j is a power of 2 in the range $2 \,..\, n$ (simply measure blocks and intervals in units of size u). Let $m = \lfloor \log n \rfloor$ and for $i = 1, \ldots, m$, let $B_i = \{j : 1 \leq j \leq n \text{ and } x'_j = 2^i\}$ and $b_i = |B_i|$. At this point the problem is trivial if $\tau \geq \log n$. Otherwise use Theorem 4.9 to compute estimates $\hat{b}_1, \ldots, \hat{b}_m$ such that $b_i \leq \hat{b}_i \leq K b_i$, for $i = 1, \ldots, m$ and for some constant K. Now for $i = 1, \ldots, m$, two things remain: To allocate sufficient space to B_i, i.e., to allocate $c\hat{b}_i \cdot 2^i$ cells to B_i from a common base segment of size $O(W)$, for a suitable constant $c \in \mathbb{N}$, and to divide the space allocated to B_i among the elements of B_i, with each element receiving 2^i cells. The first problem can be solved using

Corollary 2.5, and the second problem reduces to the multi-compaction problem of size n and with parameters $\hat{b}_1, \ldots, \hat{b}_m$ and can be solved using Theorem 4.7. ∎

Whereas the use of Theorem 4.10 in memory allocation is obvious, one additional observation is needed for its application to the allocation of processors. The reason is that a processor is an active device that needs to know about the task that it is to execute. Theorem 4.10 can be used to communicate this information to the first processor in each "block", i.e., in each group of consecutively numbered processors allocated to a common task, but the information must subsequently be broadcast to the remaining processors in each block. In recognition of this fact, we consider a slight variation of the interval allocation problem.

Definition: For $n \in \mathbb{N}$, the ***interval marking*** problem of size n is the following: Given n nonnegative integers $x_1, \ldots, x_n$ with $\sum_{j=1}^{n} x_j = O(n)$, compute nonnegative integers $s, z_1, \ldots, z_s$ with $s = O(n)$ such that

(1) For all integers i, j, k with $1 \le i \le j \le k \le s$, if $z_i = z_k \neq 0$, then $z_j = z_i$;

(2) For $i = 1, \ldots, n$, $|\{j : 1 \le j \le s \text{ and } z_j = i\}| = x_i$.

The broadcasting problem mentioned above reduces to the ***all nearest zero bit*** problem defined below. Lemma 4.11, due to Berkman and Vishkin (1989), states that this problem can be solved using negligible resources. Berkman and Vishkin in fact prove slightly stronger results that those cited in the lemma.

Definition: The ***all nearest zero bit*** problem of size n is, given a bit vector A of size n, to mark each position in A with the position of the nearest zero in A to its left, if any.

Lemma 4.11: All nearest zero bit problems of size n can be solved on a COMMON or TOLERANT PRAM

(a) in $O(\tau)$ time using $O(n/\tau)$ processors and $O(n)$ space, for all τ with $\log^* n \le \tau \le n$;

(b) in constant time using $O(n \log^* n)$ processors and $O(n \log^* n)$ space.

Theorem 4.12: There is a constant $\epsilon > 0$ such that for all τ with $\log^* n \le \tau \le n$, interval marking problems of size n can be solved on a TOLERANT PRAM using $O(\tau)$ time, $O(n/\tau)$ processors and $O(n)$ space with probability at least $1 - 2^{-n^{\epsilon}}$.

Proof: By the above discussion. ∎

5 Estimation

In this section we show that coarse-estimation problems of size and width n can be solved optimally in $O((\log^* n)^2)$ time. We first explain the main ideas in the context of an algorithm that uses $O(n \log^* n)$ operations and later indicate how to achieve optimality. We begin by tackling a simpler problem, that of computing estimates for just the colors with large multiplicities.

Theorem 5.1: Let $x_1, \ldots, x_n$ be integers in the range $1 \,..\, n$, and let $B_i = \{j : 1 \le j \le n \text{ and } x_j = i\}$ and $b_i = |B_i|$, for $i = 1, \ldots, n$. Then for every fixed $\delta > 0$, there is a constant $\epsilon > 0$ such that it is possible, on a TOLERANT PRAM using constant time, $O(n)$ operations and $O(n)$ space, to compute nonnegative integers $\hat{b}_1, \ldots, \hat{b}_n$ such that with probability at least $1 - 2^{-n^\epsilon}$, the following holds for each $i \in \{1, \ldots, n\}$:

(1) $\hat{b}_i > 0 \Rightarrow b_i \le \hat{b}_i \le K b_i$, for some constant K;

(2) $b_i \ge n^\delta \Rightarrow \hat{b}_i > 0$.

Proof: Let $h = \lceil n^{\delta/4} \rceil$ and carry out the following algorithm:

Step 1: Draw a random sample Y of $\{1, \ldots, n\}$ by including each element of $\{1, \ldots, n\}$ in Y independently of other elements and with probability $1/h$. For $i = 1, \ldots, n$, let $B_i^Y = B_i \cap Y$ and $b_i^Y = |B_i^Y|$.

Step 2: Use Theorem 4.9 to estimate b_i^Y, for $i = 1, \ldots, n$. Note that although $|Y| = O(n^{1-\delta/4})$ w.h.p. by Chernoff bound (a), this is not directly possible, since the elements of Y as well as their values $\{x_j : j \in Y\}$ are scattered over ranges of size n. Therefore first use Corollary 4.4 to place the elements of Y and their values in arrays of size $O(n^{1-\delta/8})$, then, simulating an ARBITRARY PRAM with fewer processors on the available TOLERANT PRAM, select a representative in each nonempty set B_i^Y, and finally use the positions of the representatives as new values in a range of size $O(n^{1-\delta/8})$, while keeping the bijection between old and new values. Now Theorem 4.9 provides estimates $\hat{b}_1^Y, \ldots, \hat{b}_n^Y$ such that w.h.p., $b_i^Y \le \hat{b}_i^Y \le K' b_i^Y$, for $i = 1, \ldots, n$ and for some constant K'.

Step 3: For $i = 1, \ldots, n$, if $\hat{b}_i^Y \ge n^{\delta/2}$, then take $\hat{b}_i = 2h\hat{b}_i^Y$; otherwise take $\hat{b}_i = 0$.

Fix $i \in \{1, \ldots, n\}$. If $b_i \ge n^{\delta/2}$, then w.h.p. $b_i/(2h) \le b_i^Y \le 2b_i/h$ and hence $b_i/(2h) \le \hat{b}_i^Y \le 2K'b_i/h$, from which follows that either $\hat{b}_i = 0$ or $b_i \le \hat{b}_i \le 4K'b_i$. If $b_i \ge n^\delta$, clearly w.h.p. $\hat{b}_i > 0$. On the other hand, if $b_i < n^{\delta/2}$, then w.h.p. $b_i^Y < n^{\delta/2}/K'$, $\hat{b}_i^Y < n^{\delta/2}$ and $\hat{b}_i = 0$. ■

Lemma 5.2: There is a constant $\epsilon > 0$ such that coarse-estimation problems of size and width n can be solved on a COLLISION PRAM using $O((\log^* n)^2)$ time, $O(n \log^* n)$ operations and $O(n)$ space with probability at least $1 - 2^{-n^\epsilon}$.

We begin by describing the algorithm informally. As in the previous estimation algorithms, the basic idea is that a GCS for each color can be used to estimate the multiplicity of that color. Since we now have n colors to estimate, but want to get by with $O(n)$ space, however, we can allocate only constant space to each GCS, which yields extremely poor estimates. Our solution has several components. We indeed begin by allocating constant space to each color, but then reassign ever more resources to the estimation of colors that, based on previous less accurate estimates, appear to have large multiplicities. In other words, the algorithm proceeds in a number of *stages*, each of which produces more accurate estimates of fewer and presumably larger multiplicities than its predecessor. Furthermore, we have carefully matched the definition of the coarse-estimation problem to what the algorithm can actually produce.

