Information Processing Letters 33 (1989/90) 297–300
North-Holland

# SORTING ROUGHLY SORTED SEQUENCES IN PARALLEL

Tom ALTMAN

*Department of Computer Science, College of Arts and Sciences, University of Kentucky, 915 Patterson Office Tower, Lexington, KY 40506-0027, USA*

Bogdan S. CHLEBUS *

*Instytut Informatyki, Uniwersytet Warszawski, PKiN, p. 850, 00-901 Warsaw, Poland*

Communicated by W.M. Turski
Received 3 May 1989
Revised 18 September 1989

*Keywords*: Sorting, parallel algorithms, analysis of algorithms

## 1. Introduction

In this paper we consider sorting on a concurrent-read concurrent-write (CRCW) PRAM when the input is "roughly sorted", that is, the keys which are to be close to each other after sorting are already not too far from each other. The relevant measure of presortedness of the input is its radius defined in Section 2. We develop an algorithm using $n$ processors to sort $n$ keys in time $\mathcal{O}(\log k)$, where $k \leqslant n$ is the radius of the input sequence. The max-prefix problem of computing the prefix for the operation of taking the maximum is considered also, and it is shown that this problem can be solved in time $\mathcal{O}(\log k)$ for the sequences of radius $k$.

In the sequel of this section a short review is given of general complexity results concerning sorting and prefix computation. For a detailed description and properties of CRCW PRAMs the reader can refer to [7].

There are two known algorithms to sort $n$ items using $n$ processors in time $\mathcal{O}(\log n)$. The first one is implied by the AKS sorting network of Ajtai et al. [1]. The second one, especially tailored for PRAMs, is the mergesort of Cole [8]. These results are close to optimal because of almost matching lower bounds. Namely, a lower bound of $\Omega(\log n/\log\log n)$ for sorting $n$ items with a polynomial number of processors on a CRCW PRAM was proved by Borodin and Hopcroft [6]. The tight bound $\Omega(\log n/\log 2p/n)$ to sort $n$ items using $p > n$ processors was proved for the parallel comparison model (see [2] and [4]). Cole [8] gave a CRCW algorithm running within time $\mathcal{O}(\log n/\log\log 2p/n)$ for $2n \leqslant p \leqslant n^2$.

The parallel prefix-computation problem is the following. Let $X$ be a finite set with a binary associative operation $\circ$. For a given sequence $a_1, \ldots, a_n \in X$, stored in an array, compute the sequence

$$x_1 = a_1, \qquad x_2 = a_1 \circ a_2, \ldots,$$
$$x_i = a_1 \circ \cdots \circ a_i, \ldots, x_n = a_1 \circ \cdots \circ a_n.$$

There is a standard method to compute the prefix in time $\mathcal{O}(\log n)$ using $n/\log n$ processors (cf. [10]). Ladner and Fischer [12] gave a logarithmic-depth and linear-size circuit computing the prefix sequence. The optimality of these constructions follows from the lower bounds on a more specific problem of computing the sum of $n$ integers

* Present address: Department of Mathematics and Computer Science, University of California, Riverside, CA 92521-0135, USA.

Clearly this is a subproblem of the prefix computation where the binary operation is the integer addition. The relevant $\Omega(\log n)$ lower bounds were proven by Meyer auf der Heide and Reischuk [15], Parberry [16] and Beame [5].

The general lower bounds do not exclude a possibility of sublogarithmic times to sort or compute the prefix for certain special cases. In this paper we develop a sorting algorithm which has an $\mathcal{O}(\log n)$ time worst-case performance, but which has a sublogarithmic behavior on roughly sorted sequences. A similar result is shown for the max-prefix problem.

## 2. Roughly sortedness

Various approaches to sorting almost sorted sequences can be found in the literature. Among them there are those of general presortedness studied by Estvil-Castro and Wood [9], Mannila [13] and Mehlhorn [14], and more specific roughly sortedness studied by Altman and Igarashi [3] and Igarashi and Wood [11]. For this paper the notion of a roughly sorted sequence will be relevant.

Let $\alpha = (a_1, \ldots, a_n)$ be a sequence of $n$ items from a set with a linear ordering $\leqslant$. The sequence $\alpha$ is said to be *k-sorted*, if for all $1 \leqslant i, j \leqslant n$ such that $j - i > k$ the relation $a_i \leqslant a_j$ holds, where $k \leqslant n$. Define the *radius* of $\alpha$ to be the smallest $k$ such that $\alpha$ is $k$-sorted. We refer to sequences with a radius $k = o(n)$ as to *roughly sorted* ones. In the sequel we develop a CRCW PRAM algorithm using $n$ processors to find the radius, say $k$, of a sequence $\alpha$ and then sort $\alpha$ in time $\mathcal{O}(\log k)$. This algorithm is of complexity $\mathcal{O}(\log n)$ in the general worst case, but is faster for roughly sorted sequences.

