# Optimal deterministic approximate parallel prefix sums and their applications *

*Tal Goldberg*
Dept. of Computer Science
Tel Aviv university
Tel Aviv 69978, Israel

*Uri Zwick* †
Dept. of Computer Science
Tel Aviv university
Tel Aviv 69978, Israel

**Abstract**

We show that extremely accurate approximation to the prefix sums of a sequence of $n$ integers can be computed *deterministically* in $O(\log\log n)$ time using $O(n/\log\log n)$ processors in the COMMON CRCW PRAM model. This complements randomized approximation methods obtained recently by Goodrich, Matias and Vishkin and improves previous deterministic results obtained by Hagerup and Raman. Furthermore, our results completely match a lower bound obtained recently by Chaudhuri. Our results have many applications. Using them we improve upon the best known time bounds for deterministic approximate selection and for deterministic padded sorting.

## 1 Introduction

The computation of prefix sums is one of the most basic tools in the design of fast parallel algorithms (see Blelloch [9] and JáJá [33]). Prefix-sums can be computed in $O(\log n)$ time and linear work in the EREW PRAM model (Ladner and Fischer [34]) and in $O(\log n/\log\log n)$ and linear work in the COMMON CRCW PRAM model (Cole and Vishkin [14]). These two results are optimal as shown by lower bounds of Cook, Dwork and Reischuk [15] and of Beame and Håstad [8], even if randomization is allowed.

One of the typical uses of prefix sums in parallel computation is in the solution of *compaction* problems which are required, in turn, for the solution of *processor allocation* problems. Recent years have seen tremendous progress in the development of parallel algorithms with extremely fast running times. Running times of $O(\log\log n)$ and $O(\log^* n)$ are now typical (see the works of Gil, Goodrich, Hagerup, Matias, Raman, Vishkin [13],[20],[21],[23],[24],[25],[28],[38],[42]).

It is clear that prefix sums cannot be used by such fast algorithms. To obtain these extremely fast algorithms, new primitives that replace parallel prefix sums had to be invented. One such primitive, used in many of the previously cited works, is *linear approximate compaction*: given an array $A$ of size $n$ that contains $k$ non-zero elements, create an array $B$ of size $(1+\epsilon)k$ that contains the non-zero elements of $A$, where $\epsilon = o(1)$ (with respect to $n$). Unused cells in $B$ should contain a special *null* value.

Gil, Matias and Vishkin [20], Goodrich [21] and Hagerup [24] have shown that linear approximate compaction can be solved, with high probability, in $O(\log^* n)$ time using $O(n/\log^* n)$ processors in the *randomized* CRCW PRAM model. This result is work-time optimal as shown by a lower bound of MacKenzie [36].

Using the tool of linear approximate compaction (and some other tools), many interesting problems were shown to have extremely fast *randomized* algorithms. Among these problems are parallel dictionaries, parallel hashing, padded sorting and padded integer sorting, and approximate and exact selection.

Goodrich, Matias and Vishkin [22],[23] and Hagerup and Raman [29] have recently shown that *consistent* approximate prefix sums can be computed, with high probability, in $O(\log^* n)$ parallel time and linear work using a *randomized* algorithm. A sequence $0 = b_0, b_1, \ldots, b_n$ is said to be a *consistent $\epsilon$-approximate prefix sums sequence* of a sequence $a_1, a_2, \ldots, a_n$, if for every $1 \le i \le n$ we have $\sum_{j=1}^{i} a_j \le b_i \le (1+\epsilon)\sum_{j=1}^{i} a_j$ (approximation) and $b_i - b_{i-1} \ge a_i$ (consistency). Consistent approximate prefix sums

*Work supported in part by **The basic research foundation** administrated by **The Israel academy of sciences and humanities**.

†E-mail address: zwick@math.tau.ac.il.

give a much 'cleaner' solution to the linear approximate compaction problem. They solve in fact the seemingly harder *ordered approximate compaction* in which the order of the elements should be preserved.

Can such fast running times be obtained without randomization? Hagerup [27] obtained a deterministic $O((\log\log n)^3)$ time linear work algorithm for the linear approximate compaction problem. Hagerup and Raman [29] obtained a deterministic $O((\log\log n)^4/\log\log\log n)$ time linear work algorithm for the approximate prefix sums problem. Chaudhuri [11] has recently obtained a lower bound of $\Omega(\log\log n)$ on the time needed to solve a linear approximate problem of size $n$ using $O(n)$ processors. Chaudhuri's result shows that randomization is essential if a running time of $O(\log^* n)$ is desired. Hagerup's result shows that although running times of $O(\log^* n)$ cannot be attained by deterministic algorithms, extremely fast running times, much below $O(\log n)$, are still possible.