It turns out that the resources allocated to each surviving color can be increased very rapidly, causing the necessary number of stages to be $O(\log^* n)$. This, however, is achieved at a considerable loss of precision, as compared to an estimation at a more sedate pace, and the estimates obtained from the basic procedure are not sufficiently accurate and have to undergo a final readjustment. The latter is made possible by the less accurate estimates (a kind of bootstrapping), which allows us to divide the available resources between colors in approximate proportion to their multiplicities.

We now provide a more formal description of the algorithm. By Theorem 5.1, we can assume that $b_i \le n^{1/8}$, for $1 \le i \le n$. Begin by computing integers T and $v_1, \ldots, v_T$ such that (1) $v_T = \lceil n^{1/9} \rceil$, (2) for $t = 2, \ldots, T$, $v_{t-1} = \lceil \log v_t \rceil$, and (3) $v_1 = 1$. Then execute the following steps:

```
(1)  for $i \in \{1,\ldots,n\}$ pardo
(2)    if $b_i = 0$ then $\hat{b}_i := 0$ else let $i$ be active;
(3)  for $t := 1$ to $T$ do
(4)    for each active $i \in \{1,\ldots,n\}$ pardo
(5)      begin
(6)        Allocate a $2v_t \times v_t$ array and $2v_t$ processors to $i$;
(7)        Let the elements of $B_i$ carry out a GCS $\mathcal{S}_i$
(8)          with range $2v_t \times v_t$;
(9)        $l_i :=$ last 1-row of $\mathcal{S}_i$;
(10)       if $l_i < 2v_t$ (* not entirely full *) then
(11)         begin
(12)           Make $i$ inactive;
(13)           $\tilde{b}_i := c_2 \cdot 2^{l_i} v_t$; (* preliminary estimate *)
(14)         end;
(15)     end;
(16) for each $i \in \{1,\ldots,n\}$ with $b_i > 0$ pardo
(17)   begin
(18)     $r_i := 2\lceil \log \tilde{b}_i \rceil + 4$; $s_i := 3\lceil \log \tilde{b}_i \rceil$;
(19)     Allocate an $r_i \times s_i$ array and $\tilde{b}_i$ processors to $i$;
(20)     Let the elements of $B_i$ carry out a GCS $\mathcal{S}_i$
(21)       with range $r_i \times s_i$;
(22)     $l_i :=$ last 1/2-row of $\mathcal{S}_i$;
(23)     $\hat{b}_i := \max\{\tilde{b}_i, c_2 \cdot 2^{l_i} s_i\}$;
(24)   end;
```

Each allocation of space and processors in lines (6) and (19) can be done in $O(\log^* n)$ time using Theorems 4.10 and 4.12. Provided that the allocated resources stay within the limits imposed by Lemma 5.2, it is not difficult to see that the whole algorithm can be executed within the time, processor and space bounds allowed by the lemma (use Lemmas 3.4 and 3.5). The lemmas below show that the resources allocated are not excessive and that the output of the algorithm is indeed a coarse-estimator with the required probability. For $t = 1, \ldots, T$, *Stage* t is the tth execution of lines (4)–(15). Ignore color classes of multiplicity 0.

Lemma 5.3: W.h.p., every $\tilde{b}_i$, for $i = 1, \ldots, n$, is defined in some stage.

Proof: Fix $i \in \{1,\ldots,n\}$. If $\tilde{b}_i$ is not defined in any stage, $l_i = 2v_T$ in Stage T. By Lemma 3.3(a), the probability of this is at most $(2^{-2v_T} en/v_T)^{v_T}$, i.e., negligible. ∎

Definition: For $i = 1,\ldots,n$ and $t = 1,\ldots,T$, call i ***tardy*** in Stage t if $\tilde{b}_i$ is defined in Stage t or later and $b_i < c_1 v_t^2$.

Lemma 5.4: For $i = 1,\ldots,n$ and $t = 2,\ldots,T$, $\Pr(i \text{ is tardy in Stage } t) \le v_t^{-7}$.

Proof: If i is tardy in Stage t, $l_i = 2v_{t-1}$ in Stage $t-1$. By Lemma 3.3(a), the probability of this is at most
$$\left(\frac{2^{-2v_{t-1}} e c_1 v_t^2}{v_{t-1}}\right)^{v_{t-1}} \le (c_1 e)^{v_{t-1}} \le 2^{-7v_{t-1}} \le v_t^{-7}. \quad ∎$$

Lemma 5.5: W.h.p., the total amount of space (and hence processors) allocated in line (6) over all stages is $O(n)$.

Proof: Fix $i \in \{1,\ldots,n\}$, let S_i be the total amount of space allocated to i in line (6), and for $t = 1,\ldots,T$, denote by A_t the event that i is active at the beginning of Stage t. By a martingale argument, it suffices to show that $E(S_i) = O(b_i)$. But
$$\begin{aligned} E(S_i) &= \sum_{t=1}^{T} 2v_t^2 \Pr(A_t) \\ &= O(1) + \sum_{\substack{t=2 \\ v_t^2 \le b_i/c_1}}^{T} 2v_t^2 \Pr(A_t) + \sum_{\substack{t=2 \\ v_t^2 > b_i/c_1}}^{T} 2v_t^2 \Pr(A_t) \\ &= O(1) + \sum_{\substack{t=2 \\ v_t^2 \le b_i/c_1}}^{T} 2v_t^2 + \sum_{t=2}^{T} 2v_t^2 \Pr(i \text{ is tardy in Stage } t). \end{aligned}$$

The first sum is $O(b_i)$ since the sequence $\{v_t\}_{t=1}^{T}$ grows faster than a geometric series, while the second sum is $O(1)$ by Lemma 5.4. ∎

Lemma 5.6: W.h.p., $\max\{\tilde{b}_i : 1 \le i \le n\} = O(n^{1/4})$.

Proof: Fix $i \in \{1,\ldots,n\}$. If $\tilde{b}_i$ is defined in Stage $T-1$ or earlier, $\tilde{b}_i \le c_2 \cdot 2^{2v_{T-1}} v_{T-1}$, which for sufficiently large n is bounded by $n^{1/4}$. If $\tilde{b}_i$ is defined in Stage T, Lemma 3.3(b) implies that $\Pr(\tilde{b}_i > c_2 \cdot \max\{v_T, n^{1/8} b_i\}) \le (2en^{-1/8})^{v_T}$. The claim now follows from the assumption that $b_i \le n^{1/8}$. ∎

Lemma 5.7: W.h.p., $\sum_{i=1}^{n} \tilde{b}_i = O(n)$.

Proof: Fix $i \in \{1,\ldots,n\}$. By Lemmas 2.3 and 5.6, it suffices to show that $E(\tilde{b}_i) = O(b_i)$. For $t = 1,\ldots,T$, denote by D_t and Z_t the events that $\tilde{b}_i$ is defined in Stage t and that i is tardy in Stage t, respectively. Then
$$\begin{aligned} E(\tilde{b}_i) = O(1) &+ \sum_{t=2}^{T} E(\tilde{b}_i \mid D_t \cap Z_t) \Pr(D_t \cap Z_t) \\ &+ \sum_{t=2}^{T} E(\tilde{b}_i \mid D_t \cap \overline{Z_t}) \Pr(D_t \cap \overline{Z_t}). \end{aligned}$$

Fix $t \in \{2, \ldots, T\}$. By Lemma 3.3(b), $\Pr(\bar{b}_i > 16c_2eb_i \mid D_t \cap \overline{Z_t}) \leq 2^{-3v_t}$. Since $\bar{b}_i \leq c_2 \cdot 2^{2v_t}v_t$ if $\bar{b}_i$ is defined in Stage t, it follows that $E(\bar{b}_i \mid D_t \cap \overline{Z_t}) \leq 16c_2eb_i + c_2 \cdot 2^{2v_t}v_t \cdot 2^{-3v_t} = O(b_i)$, and hence that $\sum_{t=2}^{T} E(\bar{b}_i \mid D_t \cap \overline{Z_t})\Pr(D_t \cap \overline{Z_t}) = O(b_i)$.