### 2.1. Some special prefix computation

Let $\alpha = (a_1, \ldots, a_n)$ be a sequence of items. Consider the max-prefix problem, that is computing the prefix for $\alpha$ and the binary operation $\max(x, y)$ of taking the greater element among $x$ and $y$.

**Theorem 1.** *If a sequence $\alpha$ has radius $k$, then the max-prefix for $\alpha$ can be solved in time $\mathcal{O}(\log k)$ on a CRCW PRAM using $n$ processors.*

**Proof.** Let $\alpha$ be stored in the array $A[1..n]$. Consider the following algorithm:

```
begin
(1) for each processor i in parallel do
(2)   for r := 0 to log n do
(3)     if A[1..n] is sorted in a nondecreasing order
(4)       then exit
(5)       else if i − 2^r ⩾ 1
                then A[i] := max(A[i], A[i − 2^r])
end
```

The instruction **exit** in the above algorithm can be performed in constant time by a CRCW with $n$ processors. Therefore the overall time is proportional to the number of iterations of loop (2) until $A[1..n]$ is sorted.

For each $1 \leqslant i \leqslant n$, let $m_i \leqslant i$ be such that $A[m_i]$ is maximal in the initial array $A[1..i]$. We prove by induction on $i - m_i$ that if $i > m_i$, then after at most $\lfloor \log(i - m_i) \rfloor + 1$ steps $A[i]$ stores the initial value $A[m_i]$. If $i - m_i = 1$, then this is clear. Suppose that $i > m_i + 1$ and let $r > 0$ be such that $i - 2^r \geqslant m_i$ but $i - 2^{r+1} < m_i$. Then $(i - 2^r) - m_i < i - m_i$ and, by the inductive assumption, $A[i - 2^r]$ stores the original value $A[m_i]$ after $\lfloor \log(i - 2^r - m_i) \rfloor + 1 \leqslant \lfloor \log(i - m_i) \rfloor + 1$ steps. From this it follows that if $k$ is the radius of $\alpha$, then after $\lfloor \log k \rfloor + 1$ iterations loop (2) will terminate and $A[1..n]$ will store the prefixes. □

### 2.2. Finding the radius

Let $\alpha = (a_1, \ldots, a_n)$ have radius $k$. Define the sequences $\beta = (b_1, \ldots, b_n)$ and $\gamma = (c_1, \ldots, c_n)$ as follows: $b_i = \max\{a_1, \ldots, a_i\}$, $c_i = \min\{a_i, \ldots, a_n\}$. Observe that $\beta$ and $\gamma$ are nondecreasing and $c_i \leqslant b_i$. Finally consider $\delta = (d_1, \ldots, d_n)$, where $d_i = \max(\{i - j: c_i < b_j\} \cup \{0\})$. It was proved by Altman and Igarashi [3] that the radius of $\alpha$ is the same as the largest element among $d_1, \ldots, d_n$. Observe that computing $\beta$ is the max-prefix problem and that $\gamma$ can be found similarly in time $\mathcal{O}(\log k)$. Then the sequence $\delta$ can be found by the doubling-like method in time $\mathcal{O}(\log k)$. What remains is finding the maximum.

**Lemma 2.** *Suppose that the array $C[1..n]$ stores $n$ integers from the range $[1..n]$. Then the maximum value, say $m$, stored in $C$ can be found in time $\mathcal{O}(\log m)$ by a CRCW using $n$ processors.*

**Proof.** Consider the following procedure:

**begin**
(1) **for** each processor $i$ **in parallel do**
(2) $r := 0$;
(3) **if** $C[i] < 2^{r+1}$ for each $1 \leqslant i \leqslant n$
(4) **then exit**
(5) **else** $r := r + r$
**end**

After its completion, the number $r$ satisfies the following inequality: $2^r \leqslant m \leqslant 2^{r+1}$. Now the exact value of $m$ can be found by a similar binary search in the interval $[2^r..2^{r+1}]$. □

This shows that the radius, say $k$, of a sequence $\alpha = (a_1, \ldots, a_n)$ can be found in time $\mathcal{O}(\log k)$ by a CRCW with $n$ processors.

### 2.3. Sorting

In this subsection we prove the following main theorem.

**Theorem 3.** *There is an algorithm on a CRCW PRAM using a linear number of processors which sorts each sequence $\alpha$ in time $\mathcal{O}(\log k)$, with $k$ the radius of $\alpha$.*

To prove it we consider algorithm roughsort defined below. It uses the algorithm mergesort of Cole. Mergesort, if applied globally, has time complexity $\mathcal{O}(\log n)$. We use it locally on separate portions of the array, which depend on the radius of the sorted sequence.