In this work, we show that consistent $\epsilon$-approximate prefix sums, where $\epsilon = (\log\log n)^{-a}$ for any fixed $a > 0$, can be computed *deterministically* in $O(\log\log n)$ time using $O(n/\log\log n)$ processors in the COMMON CRCW PRAM model. The algorithm given is *uniform*. In other words, the same algorithm is used for every input size and no precomputed tables are used by the algorithm. As a consequence we get that ordered approximate compaction, and also ordered processor allocation, can be solved *deterministically* within the same bounds. This greatly improves the results of Hagerup [27] and Hagerup and Raman [29] and completely matches the lower bound of Chaudhuri [11].

Hagerup (personal communication to [11]) had obtained a *non-uniform* deterministic approximate compaction algorithm that runs in $O(\log\log n)$ time using $O(n/\log\log n)$ processors. All the algorithms described in this paper are uniform.

The 'engine' of our results in a construction by Ajtai [2] of *uniform* constant depth and polynomial size circuits for approximate counting. We translate Ajtai's result to the PRAM language and obtain a constant time approximate counting algorithm that uses a polynomial number of processors. This approximate counting algorithm is used as a building block in all our subsequent algorithms. This may be viewed as a 'positive' contribution of circuit complexity to the world of parallel algorithms. The 'negative' contribution being the proof that exact counting cannot be done in constant time using a polynomial number of processors ([1],[18],[43],[30]). The $\Omega(\log n/\log\log n)$ lower bound of Beame and Håstad [8] is essentially a formulation of this result in the PRAM language.

The fact that Ajtai's result [2] is *uniform* and *explicit* should be stressed. Non-explicit constant depth polynomial size circuits for approximate counting can be easily obtained using probabilistic arguments (see Ajtai and Ben-Or [3]). To achieve uniformity Ajtai simulates short random walks in explicitly constructed expander graphs. Explicit constructions of appropriate expander graphs were obtained by Galil and Gaber [19] and Lubotzky, Phillips and Sarnak [35].

The rest of the paper is organized as follows. In the next section we translate Ajtai's result into the PRAM setting. In Section 3 we combine Ajtai's engine with ideas of Goodrich, Matias and Vishkin and show that consistent approximate prefix sums can be computed deterministically in $O(1)$ time using $n^{O(1)}$ processors. In Section 4, we use standard methods to achieve our time-work optimal result. We show that consistent approximate prefix sums can be computed deterministically in $O(\log\log n)$ time using $O(n/\log\log n)$ processors. In Section 5 we derive the corresponding bounds for approximate compaction, and for the interval and processor allocation problems. In Section 6, we state the improved time bounds for approximate selection and for padded sorting that are implied by our improved algorithms. Finally, in the last section we mention some possible further applications.

## 2 Deterministic approximate counting

The results of this paper are based on the following result that follows from the work of Ajtai [2].

**Theorem 2.1** *Let $A$ be a Boolean array of size $n$. For any fixed $a > 0$, the number of 1's in $A$ can be approximated to within a factor of $1 + (\log n)^{-a}$ in constant time using a polynomial number of processors in the* COMMON CRCW PRAM *model.*

Ajtai did not consider PRAMs in his work. He showed instead that there exists a uniform family of $AC^0$ circuits that can perform approximate counting with the desired accuracy. In his proof Ajtai uses yet another different formulation of the problem, that of first-order definability over finite structures (see Ajtai [1]).

The three different settings, are easily shown to be equivalent. The equivalence of first-order definability and uniform families of $AC^0$ is explained, for example, by Immerman [32]. The equivalence of uniform families of $AC^0$ and CRCW PRAMs is demonstrated, for example, by Stockmeyer and Vishkin [41].

Though these three different setting are easily shown to be equivalent, the fact that the result of Ajtai is not stated in the PRAM model may explain the fact that his result has been ignored so far by the designers of PRAM algorithms. We think, therefore, that a direct implementation of Ajtai's approximate counting algorithm in the CRCW PRAM model is of some value. Furthermore, while examining the implementation of Ajtai's algorithm we find out that certain preliminary stages in Ajtai's algorithm were independently discovered by the designers of PRAM algorithms.

In the remainder of this section we sketch a proof of Theorem 2.1. A complete proof, supplemented by the exact implementation details, will appear in the full version of the paper.