On the other hand and again by Lemma 3.3(b), $\Pr(\bar{b}_i > c_2v_t^2 \mid D_t \cap Z_t) \leq (2ec_1)^{v_t} \leq v_{t+1}^{-6}$ and therefore $E(\bar{b}_i \mid D_t \cap Z_t) \leq c_2v_t^2 + c_2 \cdot 2^{2v_t}v_tv_{t+1}^{-6} = O(v_t^2)$. Since $\Pr(Z_t) \leq v_t^{-7}$ by Lemma 5.4, we have $\sum_{t=2}^{T} E(\bar{b}_i \mid D_t \cap Z_t)\Pr(Z_t) = O(1)$. The claim follows. ■

Lemma 5.8: W.h.p., $\sum_{i=1}^{n} \hat{b}_i = O(n)$.

Proof: Fix $i \in \{1, \ldots, n\}$ and consider $\bar{b}_i$ as a constant. $\hat{b}_i \leq c_2 \cdot 2^{2\lceil \log \bar{b}_i \rceil + 4} \cdot 3 \log \bar{b}_i \leq 3 \cdot 2^6 c_2 \bar{b}_i^2 \log \bar{b}_i$. Hence by Lemmas 2.3, 5.6 and 5.7, it suffices to show that $E(\hat{b}_i) = O(b_i + \bar{b}_i)$. But by Lemma 3.2(c),

$$\Pr(\hat{b}_i > \bar{b}_i + c_2 \cdot \max\{3\lceil \log \bar{b}_i \rceil, b_i/c_1\}) \leq 2^{-3 \log \bar{b}_i} = \bar{b}_i^{-3}.$$

The claim now follows from the above upper bound on $\hat{b}_i$. ■

Lemma 5.9: For $i = 1, \ldots, n$ and for all $a \geq 1$, $\Pr(b_i > a\bar{b}_i \geq \bar{b}_i \log \bar{b}_i) \leq 2^{-3a}$.

Proof: Fix $t \in \{1, \ldots, T\}$. If $a \geq \log \bar{b}_i$ and $\bar{b}_i$ is defined in Stage t, then $a \geq \log(c_2v_t)$. Hence by Lemma 3.3(c),

$$\Pr(b_i > a\bar{b}_i \geq \bar{b}_i \log \bar{b}_i \text{ and } \bar{b}_i \text{ is defined in Stage } t)$$
$$\leq v_t \cdot 2^{1-6a} \leq v_t \cdot 2^{1-3\log(c_2v_t)}2^{-3a} \leq 2^{-3a} \cdot 2/(c_2^3v_t^2).$$

It follows that

$$\Pr(b_i > a\bar{b}_i \geq \bar{b}_i \log \bar{b}_i) \leq 2^{-3a} \sum_{t=1}^{T} \frac{2}{(c_2^3v_t^2)} \leq 2^{-3a}. \quad ■$$

Lemma 5.10: For $i = 1, \ldots, n$ and for all $a \geq 1$, $\Pr(b_i > a\hat{b}_i) \leq 2^{-a}$.

Proof: Without loss of generality assume that $b_i \geq 2$.

$$\begin{aligned}\Pr(b_i > a\hat{b}_i) \leq{}& \Pr(b_i > a\hat{b}_i \geq \hat{b}_i \log \bar{b}_i) \\ &+ \Pr(a < \log \bar{b}_i \text{ and } b_i > \bar{b}_i \log b_i) \\ &+ \Pr(b_i > a\hat{b}_i \text{ and } a < \log \bar{b}_i \text{ and } b_i \leq \bar{b}_i \log b_i).\end{aligned}$$

Since $\hat{b}_i \geq \bar{b}_i$, the first term can be bounded by Lemma 5.9:

$$\Pr(b_i > a\hat{b}_i \geq \hat{b}_i \log \bar{b}_i) \leq \Pr(b_i > a\bar{b}_i \geq \bar{b}_i \log \bar{b}_i) \leq 2^{-3a}.$$

The second term is zero if $a > \log b_i$. If $a \leq \log b_i$,

$$\Pr(b_i > \bar{b}_i \log b_i) = \Pr(b_i > \bar{b}_i \log b_i \geq \bar{b}_i \log \bar{b}_i) \leq 2^{-3 \log b_i} \leq 2^{-3a}.$$

In order to bound the last term, note that $b_i \leq \bar{b}_i \log b_i$ implies $2 \log \bar{b}_i + 4 \geq \log b_i$. By Lemma 3.2(d), it follows that for each fixed value of $\bar{b}_i$ with $a < \log \bar{b}_i$ and $b_i \leq \bar{b}_i \log b_i$,

$$\Pr(b_i > a\hat{b}_i) \leq 2^{-3 \log \bar{b}_i} \leq 2^{-3a}.$$

Hence $\Pr(b_i > a\hat{b}_i \text{ and } a < \log \bar{b}_i \text{ and } b_i \le \bar{b}_i \log b_i) \le 2^{-3a}$ and $\Pr(b_i > a\hat{b}_i) \le 3 \cdot 2^{-3a} \le 2^{-a}$. This ends the proof of Lemma 5.2. ∎

We now describe an implementation of scattering that might be called "scattering in time", as opposed to "scattering in space". Suppose that we want to determine the number of occupied values in a conditional scattering with probability 1 and range s of some set U. Instead of providing an array of s memory cells, we may provide a single counter, initialized to zero, and an array of s *time slots*. Each element in U chooses a random time slot and increments the counter by one in that time slot (perhaps together with other elements; the increment is by one in any case). The final value of the counter is the desired quantity. An associated processor allocation problem can be solved using Lemma 2.8 and Theorem 4.7.

Theorem 5.11: There is a constant $\epsilon > 0$ such that coarse-estimation problems of size and width n can be solved on a COLLISION PRAM using $O((\log^* n)^2)$ time, $O(n)$ operations and $O(n)$ space with probability at least $1 - 2^{-n^{\epsilon}}$.

Proof: Carry out the following algorithm:

Step 1: For $i = 1, \ldots, n$, let B_i carry out a conditional scattering S_i in time with probability 1 and range $\log^* n$ and take $\hat{b}_i^{(1)} = \lceil 64e \rceil n_i$, where n_i is the number of occupied values in S_i.

Step 2: Draw a random sample $Y \subseteq \{1, \ldots, n\}$ by including each element of $\{1, \ldots, n\}$ in Y independently of other elements and with probability $1/\log^* n$. Pack both the sample and the values represented in the sample into ranges of size $O(n/\log^* n)$, as described in the proof of Theorem 5.1. Then apply the algorithm of Lemma 5.2 to Y to obtain estimates $\hat{b}_1^Y, \ldots, \hat{b}_n^Y$. For $i = 1, \ldots, n$, let $\hat{b}_i^{(2)} = 4 \log^* n \cdot \hat{b}_i^Y$.

Step 3: For $i = 1, \ldots, n$, if $b_i > 0$, then compute the final estimate of b_i as $\hat{b}_i = \max\{\hat{b}_i^{(1)}, \hat{b}_i^{(2)}, 1\}$; otherwise take $\hat{b}_i = 0$.

We can assume that $\hat{b}_1^Y, \ldots, \hat{b}_n^Y$ is indeed a coarse-estimator for Y. It is easy to see that w.h.p., the algorithm works in $O((\log^* n)^2)$ time and uses $O(n)$ space. The correctness of the algorithm is demonstrated in the lemmas below. ∎

Lemma 5.12: W.h.p., $\sum_{i=1}^{n} \hat{b}_i = O(n)$.

Proof: Easy. ∎

Lemma 5.13: For $i = 1, \ldots, n$ and for all $a \ge 1$, $\Pr(b_i > a\hat{b}_i) \le 2^{-a}$.