Suppose that $\alpha = (a_1, \ldots, a_n)$ has radius $k$. We can divide $\alpha$ into blocks as follows:

$$A_1 = (a_1, \ldots, a_k),$$
$$B_1(a_{k+1}, \ldots, a_{2k}), \ldots$$
$$\ldots$$
$$A_i = (a_{2(i-1)k+1}, \ldots, a_{2(i-1)k+k}),$$
$$B_i = (a_{2(i-1)k+k+1}, \ldots, a_{2(i)k}), \ldots .$$

The last block may have fewer than $k$ elements. Denote by $A_iB_i$ or $B_iA_{i+1}$ the concatenation of the respective sequences.

**Roughsort**
1. Compute the radius $k$ of $\alpha$ and partition $\alpha$ into blocks $A_i$ and $B_i$;
2. Sort each $A_iB_i$ in parallel using mergesort;
3. Sort each $B_iA_{i+1}$ in parallel using mergesort.

It is clear that the algorithm operates within time $\mathcal{O}(\log k)$. It remains to show its correctness. First let us observe the following useful fact.

**Lemma 4.** *If $\alpha = (a_1, \ldots, a_n)$ is $k$-sorted and some $a_i$ and $a_j$ are interchanged, for $i < j$ and $a_i > a_j$, then $\alpha$ remains $k$-sorted.*

Since each sequence can be sorted by consecutive transpositions, it follows from the above lemma, that after step 2 of the algorithm, if $a_k > a_m$ for $k < m$, then there is $1 \leqslant i \leqslant n$ such that $a_k \in B_i$ and $a_m \in A_{i+1}$. Hence after step 3 of the algorithm the input is sorted. This completes the proof of correctness of roughsort, and hence the proof of Theorem 3.

## References


[1] M. Ajtai, J. Komlos and E. Szemeredi, An $\mathcal{O}(n \log n)$ sorting network, *Combinatorica* 3 (1983) 1–19.

[2] N. Alon, Y. Azar and U. Vishkin, Tight complexity bounds for parallel comparison sorting, In: *Proc. 27th Ann. IEEE Symp. on Foundations of Computer Science* (1986) 502–510.

[3] T. Altman and Y. Igarashi, Roughly sorting: sequential and parallel approach, *J. Inform. Process.* 12 (1989) 154–158.

[4] Y. Azar and U. Vishkin, Tight comparison bounds on the complexity of parallel sorting, *SIAM J. Comput.* 16 (1987) 458–464.

[5] P. Beame, Limits on the power of concurrent-write parallel machines, In: *Proc. 18th Ann. ACM Symp. on Theory of Computing*, Berkeley, CA (1986) 169–176.

[6] A. Borodin and J. Hopcroft, Routing, merging and sorting on parallel models of computation, *J. Comput. System Sci.* 30 (1985) 130–145.

[7] B.S. Chlebus, K. Diks, T. Hagerup and T. Radzik, Efficient simulations between concurrent-read concurrent-write PRAM models, In: *Proc. 13th Symp. on Mathematical Foundations of Computer Science*, Karlove Vary, Lec-

ture Notes in Comput. Sci. **324** (Springer, New York, 1988) 231–239.

[8] R.Cole, Parallel merge sort, *SIAM J. Comput.* **17** (1988) 770–785.

[9] V. Estvil-Castro and D. Wood, A new measure of presortedness, Tech. Report CS-87-58, Dept. Comput. Sci., Univ. Waterloo, Ont., 1987.

[10] A. Gibbons and W. Rytter, *Efficient Parallel Algorithms* (Cambridge Univ. Press, New York, 1988).

[11] Y. Igarashi and D. Wood, *Roughly Sorting: A Generalization of Sorting*, Tech. Report CS-87-55, Dept. Comput. Sci., Univ. Waterloo, Ont., 1987.

[12] R.E. Ladner and M.J. Fischer, Parallel prefix computation, *J. Assoc. Comput. Mach.* **27** (1980) 831–838.

[13] H. Mannila, Measures of presortedness and optimal sorting algorithms, *IEEE Trans. Comput.* **34** (1985) 318–325.

[14] K. Mehlhorn, Sorting presorted files, In: *Proc. 4th GI Conf. on Theory of Computer Science* (1979) 199–212.

[15] F. Meyer auf der Heide and R. Reischuk, On the limits to speed up parallel machines by large hardware and unbounded communication, In: *Proc. 25th Ann. IEEE Symp. on Foundations of Computer Science*, Singer Island, FL (1984) 56–64.

[16] I. Parberry, On the time required to sum $n$ semigroup elements on a parallel machine with simultaneous writes, In: *Proc. 2nd Internat. Workshop on Parallel Computing and VLSI*, Loutraki, Greece, Lecture Notes on Comput. Sci. **227** (Springer, New York, 1986) 296–304.