**Proof of Theorem 2.1 (sketch) :**

Let $A$ be a Boolean array of size $n$. Let $|A|$ denote the number of 1's in $A$. We mimic Ajtai's proof and obtain the approximate counting algorithm by establishing the following sequence of results:

1. For any fixed $a > 0$, the number $\min\{|A|, (\log n)^a\}$ can be determined in constant time using a polynomial number of processors.
2. For any fixed $\epsilon > 0$, if $|A| \le k$ then it is possible, in constant time using a polynomial number of processors, to compact the 1's of $A$ into an array $B$ of size $k^{1+\epsilon}$.
3. There is a (*uniform* and *explicit*) constant time algorithm that uses only a polynomial number of processors that outputs 0 if $|A| < n/\log^2 n$ and that outputs 1 if $|A| > n/3$.
4. An estimate $f(A)$ satisfying $|A| \le f(A) \le 2|A|$ can be computed in constant time using a polynomial number of processors.
5. For any fixed $a > 0$, an estimate $f(A)$ satisfying $|A| \le f(A) \le (1 + (\log|A|)^{-a}) \cdot |A|$ can be computed in constant time using a polynomial number of processors.
6. For any fixed $a > 0$, an estimate $f(A)$ satisfying $|A| \le f(A) \le (1+(\log n)^{-a}) \cdot |A|$ can be computed in constant time using a polynomial number of processors.

The first result, stating that counting up to powers of $\log n$ is possible in constant time, is an old result in circuit complexity (see e.g., Denenberg *et al.* [16], Fagin *et al.* [17], Håstad *et al.* [31] and Newman *et al.* [39]).

The second result is very similar to the compaction results of Ragde [40] and Hagerup [26]. Ragde had shown that the 1's in $A$ can be compacted in constant time into an array of size $k^2$ using $O(n^2)$ processors and into an array of size $k^4$ using $O(n)$ processors. Hagerup [26] strengthened this result and showed that for every fixed $\epsilon > 0$ the 1's in $A$ can be compacted, in constant time, into an array of size $k^{1+\epsilon}$ using $O(n)$ processors. Hagerup's result is therefore stronger than the required result, it uses a linear number of processors while a polynomial number is allowed. The proof given by Hagerup is extremely technical. Ajtai, on the other hand, gives a very elegant proof of the weaker claim which is sufficient for his purposes.

It is interesting to note that using the second result, we can obtain a direct proof of the first result. Simply compact the 1's in $A$ into an array $B$ of size $(\log n)^{a+1}$, using $\epsilon = 1/a$. If the compaction does not succeed, we know that the number of 1's in $A$ is at least $(\log n)^a$. If it does, all we have to do is to count the number of 1's in an array of size $(\log n)^{a+1}$. As we have $n^{O(1)}$ processors, this can be easily done in constant time (see, e.g., Cole and Vishkin [14]).

The second result implies that, for any $0 < \alpha < \beta < 1$, we can distinguish cases in which $|A| \le n^\alpha$ from cases in which $|A| \ge n^\beta$.

The third step is the major step on the road to accurate approximate counting. To obtain this result Ajtai simulates short random walks in expander graphs, a method introduced by Ajtai, Komlós and Szemerédi [4] (see also Alon and Spencer [6], p. 123).

**Lemma 2.2 ([4])** *Let $G = (V, E)$ be a $d$-regular graph on $n$ vertices. Let $A(G)$ be the adjacency matrix of $G$ and suppose that the second largest eigenvalue (in absolute value) of $A(G)$ has an absolute value of less than $(1-\epsilon)d$, for some $\epsilon > 0$. Then, for any $a > 0$, there exists $c > 0$ such that for any set $B \subseteq V$ with $|B| \ge n/3$, the probability that a random walk of length $s = c \log n$ in $G$ misses $B$ is at most $n^{-a}$.*

A random walk of length $s$ in a $d$-regular $G = (V, E)$ is obtained by choosing a random starting vertex (each vertex is chosen with probability $1/n$), and then selecting, with equal probabilities, one of the $d^s$ sequences that describe the edges that should be followed. If $s = c\log n$ then the number of such walks is polynomial.

Gaber and Galil [19] and Lubotzky, Phillips and Sarnak [35] describe explicit constructions of graphs that satisfy the conditions of Lemma 2.2. The graphs obtained by Gaber and Galil are 3-regular. The graphs obtained by Lubotzky *et al.* are 6-regular. These explicit constructions do not supply such graphs for every $n$, only for $n$'s which are powers of two in the first construction, and for prime $n$'s congruent to 1 modulo 4 in the second construction. This is a technical matter that will be ignored in this extended abstract. It can be shown, without much difficulty that, for any $c > 0$, all the walks of length $s = c\log n$ in these explicit graphs can be constructed in constant time using a polynomial number of processors *without* the use of any precomputed tables. A proof of this claim will appear in the full version of the paper.