Proof:

Case 1: $b_i \le 16a \log^* n$. $a\hat{b}_i^{(1)} < b_i \le 16a \log^* n$ implies $64en_i \le 16 \log^* n$, from which follows that $n_i \le \min\{\log^* n/(4e), b_i/2\}$. By Lemma 3.1(d), this happens with probability at most $2^{-b_i} \le 2^{-a}$.

Case 2: $b_i > 16a \log^* n$. Let $b_i^Y = |B_i \cap Y|$. b_i^Y is binomially distributed, and $E(b_i^Y) = b_i/\log^* n$. Hence by Chernoff bound (b), $\Pr(b_i^Y < b_i/(2\log^* n)) \le e^{-b_i/(8 \log^* n)} \le 2^{-2a}$. By property (B) of a coarse-estimator, $\Pr(b_i^Y > 2a\hat{b}_i^Y) \le 2^{-2a}$. But $b_i^Y \ge b_i/(2\log^* n)$ and $b_i^Y \le 2a\hat{b}_i^Y$ together imply $b_i \le 2\log^* n \cdot b_i^Y \le 4a \log^* n \cdot \hat{b}_i^Y = a\hat{b}_i^{(2)}$. Hence $\Pr(b_i > a\hat{b}_i) \le 2^{-2a} + 2^{-2a} \le 2^{-a}$. ∎

6 Integer sorting

The parallel complexity of integer sorting has been intensively studied. We specialize here to the case in which n integers in the range $1\,..\,n$ are to be sorted on a CRCW PRAM. Rajasekaran and Reif (1989) describe a randomized algorithm with optimal speedup for this problem that uses $O(\log n)$ time and $O(n/\log n)$ processors with high probability. Bhatt *et al.* (1989) give a deterministic algorithm that works in $O(\log n/\log\log n)$ time using $O(n(\log\log n)^2/\log n)$ processors. By the result of Beame and Hastad (1989, Corollary 4.2), this is as fast as possible for any algorithm that uses a polynomial number of processors. The lower bound actually is a reduction from the parity problem: If one can sort a sequence of n bits, one can clearly compute their parity in $O(1)$ additional time. This argument, however, breaks down if sorting is replaced by chain-sorting, since the global information is lost, and, as shown in this section, we are indeed able to chain-sort much faster than in $\Theta(\log n/\log\log n)$ time.

Our algorithms share a common overall structure consisting of three parts. Suppose that n integers $x_1,\ldots,x_n$ in the range $1\,..\,n$ are to be chain-sorted. The first part of each algorithm computes more or less accurate estimates $\hat{b}_1,\ldots,\hat{b}_n$ of $b_1,\ldots,b_n$. The second part allocates an array, called a ***bucket*** and of size roughly proportional to $\hat{b}_i$, to each color i and then stores the elements of B_i in this bucket through a random scattering process. The third part chains the elements together in the right order. The same three parts can be distinguished in the algorithms of (Rajasekaran and Reif, 1989).

For $m \in \mathbb{N}$ and $V \subseteq \{1,\ldots,m\}$, define $\text{MAXGAP}_m(V)$ as the size of the largest set disjoint from V and of the form $\{a,\ldots,b\}$ or $\{b,\ldots,m,1,\ldots,a\}$, where $a,b \in \{1,\ldots,m\}$, or as zero if $V=\{1,\ldots,m\}$.

Lemma 6.1: Let m and k be integers with $1 \le k \le m$ and let V be a set chosen randomly from the uniform distribution over all k-element subsets of $\{1,\ldots,m\}$. Then for every $z \ge 0$, $\Pr(\text{MAXGAP}_m(V) \ge z) \le me^{-zk/m}$.

Proof: Easy and omitted. ∎

Theorem 6.2: For every fixed $k \in \mathbb{N}$, n integers in the range $1\,..\,n$ can be chain-sorted on a COLLISION$^+$ PRAM using constant time and $O(kn\log n/\log\log n)$ processors with probability at least $1-n^{-k}$.

Proof: Assume first that $k=1$. We describe the three parts of an algorithm with the desired properties. For simplicity, let us ignore questions of rounding.

Part 1: For $i=1,\ldots,n$, let the elements of B_i carry out a modified GCS S_i with range $(\log n/\log\log n)\times\log n$ and let l_i be the last 1-row of S_i. The modification, which saves a factor of $\Theta(\log n/\log\log n)$ in the number of processors, is that the probability associated with the ith row of S_i is $(\log n)^{-i}$ instead of 2^{-i}, for $i=1,\ldots,\log n/\log\log n$. Using the techniques of Section 3, it is easy to derive from l_i an estimate $\hat{b}_i$ of b_i such that, for some constant $K>0$ and except with negligible probability, $b_i \le \hat{b}_i \le K(\log n)^3 b_i$.

Part 2: For $i = 1, \ldots, n$, associate $8 \log n / \log\log n$ processors with each element of B_i and scatter these over a bucket Q_i of $16e(\log n)^2 \cdot \hat{b}_i$ cells. Call $j \in \{1, \ldots, n\}$ ***lucky*** if some processor associated with j does not collide. By Lemma 4.2, if $b_i \le \hat{b}_i$ for all $i \in \{1, \ldots, n\}$, then the probability that some $j \in \{1, \ldots, n\}$ is unlucky is at most n^{-2}, i.e., negligible. We therefore determine for each $j \in \{1, \ldots, n\}$ the lowest-numbered noncolliding processor associated with j and place j in the cell chosen by that processor.

Part 3: For $i = 1, \ldots, n$, call Q_i ***empty*** if $b_i = 0$. If Q_i is nonempty, define its ***dilation*** as $|Q_i|/b_i$, where $|Q_i|$ denotes the number of cells in Q_i. The assumption $\hat{b}_i \le K(\log n)^3 b_i$ implies that the dilation of Q_i is $O((\log n)^5)$. Hence by Lemma 6.1, the size of the largest gap between consecutive elements stored in Q_i can be bounded by an integer m, where $m = O((\log n)^6)$, except with negligible probability. Now the successor of each element in Q_i whose successor also belongs to Q_i can be determined as follows: Divide Q_i into ***subarrays*** of size $2(m+1)$ and consider the cells of each subarray as the leaves of a complete $(\log n/\log\log n)$-ary tree of constant height. Define a node in such a tree to be ***empty*** if no element is stored in a leaf of the subtree rooted at that node. Using the $\Theta(\log n/\log\log n)$ processors associated with each element, compute for each nonempty node its nearest nonempty right sibling and its leftmost nonempty child, if any (Corollary 2.7). Using this information, compute the successor of each element whose successor belongs to the same tree. In order to handle elements with successors in the tree following their own, repeat the computation with all trees "shifted" by $m + 1$. This leaves only elements with successors in a different bucket. Deal with these, finally, by using Lemma 4.11(b) to compute for each nonempty bucket the next nonempty bucket, if any. This ends the proof of Theorem 6.2 for $k = 1$. In order to extend the result to general $k \in \mathbb{N}$, simply consider k independent computations as above carried out simultaneously. The probability that all fail is at most n^{-k}. ∎

Our next algorithm is allowed $\Theta(\log n/\log\log n)$ time, which puts it in a time range where parallel computing is easy. In particular, we have sufficient time to convert the linked list produced by a chain-sorting routine to the standard output format of n numbers stored in order in an array of size n. The resulting algorithm for sorting integers of linear size achieves at the same time optimal speed and optimal speedup, which makes it superior to all previously published algorithms. The same result was found independently by Matias and Vishkin (1991) and by Raman (1991).

Our algorithm makes use of a subroutine for optimal monotonic list ranking. The ***monotonic list ranking*** problem of size n is, given a linked list of n labeled elements such that the labels strictly increase along the list, to mark each element of the list with its position within the list.

Lemma 6.3 (Bhatt *et al.*, 1989): Monotonic list ranking problems of size n can be solved on a (deterministic) COMMON or TOLERANT PRAM using $O(\log n/\log\log n)$ time, $O(n)$ operations and $O(n)$ space.