Let $A$ be a Boolean array of size $n$. Let $G$ be an explicit $d$-regular graph on $n$ vertices that satisfies the requirements of the Lemma. We associate each cell of $A$ with a vertex in $G$. Let $B$ be the set of vertices of $G$ associated with cells of $A$ that contain 1.

There are only $nd^s = n^b$ walks in $G$ of length $s = c\log n$, where $b = 1+c\log d$. We can check, in constant time using a polynomial number of processors which of these walks passes through $B$.

If $|B| = |A| \geq n/3$ and if $c$ is large enough, then according to the Lemma (with $a = 2$) at most $n^{b-2}$ walks of length $s = c\log n$ miss $B$. In can be easily checked, on the other hand, that if $|B| = |A| \leq n/\log^2 n$, then at least $n^{b-1}$ of the walks of length $s = c\log n$ miss $B$. Using the second result we can distinguish between these two situations. This completes the proof of the third result.

We omit the proofs of the last three results although they also contain some interesting ideas. The implementation of these steps will be described in the full version of the paper.

## 3 Deterministic consistent approximate prefix sums

The previous section describes an approximate counting algorithm. It is easy, using standard techniques, to extend this algorithm into an approximate summation algorithm. The basic idea is very simple. Given $n$ integers, each represented using at most $O(\log n)$ bits, we approximate, for each $1 \leq i \leq O(\log n)$, the number of 1's among the $i$-th bits of the binary representations of the input numbers. We are then left with $O(\log n)$ numbers, each containing $O(\log n)$ bits. The exact sum of these numbers can be easily found in constant time using a polynomial number of processors. This gives us the following result:

**Theorem 3.1** *Let $A$ be a Boolean array of size $n$ each of whose elements is an $O(\log n)$-bit integer. For any fixed $a > 0$, the sum $\sum_{i=1}^{n} a_i$ (where $a_i$ is the $i$-th element of $A$) can be approximated to within a factor of $1 + (\log n)^{-a}$ in constant time using a polynomial number of processors in the* COMMON CRCW PRAM *model.*

The prefix sums of a sequence $a_1, a_2, \ldots, a_n$ are the elements of the sequence $0 = b_0, b_1, \ldots, b_n$, where $b_i = \sum_{j=1}^{i} a_i$. Once we know how to approximate a sum, it is easy to approximate each element of the prefix sums sequence. However, for such an approximation to be of value, it should satisfy a natural consistency condition introduced by Goodrich, Matias and Vishkin [22],[23] and by Hagerup and Raman [28].

A sequence $0 = b_0, b_1, \ldots, b_n$ is said to be a *consistent $\epsilon$-approximate prefix sums sequence* of a sequence $a_1, a_2, \ldots, a_n$, if for every $1 \leq i \leq n$ we have $\sum_{j=1}^{i} a_j \leq b_i \leq (1+\epsilon)\sum_{j=1}^{i} a_j$ (approximation) and $b_i - b_{i-1} \geq a_i$ (consistency).

Goodrich *et al.* [22],[23] and Hagerup and Raman [29] show how to compute consistent $2\epsilon\log n$-approximate prefix sums by calls to an $\epsilon$-approximate summation algorithm. The presentation in [22],[23] is complicated by the fact that their approximate summation 'box' is randomized. The fact that we compute approximate sums deterministically simplifies matters. We therefore sketch their reduction.

Start by 'planting' a complete binary tree whose leaves are the elements of the sequence $a_1, a_2, \ldots, a_n$ (we assume for simplicity that $n$ is a power of 2). For each vertex $v$ in the tree compute an $\epsilon$-approximate estimation $S'(v)$ of the sum of the elements that appear in the leaves of $v$'s subtree. The estimates corresponding to vertices at level $i$ (where leaves are considered to be at level 0) are multiplied by $(1+\epsilon)^i$. Let $\bar{S}(v)$ denote the value now contained in vertex $v$.

It is easy to check that the values $\bar{S}(v)$ associated with the vertices of the tree are now consistent, in the sense that if $v_1$ and $v_2$ are the two children of $v$ then $\bar{S}(v) \geq \bar{S}(v_1) + \bar{S}(v_2)$.

The exact prefix sum $\sum_{j=1}^{i} a_j$, for any $1 \leq j \leq n$, can be obtained by summing the exact sums corresponding to at most $\log n$ vertices of tree. Take the approximation $b_i$ of this prefix sum to be the exact sum of the approximations to values of these vertices. As this sum involves only $O(\log n)$ numbers, it can be easily computed in constant time using a polynomial number of processors. It can be easily shown that the obtained approximate prefix sums are consistent. For the exact details the reader is referred to [22],[23].

Certainly, each approximate prefix sum is a $(1+\epsilon)^{\log n}$-approximation of the exact prefix sum. For small enough $\epsilon$ we have $(1+\epsilon)^{\log n} < 1 + 2\epsilon \log n$ as required. As we can take $\epsilon = (\log n)^{-a}$, for any fixed $a > 0$, the fact that the accuracy parameter is multiplied by $2 \log n$ is of no concern. This establishes the following result:

**Theorem 3.2** *Let $A$ be a Boolean array of size $n$ each of whose elements is an $O(\log n)$-bit integer. For any fixed $a > 0$, consistent $(\log n)^{-a}$-approximate prefix sums of the sequence contained in $A$ can be computed in constant time using a polynomial number of processors in the* COMMON CRCW PRAM *model.*

## 4 Achieving optimal speedup

All the algorithms of the two previous sections run in constant time. They are wasteful however as they use a large (though polynomial) number of processors. In this section we show that all the above mentioned tasks can be performed extremely fast even if only a linear number of operations is allowed. The accuracy of the approximation is slightly effected. It is now only $(\log \log n)^{-a}$ for any fixed $a > 0$.

**Theorem 4.1** *Let $A$ be a Boolean array of size $n$ each of whose elements in an $O(\log n)$-bit integer. For any fixed $a > 0$, consistent $(\log \log n)^{-a}$-approximate prefix sums of the sequence contained in $A$ can be computed in $O(\log \log n)$ time using $O(n/ \log \log n)$ processors in the* COMMON CRCW PRAM *model.*

**Proof :** We give at first an $O(\log \log n)$ time algorithm that uses $O(n)$ processors. Suppose that consistent $(\log n)^{-a}$-approximate prefix sums of an array of size $n$ can be computed, by the algorithm of the previous section, in constant time using $O(n^b)$ processors, for some $b > 0$. Break the array $A$ into $n^{1/b}$ subarrays each of size $n^{1-1/b}$ and compute consistent approximate prefix sums of each one of them using a recursive call to the algorithm being described, or using the standard logarithmic time algorithm if the size of the subarray is at most, say, $O(\log n)$ (the prefix sums of such small arrays can be computed exactly in $O(\log \log n)$ time using a linear number of operations). Then, compute consistent approximate prefix sums of an array of size $n^{1/b}$ obtained by taking the last approximate prefix sum (i.e., the approximate sum) of each subarray. This can be done in constant time using $O(n)$ processors by applying the algorithm of the previous section. Consistent approximate prefix sums of the array $A$ can now be computed in constant additional time. It is easily seen that such an approach leads to a running time of $O(\log \log n)$.

To reduce the number of processors required to $O(n/ \log \log n)$, break the original array into $n/ \log \log n$ subarrays each of size $\log \log n$. Allocate a processor to each such subarray. This processor computes sequentially, in $O(\log \log n)$ time, the exact prefix sums of its subarray. A single problem of size $n/ \log \log n$ is then solved using the previously described algorithm. A consistent prefix sums sequence of the whole array can be obtained now in constant time.

Let us consider now the accuracy of the approximate prefix sums obtained by the above algorithm. Suppose approximate prefix sums of an array $A$ of size $m$ are computed by breaking the array into $m_1$ subarrays of size $m_2$, computing $\epsilon_1$-approximate (and consistent) prefix sums of each subarray, and then computing $\epsilon_2$-approximate (and consistent) prefix sums of the sums, and then obtaining approximations to the prefix sums of the whole array. Then $\epsilon$-approximate (and consistent), where $\epsilon = (1+\epsilon_1)(1+\epsilon_2) - 1$, prefix sums of the whole array are obtained. If $\epsilon_1$ and $\epsilon_2$ are small we may use the approximation $\epsilon \approx \epsilon_1 + \epsilon_2$.