Theorem 6.4: For every fixed $k \in \mathbb{N}$, n integers in the range $1\,..\,n(\log n)^k$ can be sorted on a TOLERANT PRAM using $O(\log n/\log\log n)$ time, $O(n)$ operations and $O(n)$ space with probability at least $1 - 2^{-(\log n)^k}$.

Remark: Using Theorem 6.6, the probability bound of Theorem 6.4 can be improved for the COLLISION PRAM to $1 - 2^{-n^\epsilon}$, for some fixed $\epsilon > 0$.

Proof: Let $m = (\log n)^{k+6}$. By Lemma 2.8 and the principle of radix sorting, we can assume that the input numbers in fact belong to the smaller range $1\,..\,u$, where $u = n/m^2$; see (Rajasekaran and Reif, 1989, Section 3.1) for a fuller discussion of this approach. Now draw a random sample Y of the input elements by including each element with probability $1/m$ and independently of other elements. Compact the sample, chain-sort it using Theorem 6.2 and rank the resulting list using Lemma 6.3. By Chernoff bounds, $n/(2m) \le |Y| \le 2n/m$, except with negligible probability, so that these steps are not too expensive. Letting $b_i^Y = |B_i \cap Y|$, for $i = 1, \ldots, u$, take

$$\hat{b}_i = 2m \cdot b_i^Y + m^2,$$

for $i = 1, \ldots, u$. Again using Chernoff bounds, one easily shows that except with negligible probability, $b_i \le \hat{b}_i$, for all i, and $\sum_{i=1}^{u} \hat{b}_i = O(n)$. Now use a prefix summation to allocate a bucket Q_i of size $8\hat{b}_i$ to B_i, for $i = 1, \ldots, u$, such that Q_{i+1} borders Q_i on the right, for $i = 1, \ldots, u-1$.

The next task is to place the elements of B_i in Q_i, for $i = 1, \ldots, u$. This is done in a number of *stages*. All elements are initially *active*. In each stage, each active element in B_i chooses and attempts to write to a random cell in Q_i. If it does not collide, it is placed in the chosen cell and becomes inactive. We leave to the reader to show that $O(\log\log n)$ stages are sufficient to reduce the number of active elements below n/m. At this point the elements successfully placed in buckets can be collected in sorted order via a prefix summation, and the less than n/m remaining elements can be collected in nonsorted order via another prefix summation and sorted using the algorithms of Theorem 6.2 and Lemma 6.3. A final merge of the two sorted sequences completes the sorting. A processor allocation problem that was ignored above can easily be solved using Theorem 4.6. Since the number of active elements decreases geometrically over the stages, the total number of operations executed is $O(n)$. ■

Our final algorithm combines optimality with a running time of $O((\log^* n)^2)$, which causes major complications. Multiplicities must be estimated using Theorem 5.11, which means that the estimates obtained are not very reliable. Buckets must be allocated as described in Section 4, and colors cannot be handled independently, as far as the placement in buckets is concerned, since the failure probability for small color classes cannot be ignored. Instead it is necessary to monitor the progress of the colors throughout the process, pushing more resources towards colors that are not keeping pace with the rest.

Theorem 6.5: There is a constant $\epsilon > 0$ such that n integers in the range $1\,..\,n$ can be chain-sorted on a COLLISION PRAM using $O((\log^* n)^2)$ time, $O(n)$ operations and $O(n)$ space with probability at least $1 - 2^{-n^\epsilon}$.

Proof: Use the following algorithm:

Part 1: Use Theorem 5.11 to compute a coarse-estimator $\hat{b}_1, \ldots, \hat{b}_n$ for the input numbers.

Part 2: By Theorem 5.1, it is possible to obtain reliable estimates of large multiplicities. Furthermore, by an argument already used in the proof of Theorem 4.7, it is easy to place the elements of a large color class B_i in an array of size $O(b_i)$. Let us therefore without loss of generality assume that $\hat{b}_i \le n^{1/8}$, for $i = 1, \ldots, n$. Define $T, v_1, \ldots, v_T$ similarly as in Section 5, i.e., $v_T = \lceil n^{1/8} \rceil$, $v_{t-1} = \lceil \log v_t \rceil$, for $t = 2, \ldots, T$, and $v_1 = 1$. Also take $v_{T+1} = 2^{v_T}$. Then execute the following algorithm, where $K = 10 \cdot 2^8$.

```
(1)  Let all color classes and all elements be active;
(2)  for $i \in \{1, \ldots, n\}$ pardo $s_1[i] := K\hat{b}_i$;
(3)  for $t := 1$ to $T$ do
(4)    for each active color class $B_i$ pardo
(5)      begin
(6)        Allocate $v_t$ processors to each active element in $B_i$;
(7)        Allocate $v_t$ sectors of $\lceil s_t[i]/v_t \rceil$ cells each to $B_i$;
(8)        Scatter $B_i$ over each of its $v_t$ sectors;
(9)        Place each element of $B_i$ that is successful (does
(10)         not collide in every sector) and make it inactive;
(11)       if no elements in $B_i$ remain active
(12)       then make $B_i$ inactive;
(13)       $Tight_t[i] := (s_t[i] < Kv_{t+1}^3$ and $B_i$ is active);
(14)       if not $Tight_t[i]$ then
(15)         begin (* test scattering *)
(16)           Allocate $v_{t+1}$ cells to $B_i$ and let the elements
(17)             of $B_i$ carry out a conditional scattering $\mathcal{S}_i$
(18)             with probability $Kv_{t+1}^3/s_t[i]$ and range $v_{t+1}$;
(19)           if $\mathcal{S}_i$ has fullness 1 then $Tight_t[i] := true$;
(20)         end;
(21)       if $Tight_t[i]$
(22)       then $s_{t+1}[i] := Kv_{t+1}^2\hat{b}_i$ (* expand $B_i$ *)
(23)       else $s_{t+1}[i] := \lceil s_t[i]/2 \rceil$;
(24)     end;
```

Let *Stage* t, for $t = 1, \ldots, T$, be the tth execution of lines (4)–(24). For $t = 1, \ldots, T$ and $i = 1, \ldots, n$, let $N_t[i]$ denote the number of active elements in B_i at the start of Stage t. Also say that B_i *has sufficient space* in Stage t if $s_t[i] \ge HN_t[i] \cdot v_t$, where $H = 2^8$, and that B_i is *expanded* in Stage t if the assignment $s_{t+1}[i] := Kv_{t+1}^2\hat{b}_i$ in line (22) is carried out in Stage t. Finally, the scattering of B_i in Stage t *succeeds* if $N_{t+1}[i] \le N_t[i] \cdot v_{t+1}^{-4}$.

We now bound the probability that a given color class is expanded in a given stage. Suppose that B_i is expanded in Stage $t \ge 4$. Then either (1) B_i has insufficient space in Stage t, or (2) the scattering of B_i in Stage t does not succeed, although B_i has sufficient space in Stage t, or (3) B_i is expanded in Stage t, although B_i has sufficient space in Stage t and the scattering of B_i in Stage t succeeds.

Case 1: B_i has insufficient space in Stage t, i.e., $s_t[i] < HN_t[i] \cdot v_t$. We divide into two subcases:

Case 1a: B_i is expanded in Stage $t-1$, i.e., $s_t[i] = Kv_t^2\hat{b}_i$. Then $Kv_t^2\hat{b}_i < HN_t[i] \cdot v_t \le Hb_iv_t$, i.e.,

$b_i > (K/H)v_t\hat{b}_i = 10v_t\hat{b}_i$. By property (B) of a coarse-estimator, the probability of this is at most $2^{-10v_t} \le v_{t+1}^{-10}$.