Let $\epsilon(m)$ denote the relative error of the approximate prefix sums computed during the course of the algorithm for a subarray of size $m$. It is easy to see that

$$\epsilon(m) \approx \begin{cases} \epsilon(m^{1-1/b}) + (\log m^{1/b})^{-a} & \text{if } m > \log n, \\ 0 & \text{otherwise.} \end{cases}$$

The dominant part of the error comes, therefore, from the application of the constant time approximate prefix sums algorithm on subarrays whose size is just above $\log n$. The resulting relative error in these applications is about $(\log \log n)^{-a}$. This completes the proof of the Theorem. □

The accuracy of the above algorithm can be slightly improved by devoting more time to the approximation of prefix sums of small subarrays. With the help of Mike Paterson we have been able to improve the accuracy to $2^{-\Theta(\log \log n/ \log \log \log n)}$. The same result has been obtained by Hagerup after seeing a previous drarf of this work.

Using ideas similar to the ones used above, but by terminating the recursion after a constant number of steps, we obtain the following result:

**Theorem 4.2** *Let $A$ be a Boolean array of size $n$ each of whose elements is an $O(\log n)$-bit integer. For any fixed $a > 0$ and any fixed $\epsilon > 0$, consistent $(\log n)^{-a}$-approximate prefix sums of the sequence contained in $A$ can be computed in $O(1)$ time using $O(n^{1+\epsilon})$ processors in the* COMMON CRCW PRAM *model.*

In general, various trade-offs are possible between running time, number of processors, and the accuracy of the approximations obtained. Such trade-offs will appear in the final version of the paper.

## 5 Deterministic approximate compaction

The ability to compute consistent approximate prefix sums implies immediately the ability to solve the following problems:

**Approximate compaction** Given an array $A$ of size $n$ compact the indices of the places that contain 1's into an array of size $(1+\epsilon)\cdot|A|$ for some $\epsilon = o(1)$, where $|A|$ is the number of 1's in $A$.

**Interval allocation** Given an array $A$ of size $n$ containing $O(\log n)$-bit integers, allocate $n$ non-overlapping intervals of length $a_1, a_2, \ldots, a_n$ within an interval of length $(1+\epsilon)\sum_{i=1}^{n} a_i$, where $\epsilon = o(1)$ and $a_1, \ldots, a_n$ are the elements of $A$.

**Theorem 5.1** *The approximate compaction and the interval allocation problems of size $n$ can be solved in $O(\log\log n)$ time using $O(n/\log\log n)$ processors in the* COMMON *CRCW PRAM model.*

This theorem improves an $O((\log\log)^3)$ bound of Hagerup [27]. Besides being faster, our result is stronger than Hagerup's algorithm as it solves the *ordered* versions of the compaction and interval allocation problems.

Chaudhuri [11] showed that solving the approximate compaction problem using $O(n)$ processors requires $\Omega(\log\log n)$ time, even if $\epsilon$ is only required to be a constant and not $o(1)$. Our algorithm matches this lower bound even though it uses only $O(n/\log\log n)$ processors and has $\epsilon = o(1)$. Chaudhuri's $\Omega(\log\log n)$ bound for the approximate compaction problem implies the bound for the approximate prefix sums problem. Our approximate prefix sums algorithm, therefore, is also optimal.

## 6 Applications to selection and sorting

The $\Omega(\log n/\log\log n)$ lower bound on the time needed to compute the parity of $n$ bits using a polynomial number of processors implies that $\Omega(\log n/\log\log n)$ time is also required to select the *median* or to sort a set containing $n$ elements using a polynomial number of processors, even if the input elements are just bits. These lower bounds can be circumvented, however, if *approximate selection* and *padded sorting* are considered.

Chaudhuri, Hagerup and Raman [13] used Hagerup's deterministic compaction and approximate counting algorithms to obtain a deterministic algorithm for approximate selection that runs in $O((\log\log n)^4)$ time with an optimal number of processors. Using the fact that approximate compaction and approximate counting can be performed in $O(1)$ time using a polynomial number of processors, we can improve the running time of their algorithm to $O(\log\log n)$.

**Theorem 6.1** *Let $X$ be a set containing $n$ elements and let $1 \le r \le n$. For any fixed $\epsilon > 0$, an element $x \in X$ whose rank $r'$ in $X$ satisfies $r \le r' \le (1+\epsilon)\cdot r$ can be found in $O(\log\log n)$ time using $O(n/\log\log n)$ processors in the* COMMON *CRCW PRAM model.*

The above result is best possible as shown by Chaudhuri [12]. Note that the elements of $X$ are not assumed to be taken from a small domain. The only operations allowed on them however are comparisons. Again, various tradeoff are possible between time and accuracy.

The *padded sorting* problem was introduced by MacKenzie and Stout [37]. A padded sorting algorithm is required to place the $n$ input keys in sorted order in an array of size $(1+\epsilon)n$, for some fixed $\epsilon > 0$. Unused cells of the output array should contain a special *null* value.

Hagerup and Raman [28] obtained a *randomized* algorithm for padded sorting $n$ elements in $O(\log n/\log k)$ time using $kn$ processors. This matches a lower bound of $\Omega(\log n/\log k)$, obtained by Alon, Azar and Vishkin [5],[7] and simplied by Boppana [10], that holds in the stronger parallel comparison model. In [29], Hagerup and Raman obtain a *deterministic* version of their algorithm that runs in $(\log n/\log k)\cdot(\log\log k)^5\cdot 2^{O(\log^* n-\log^* k)}$ time. Using our improved results we can somewhat reduce this time bound.

**Theorem 6.2** *A set of $n$ keys can be padded-sorted using $kn$ processors in* $(\log n/\log k)\cdot(\log\log k)^3\cdot$

$2^{O(\log^* n - \log^* k)}$ *time in the* COMMON *CRCW PRAM model.*

## 7 Further applications

Our optimal prefix sums algorithm can also be used to obtain deterministic versions of *optimization schemes* that were obtained by Goodrich [21], Gil, Matias and Vishkin [38],[20] and by Hagerup [25]. Such optimization schemes take a *loosely specified* algorithm that ignores the issue of processor allocation and produce an algorithm that solves the processor allocation problems. Our algorithm may also be applied to certain computational geometry probelms mentioned in Goodrich [21] and Goodrich, Matias and Vishkin [22],[23].

## Acknowlegments

We would like to thank Noga Alon, Yossi Azar, Dorit Dor, Torben Hagerup, Yossi Matias, Moni Naor and Mike Paterson for many helpful discussions and comments.

## References

[1] M. Ajtai. $\Sigma_1^1$-formulae on finite structures. *Annals of Pure and Applied Logic*, 24:1–48, 1983.

[2] M. Ajtai. Approximate counting with uniform constant depth circuits. In J.-Y. Cai, editor, *Advances in Computational Complexity Theory*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pages 1–20. American Mathematical Society, 1993. A preliminary version had appeared as IBM Research Report, RJ 5896, 1987.

[3] M. Ajtai and M. Ben-Or. A theorem on probabilistic constant depth computations. In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing, Washington, D.C.*, pages 471–474, 1984.

[4] M. Ajtai, J. Komlós, and E. Szemerédi. Deterministic simulations in Logspace. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, New York City*, pages 132–140, 1987.

[5] N. Alon and Y. Azar. The average complexity of deterministic and randomized parallel comparison-sorting algorithm. *SIAM Journal on Computing*, 17:1178–1192, 1988.

[6] N. Alon and J.H. Spencer. *The probabilistic method.* Wiley, 1992.

[7] Y. Azar and U. Vishkin. Tight comparison bounds on the complexity of parallel sorting. *SIAM Journal on Computing*, 16:458–464, 1987.

[8] P. Beame and J. Håstad. Optimal bounds for decision problems on the CRCW PRAM. *Journal of the ACM*, 36:643–670, 1989.

[9] G.E. Blelloch. Prefix sums and their applications. In J.H. Reif, editor, *Synthesis of Parallel Algorithms*, chapter 1, pages 35–60. Morgan Kaufmann, 1993.

[10] R.B. Boppana. The average-case parallel complexity of sorting. *Information Processing Letters*, 33:145–146, 1989.

[11] S. Chaudhuri. A lower bound for linear approximate compaction. In *Proceedings of the 2nd Israel Symposium on Theory and Computing Systems, Natanya, Israel*, pages 25–32, 1993.

[12] S. Chaudhuri. Sensative functions and approximate problems. In *Proceedings of the 34rd Annual IEEE Symposium on Foundations of Computer Science, Palo Alto, California*, pages 186–193, 1993.

[13] S. Chaudhuri, T. Hagerup, and R. Raman. Approximate and exact deterministic parallel selection. In *Symposium on Mathematical Foundations of Computer Science*, pages 352–361, 1993.

[14] R. Cole and U. Vishkin. Faster optimal parallel prefix sums and list ranking. *Information and Computation*, 81:334–352, 1989.

[15] S. Cook, C. Dwork, and R. Reischuk. Upper and lower bounds for parallel random access machines without simultaneous writes. *SIAM Journal on Computing*, 15:87–97, 1986.

[16] L. Denenberg, Y. Gurevich, and S. Shelah. Definability by constant-depth polynomial-size circuits. *Information and Computation*, 70:216–240, 1986.