Case 1b: B_i is not expanded in Stage $t-1$. Then a scattering $\mathcal{S}_i$ is carried out in Stage $t-1$, and the fullness of $\mathcal{S}_i$ is below 1. By Lemma 3.1(c), the probability of this is at most

$$\begin{aligned} v_t \cdot 2^{-N_t[i]\cdot(Kv_t^3/s_{t-1}[i])/v_t} &\le v_t \cdot 2^{-KN_t[i]\cdot v_t^2/(2s_t[i])} \\ &\le v_t \cdot 2^{-Kv_t/(2H)} \le v_{t+1}^{-4}. \end{aligned}$$

Altogether, Case 1 occurs with probability at most $2v_{t+1}^{-4}$.

Case 2: Now B_i has sufficient space in Stage t, i.e., $s_t[i] \ge HN_t[i]\cdot v_t$. Then

$$E(N_{t+1}[i]) \le N_t[i]\left(\frac{N_t[i]\cdot v_t}{s_t[i]}\right)^{v_t} \le N_t[i]\cdot H^{-v_t} \le N_t[i]\cdot v_{t+1}^{-8}$$

and hence $\Pr(N_{t+1}[i] > N_t[i]\cdot v_{t+1}^{-4}) \le v_{t+1}^{-4}$. It follows that Case 2 occurs with probability at most v_{t+1}^{-4}.

Case 3: Now B_i has sufficient space in Stage t and the scattering of B_i in Stage t succeeds, i.e., $s_t[i] \ge HN_t[i]\cdot v_t$ and $N_{t+1} \le N_t[i]\cdot v_{t+1}^{-4}$. If $s_t[i] < Kv_{t+1}^3$, then

$$N_{t+1}[i] \le N_t[i]\cdot v_{t+1}^{-4} \le \frac{s_t[i]}{Hv_tv_{t+1}^4} \le \frac{10}{v_tv_{t+1}} < 1,$$

i.e., B_i is inactive after Stage t. Hence Case 3 always involves the execution of a scattering $\mathcal{S}_i$. By Lemma 3.1(e), the probability that the fullness of $\mathcal{S}_i$ equals 1 is at most

$$\begin{aligned} \left(N_{t+1}[i]\cdot\frac{Kv_{t+1}^3}{s_t[i]}\cdot\frac{e}{v_{t+1}}\right)^{v_{t+1}} &\le \left(\frac{N_t[i]}{v_{t+1}}\cdot\frac{K\cdot e}{HN_t[i]\cdot v_tv_{t+1}}\right)^{v_{t+1}} \\ \le \left(\frac{Ke}{H}\cdot\frac{1}{v_tv_{t+1}^2}\right)^{v_{t+1}} &\le \left(\frac{10e}{5\cdot(17)^2}\right)^{v_{t+1}} \le 2^{-4v_{t+1}} \le v_{t+1}^{-4}. \end{aligned}$$

Hence Case 3 occurs with probability at most v_{t+1}^{-4}. Summing up, we have argued that the probability that B_i is expanded in Stage $t \ge 4$ is at most $4v_{t+1}^{-4} \le v_{t+1}^{-3}$; in particular, no expansion takes place in Stage T, except with negligible probability. Since the amount of space allocated to B_i in Stage t is $O(v_{t+1}^2\hat{b}_i)$, it is easy to see that the expected amount of space allocated to B_i over the course of the algorithm is $O(\hat{b}_i)$, and hence, by property (A) of a coarse-estimator, that the expected total amount of space used by the algorithm is $O(n)$. By a martingale argument, the actual space requirements of the algorithm are $O(n)$, except with negligible probability. It also follows from the analysis that, except with negligible probability, all elements are successfully placed, and it is clear that the algorithm can be executed within the stated time bounds.

The allocations of space and processors in the algorithm can be carried out using Theorem 4.10. It is easy to see that the total number of other operations executed by the algorithm, exclusive of those caused by color classes with insufficient space, is at most proportional to the total amount of space allocated. But the number of operations caused by a color class B_i in Stage t is $O(v_tb_i)$, whereas the probability that B_i has insufficient space in Stage t was shown above to be at most $2v_{t+1}^{-4}$. It now follows as in the case of space and using an argument as in the proof of Theorem 5.11 that the total number of operations executed is $O(n)$, except with negligible probability.

Part 3: Contrary to our informal introductory description, the buckets allocated by this last algorithm are not contiguous blocks of memory; instead each bucket is spread over $O(\log^* n)$ contiguous *segments*. It is easy, for each bucket, to chain together its constituent segments into a linked list, with segments containing no elements left out of the list. Furthermore, using Lemma 4.11(a), the lists corresponding to nonempty color classes can be concatenated in ascending order to a list L_1. Another application of Lemma 4.11(a), this time to an array of size $O(n)$ containing all segments, provides another list L_2 that chains together all elements. Each element j now determines its successor in the final output list as follows: It is the successor of j in L_2 if this successor belongs to the same segment S as j; otherwise it is the first element in the segment following S in L_1. This completes the proof of Theorem 6.5. ∎

With little extra effort, our chain-sorting algorithm can be modified to deliver its output in a different format that also appears to be useful. We first need two definitions.

A *padded representation* of size $s \ge n$ of a sequence $x_1, \ldots, x_n$ is a vector of size s whose non-*nil* elements, taken in order, precisely form the sequence $x_1, \ldots, x_n$. Here *nil* is a special value that cannot occur as an element of a sequence. Given a sequence $x_1, \ldots, x_n$, say that a permutation $\pi_1, \ldots, \pi_n$ of $1, \ldots, n$ is *duplicate-grouping* for $x_1, \ldots, x_n$ if for all $i, j, k \in \mathbb{N}$, if $1 \le i \le j \le k \le n$ and $x_{\pi_i} = x_{\pi_k}$, then $x_{\pi_j} = x_{\pi_i}$. Intuitively, a duplicate-grouping permutation brings together duplicates without necessarily sorting the elements.

Theorem 6.6: There is a constant $\epsilon > 0$ such that the following problem can be solved on a COLLISION PRAM using $O((\log^* n)^2)$ time, $O(n)$ operations and $O(n)$ space with probability at least $1 - 2^{-n^\epsilon}$: Given a sequence $x_1, \ldots, x_n$ of n integers in the range $1\,..\,n$, compute a padded representation of size $O(n)$ of a duplicate-grouping permutation for $x_1, \ldots, x_n$.

Proof: As described in the last paragraph of its correctness proof, the algorithm of Theorem 6.5 spreads the elements of each color class over $O(\log^* n)$ segments. During the execution of the algorithm, it is easy, for each bucket, to compute the prefix sums of the sizes of the segments constituting the bucket. When the algorithm terminates, Theorem 4.10 can therefore be used to allocate a single contiguous segment of the required size to each color class, after which the elements of the color class can be trivially moved to the new segment. This produces the desired output. ∎

As an application of Theorem 6.6, we sketch an improvement of a recent result due to MacKenzie and Stout (1991) concerning the following problem, which they call *padded sorting*: Given n independent random numbers $r_1, \ldots, r_n$ drawn from the uniform distribution over the interval $(0, 1]$, compute a padded representation of size $n + o(n)$ of a sequence containing the elements $r_1, \ldots, r_n$ in nondecreasing order. MacKenzie and Stout show that padded sorting can be carried out optimally in expected time $O(\log\log n)$, which we improve to $O(\log\log n/\log\log\log n)$. The same result was obtained independently by MacKenzie and Stout (MacKenzie, personal communication, April 1991).

Theorem 6.7: For every fixed $k \in \mathbb{N}$, the following problem can be solved on a COLLISION PRAM using $O(\log\log n/\log\log\log n)$ time, $O(n)$ operations and $O(n)$ space with probability at least $1 - 2^{-(\log n)^k}$: Given $n \ge 2$ independent random numbers $r_1, \ldots, r_n$ drawn from the uniform distribution over the interval $(0,1]$, compute a padded representation of size at most $n(1+(\log n)^{-k})$ of a sequence containing the elements $r_1,\ldots,r_n$ in nondecreasing order.