[17] R. Fagin, M.M. Klawe, N.J. Pippenger, and L. Stockmeyer. Bounded-depth, polynomial-size circuits for symmetric functions. *Theoretical Computer Science*, 36:239–250, 1985.

[18] M. Furst, J.B. Saxe, and M. Sipser. Parity, circuits and the polynomial time hierarchy. *Mathemaical Systems Theory*, 17:13–17, 1984.

[19] O. Gaber and Z. Galil. Explicit constructions of linear sized superconcentrators. *Journal of Computer and System Sciences*, 22:407–420, 1981.

[20] J. Gil, Y. Matias, and U. Vishkin. Towards a theory of nearly constant time parallel algorithms. In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science, San Juan, Puerto Rico*, pages 698–710, 1991.

[21] M.T. Goodrich. Using approximation algorithms to design parallel algorithms that may ignore processor allocation. In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science, San Juan, Puerto Rico*, pages 711–722, 1991.

[22] M.T. Goodrich, Y. Matias, and U. Vishkin. Approximate parallel prefix computation and its applications. In *Proceedings of the 7th International Parallel Processing Symposium*, pages 318–325, 1993.

[23] M.T. Goodrich, Y. Matias, and U. Vishkin. Optimal parallel approximation for prefix sums and integer sorting. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms, Arlington, Virginia*, pages 241–250, 1994.

[24] T. Hagerup. Fast parallel space allocation, estimation and integer sorting. Technical Report MPI-I-91-106, Max-Planck-Institut für Informatik, Saarbrücken, 1991.

[25] T. Hagerup. The log-star revolution. In *Proceedings of the 9th Annual Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science, Vol. 577, pages 259–278. Springer, 1992.

[26] T. Hagerup. On a compaction theorem of Ragde. *Information Processing Letters*, 43:335–340, 1992.

[27] T. Hagerup. Fast deterministic processor allocation. In *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms, Austin, Texas*, pages 1–10, 1993.

[28] T. Hagerup and R. Raman. Waste makes haste: Tight bounds for loose parallel sorting. In *Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science, Pittsburgh, Pennsylvania*, pages 628–637, 1992.

[29] T. Hagerup and R. Raman. Fast deterministic approximate and exact parallel sorting. In *Proceedings of the 5th Annual ACM Symposium on Parallel algorithms and architectures, Velen, Germany*, pages 346–355, 1993.

[30] J. Håstad. Almost optimal lower bounds for small depth circuits. In S. Micali, editor, *Advances in Computing Research*, volume 5, pages 143–170. JAI Press, 1989.

[31] J. Håstad, I. Wegener, N. Wurm, and S.-Z. Yi. Optimal depth, very small size circuits for symmetric functions in $AC^0$. *Information and Computation*, 108:200–211, 1994.

[32] N. Immerman. Languages that capture complexity classes. *SIAM Journal on Computing*, 16:760–778, 1987.

[33] J. JáJá. *An Introduction to Parallel Algorithms.* Addison Wesley, 1992.

[34] R.E. Ladner and M.J. Fischer. Parallel prefix computation. *Journal of the ACM*, 27:831–838, 1980.

[35] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8:261–277, 1988.

[36] P.D. MacKenzie. Load balancing requires $\Omega(\log^* n)$ time. In *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms, Orlando, Florida*, pages 94–99, 1992.

[37] P.D. MacKenzie and Q.F. Stout. Ultra-fast expected time parallel algorithms. In *Proceedings of the 2nd Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, California*, pages 414–423, 1991.

[38] Y. Matias and U. Vishkin. Converting high probability into nearly-constant time – with applications to parallel hashing. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, pages 307–316, 1991.

[39] I. Newman, P. Ragde, and A. Wigderson. Perfect hashing, graph entropy and circuit complexity. In *Proceedings of the 5th Annual Structure in Complexity Theory Conference*, pages 91–99, 1990.

[40] P. Ragde. The parallel simplicity of compaction and chaining. In *Proceedings of the 17th International Colloquium on Automata Languages and Programming, Warwick, England*, Lecture Notes in Computer Science, Vol. 443, pages 744–751. Springer, 1990.

[41] L. Stockmeyer and U. Vishkin. Simulation of parallel random access machines by circuits. *SIAM Journal on Computing*, 14:409–422, 1984.

[42] U. Vishkin. Structural parallel algorithms. In *Proceedings of the 18th International Colloquium on Automata Languages and Programming, Barcelona, Spain*, pages 363–380, 1991.

[43] A.C.C. Yao. Separating the polynomial time hierarchy by oracles. In *Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science, Portland, Oregon*, pages 1–10, 1985.