Proof: For $j=1,\ldots,n$, let $x_j=\lceil nr_j\rceil$. As usual, for $i=1,\ldots,n$, let $B_i=\{j: 1\le j\le n$ and $x_j=i\}$ and $b_i=|B_i|$ and call B_i a color class. As follows easily from Chernoff bound (c), $\max\{b_i: 1\le i\le n\}=(\log n)^{O(1)}$, except with negligible probability. Hence if the algorithm of Theorem 6.5 is modified to operate in its last stage with $v_T=\lceil\log n\rceil^c$ instead of $v_T=\lceil n^{1/8}\rceil$, for a suitable constant $c\in\mathbb{N}$, the correspondingly modified algorithm of Theorem 6.6 with high probability places each color class in a segment of size $(\log n)^{O(1)}$. Standard prefix summation techniques can now be used to compute $b_1,\ldots,b_n$ and to place the elements of each color class in consecutive positions. Use Theorem 4.12 to allocate b_i^2 (virtual) processors to i, for $i=1,\ldots,n$. These processors can sort $R_i=\{r_j: j\in B_i\}$ using $O(\log b_i/\log\log b_i)$ time and $O(b_i^2)$ operations, for $i=1,\ldots,n$. Since it is known that $\sum_{i=1}^n b_i^2=O(n)$, except with negligible probability, the sorting of $R_1,\ldots,R_n$ altogether uses $O(\log\log n/\log\log\log n)$ time and $O(n)$ operations.

Now choose $t\in\mathbb{N}$ with $t\ge(\log n)^{k+1}$, but $t=(\log n)^{O(1)}$. For $l=1,\ldots,\lfloor n/t^3\rfloor$, $S_l=\sum_{i=(l-1)t^3+1}^{lt^3} b_i$ is binomially distributed with expected value t^3. Hence by Chernoff bound (a), $\Pr(S_l\ge(1+t^{-1})t^3)\le e^{-t/3}$. It follows that $\max\{S_l: 1\le l\le\lfloor n/t^3\rfloor\}\le(1+t^{-1})t^3$, except with negligible probability. Using a prefix summation of $b_{(l-1)t^3+1},\ldots,b_{lt}$, it is now easy to place the elements of $\bigcup_{i=(l-1)t^3+1}^{lt^3} R_i$ in sorted order in an array of size $\lfloor(1+t^{-1})t^3\rfloor$, for $l=1,\ldots,\lfloor n/t^3\rfloor$, and to place the elements of $\bigcup_{i=\lfloor n/t^3\rfloor t^3+1}^{n} R_i$ in sorted order in an array of size $(\log n)^{O(1)}$. The total space used is $\lfloor n/t^3\rfloor\cdot\lfloor(1+t^{-1})t^3\rfloor+(\log n)^{O(1)}$, which for sufficiently large values of n is bounded by $n(1+(\log n)^{-k})$. ∎

7 Nonoptimal algorithms

This section investigates the effect for the problems considered of allowing slightly superlinear processor and space bounds. In most cases, we also have to generalize the problems by introducing a so-called *slack parameter*. Throughout this section, we assume the input size n to be larger than some (unspecified) constant.

Definition: For $n,m\in\mathbb{N}$, $d_1,\ldots,d_m\in\mathbb{R}$ and $\lambda\ge 1$, the *multi-compaction* problem of size n and with parameters $d_1,\ldots,d_m$ and slack λ is the following: Given n integers $x_1,\ldots,x_n$ in the range $0\,..\,m$ such that for $i=1,\ldots,m$, $|\{j: 1\le j\le n$ and $x_j=i\}|\le d_i$, compute n nonnegative integers $y_1,\ldots,y_n$ such that

(1) For $j=1,\ldots,n$, $x_j=0\Leftrightarrow y_j=0$;

(2) For $1\le i<j\le n$, if $x_i=x_j\ne 0$, then $y_i\ne y_j$;

(3) For $i=1,\ldots,n$, $\max\{y_j: 1\le j\le n$ and $y_j=i\}=O(\lambda d_i)$.

Theorem 7.1: For all fixed $k\in\mathbb{N}$, there is a constant $\epsilon>0$ such that for $m=(\log n)^{O(1)}$, multi-compaction problems of size n with parameters $d_1,\ldots,d_m$ and with slack $\log^{(k)}n$ can be solved on a TOLERANT PRAM using constant time, $O(n\log^{(k)}n)$ processors and $O(n\log^{(k)}n)$ space with probability at least $1-2^{-n^\epsilon}$.

Proof: As the proof of Theorem 4.7, except that Theorem 4.3 is used only a constant number of times. ∎

Definition: For $n \in \mathbb{N}$ and $\lambda \geq 1$, the interval allocation problem of size n and with slack λ is the following: Given n nonnegative integers $x_1, \ldots, x_n$, compute n nonnegative integers $y_1, \ldots, y_n$ such that

(1) For $j = 1, \ldots, n$, $x_j = 0 \Leftrightarrow y_j = 0$;

(2) For $1 \leq i < j \leq n$, if $0 \notin \{x_i, x_j\}$, then $\{y_i, \ldots, y_i + x_i - 1\} \cap \{y_j, \ldots, y_j + x_j - 1\} = \emptyset$;

(3) $\max\{y_j : 1 \leq j \leq n\} = O(\lambda \sum_{j=1}^{n} x_j)$.

Theorem 7.2: For every fixed $k \in \mathbb{N}$, there is a constant $\epsilon > 0$ such that interval allocation problems of size n and with slack $\log^{(k)} n$ can be solved on a TOLERANT PRAM using constant time, $O(n \log^{(k)} n)$ processors and $O(n \log^{(k)} n)$ space with probability at least $1 - 2^{-n^\epsilon}$.

Proof: As the proof of Theorem 4.10, except that Theorem 7.1 is used in place of Theorem 4.7. ∎

Definition: For $n \in \mathbb{N}$ and $\lambda \geq 1$, the interval marking problem of size n and with slack λ is the following: Given n nonnegative integers $x_1, \ldots, x_n$ with $\sum_{j=1}^{n} x_j = O(n)$, compute nonnegative integers $s, z_1, \ldots, z_s$ with $s = O(\lambda n)$ such that

(1) For all integers i, j, k with $1 \leq i \leq j \leq k \leq s$, if $z_i = z_k \neq 0$, then $z_j = z_i$;

(2) For $i = 1, \ldots, n$, $|\{j : 1 \leq j \leq s \text{ and } z_j = i\}| = x_i$.

Theorem 7.3: For every fixed $k \in \mathbb{N}$, there is a constant $\epsilon > 0$ such that interval marking problems of size n and with slack $\log^{(k)} n$ can be solved on a TOLERANT PRAM using constant time, $O(n \log^{(k)} n)$ processors and $O(n \log^{(k)} n)$ space with probability at least $1 - 2^{-n^\epsilon}$.

Proof: As the proof of Theorem 4.12, using Theorem 7.2 in place of Theorem 4.10 and part (b) of Lemma 4.11 in place of part (a). ∎

Theorem 7.4: For every fixed $k \in \mathbb{N}$, there is a constant $\epsilon > 0$ such that coarse-estimation problems of size and width n can be solved on a TOLERANT PRAM using $O(\log^* n)$ time, $O(n \log^{(k)} n)$ processors and $O(n \log^{(k)} n)$ space with probability at least $1 - 2^{-n^\epsilon}$.

Proof: As the proof of Lemma 5.2, except that Theorems 7.2 and 7.3 are used in place of Theorems 4.10 and 4.12. ∎

Definition: Let $n, m \in \mathbb{N}$, and let $x_1, \ldots, x_n$ be n integers in the range $1 \,..\, m$. For $i = 1, \ldots, m$, let $b_i = |\{j : 1 \leq j \leq m \text{ and } x_j = i\}|$. For $\lambda \geq 1$, a *coarse-estimator* for $x_1, \ldots, x_n$ of width m and with slack λ is a sequence of m independent nonnegative random variables $\hat{b}_1, \ldots, \hat{b}_m$ such that

(A) $\sum_{i=1}^{m} \hat{b}_i = O(\lambda n)$;

(B) For $i = 1, \ldots, m$ and for all $a \geq 1$, $\Pr(b_i > a\hat{b}_i) \leq 2^{-a}$.

Theorem 7.5: For every fixed $k \in \mathbb{N}$, there is a constant $\epsilon > 0$ such that coarse-estimation problems of size and width n and with slack $\log^{(k)} n$ can be solved on a TOLERANT PRAM using constant time, $O(n \log^{(k)} n)$ processors and $O(n \log^{(k)} n)$ space with probability at least $1 - 2^{-n^\epsilon}$.

Proof: Execute only Stages $t_0, \ldots, T$ of the algorithm of Theorem 7.4, where $t_0 = T - k - 1$. The proof of Lemma 5.4 shows that for $i = 1, \ldots, n$ and $t = t_0 + 1, \ldots, T$, $\Pr(i \text{ is tardy in Stage } t) \le v_t^{-7}$. Since $v_{t_0} = O(\log^{(k+1)} n)$, it now follows as in the proof of Lemmas 5.5 and 5.7 that the total amount of space and processors allocated is $O(n \log^{(k)} n)$ and that $\sum_{i=1}^{n} \tilde{b}_i = O(n \log^{(k)} n)$. The remaining argumentation is as in the proof of Lemma 5.2. ∎

Theorem 7.6: For every fixed $k \in \mathbb{N}$, there is a constant $\epsilon > 0$ such that n integers in the range $1 \,.\,.\, n$ can be chain-sorted on a TOLERANT PRAM using constant time, $O(n \log^{(k)} n)$ processors and $O(n \log^{(k)} n)$ space with probability at least $1 - 2^{-n^\epsilon}$.

Proof: Execute only Stages $t_0, \ldots, T$ of the algorithm of Theorem 6.5, where $t_0 = T - k - 2$, and use Theorem 7.2 in place of Theorem 4.10. The proof of Theorem 6.5 shows that the probability that a fixed color class is expanded in Stage $t \ge \max\{4, t_0 + 1\}$ is bounded by v_{t+1}^{-3}. Since $v_{t_0+1} = O(\log^{(k+1)} n)$, the claim now follows as in the proof of Theorem 6.5. ∎

Theorem 7.7: There is a constant $\epsilon > 0$ such that the following problem can be solved on a TOLERANT PRAM using constant time, $O(n \log^{(k)} n)$ processors and $O(n \log^{(k)} n)$ space with probability at least $1 - 2^{-n^\epsilon}$: Given a sequence $x_1, \ldots, x_n$ of n integers in the range $1 \,.\,.\, n$, compute a padded representation of size $O(n \log^{(k)} n)$ of a duplicate-grouping permutation for $x_1, \ldots, x_n$.

Proof: As the proof of Theorem 6.6, using Theorem 7.6 instead of Theorem 6.5 and Theorem 7.2 instead of Theorem 4.10. ∎

Acknowledgment: Parts of Sections 4 and 5 resulted from a joint investigation with Holger Bast. I thank Peter Miltersen for pointing me to the papers by Stockmeyer and Ajtai and Ben-Or. An earlier version of the present paper achieved running times of $O(\log \log n \log^* n / \log \log \log n)$, the bottleneck being essentially compaction. After receiving a preliminary sketch of the algorithm of Theorem 4.10, Joseph Gil informed the author of the results of Matias and Vishkin and also observed that the prefix sums of few small integers can be computed in constant time (Lemma 2.4), thus removing the last obstacle to interval allocation in $O(\log^* n)$ time. The interval allocation results described in this paper are improvements and generalizations of Gil's result.

References

AJTAI, M., AND BEN-OR, M. (1984), A Theorem on Probabilistic Constant Depth Computations, *in* Proc. 16th Annual ACM Symposium on Theory of Computing, pp. 471–474.

BAST, H., AND HAGERUP, T. (1991), Fast and Reliable Parallel Hashing, *in* Proc. 3rd Annual ACM Symposium on Parallel Algorithms and Architectures, to appear.

BEAME, P., AND HASTAD, J. (1989), Optimal Bounds for Decision Problems on the CRCW PRAM, *J. ACM* **36**, pp. 643–670.

BERKMAN, O., AND VISHKIN, U. (1989), Recursive *-Tree Parallel Data-Structure, *in* Proc. 30th Annual Symposium on Foundations of Computer Science, pp. 196–202.

Bhatt, P. C. P., Diks, K., Hagerup, T., Prasad, V. C., Radzik, T., and Saxena, S. (1989), Improved Deterministic Parallel Integer Sorting, *Inform. and Comput.*, to appear.

Bollobás, B. (1987), Martingales, Isoperimetric Inequalities and Random Graphs, *in* Colloq. Math. Soc. J. Bolyai 52, pp. 113–139.

Chlebus, B. S., Diks, K., Hagerup, T., and Radzik, T. (1989), New Simulations between CRCW PRAMs, *in* Proc. 7th International Conference on Fundamentals of Computation Theory, Springer Lecture Notes in Computer Science, Vol. 380, pp. 95–104.

Cole, R., and Vishkin, U. (1989), Faster Optimal Parallel Prefix Sums and List Ranking, *Inform. and Comput.* **81**, pp. 334–352.

Eppstein, D., and Galil, Z. (1988), Parallel Algorithmic Techniques for Combinatorial Computation, *Ann. Rev. Comput. Sci.* **3**, pp. 233–283.

Gavril, F. (1975), Merging with Parallel Processors, *Comm. ACM* **18**, pp. 588–591.

Hagerup, T., and Radzik, T. (1990), Every Robust CRCW PRAM Can Efficiently Simulate a Priority PRAM, *in* Proc. 2nd Annual ACM Symposium on Parallel Algorithms and Architectures, pp. 117–124.

Hagerup, T., and Rüb, C. (1990), A Guided Tour of Chernoff Bounds, *Inf. Proc. Lett.* **33**, pp. 305–308.

Hagerup, T. (1991), Fast and Optimal Simulations between CRCW PRAMs, manuscript.

MacKenzie, P. D., and Stout, Q. F. (1991), Ultra-Fast Expected Time Parallel Algorithms, *in* Proc. 2nd Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 414–423.

Matias, Y., and Vishkin, U. (1991), Converting High Probability into Nearly-Constant Time — with Applications to Parallel Hashing, *in* Proc. 23rd Annual ACM Symposium on Theory of Computing, pp. 307–316.

Ragde, P. (1990), The Parallel Simplicity of Compaction and Chaining, *in* Proc. 17th International Colloquium on Automata, Languages and Programming, Springer Lecture Notes in Computer Science, Vol. 443, pp. 744–751.

Rajasekaran, S., and Reif, J. H. (1989), Optimal and Sublogarithmic Time Randomized Parallel Sorting Algorithms, *SIAM J. Comput.* **18**, pp. 594–607.

Raman, R. (1990), The Power of Collision: Randomized Parallel Algorithms for Chaining and Integer Sorting, *in* Proc. 10th Conference on Foundations of Software Technology and Theoretical Computer Science, Springer Lecture Notes in Computer Science, Vol. 472, pp. 161–175.

Raman, R. (1991), Optimal Sub-logarithmic Time Integer Sorting on the CRCW PRAM, Tech. Rep. no. 370, Univ. of Rochester.

Reischuk, R. (1985), Probabilistic Parallel Algorithms for Sorting and Selection, *SIAM J. Comput.* **14**, pp. 396–409.

Stockmeyer, L. (1983), The Complexity of Approximate Counting, *in* Proc. 15th Annual ACM Symposium on Theory of Computing, pp. 118–126.

Stockmeyer, L., and Vishkin, U. (1984), Simulation of Parallel Random Access Machines by Circuits, *SIAM J. Comput.* **13**, pp. 409–422.