Extend Input Domain for Integer Sorting on Sum CRCW

Hazem M. Bahig
Computer Science Division, Department of Mathematics,
Faculty of Science, Ain Shams University,
Cairo, Egypt.
hbahig@asunet.shams.edu.eg

Abstract

Given a sorted array \( A = (a_1, a_2, \ldots, a_n) \) such that the \( n \) elements are taken from an integer range \([1, n]\). We extend the domain of input data of Akl and Chen algorithm from a restricted distribution to any distribution. The modified algorithm is deterministic with optimal cost, runs in \( O\left(\frac{\log n}{\log \log n}\right) \) time using \( \frac{n \log \log n}{\log n} \) Sum CRCW processors and uses linear space.

Keywords: parallel algorithms, deterministic algorithms, optimal algorithms, integer sorting, Sum CRCW PRAM.

1. Introduction

Given an array \( A \) of \( n \) elements. The sorting problem is the rearrangement of \( n \) elements in ascending order. The sorting problem is a fundamental problem in computer science. The problem is important for the following reasons: (1) it solved by various methods, (2) the optimal algorithm for sorting exists, and (3) it has many applications in computer science such as database, computational geometry, combinatorial problems, and pattern recognition.

In some applications, such as combinatorial problems and simulation, the elements to be sorted are taken from a restricted integer range \([0, m-1]\), where \( m \) is a polynomial of \( n \). For elements with this special property, we call it integer sorting. In case of \( m = O(n) \), the Bucket sort algorithm sorts the \( n \) elements in \( O(n) \) sequential time. In case of \( m = O(n^k) \), for some constant \( k \), the Radix sort algorithm sorts the \( n \) elements in \( O(k \log n) \) sequential algorithm.

The integer sorting is studied by parallelism to speedup the time under shared and non-shared models. Many papers are concerned on PRAM, especially on CRCW PRAM. PRAM is a machine consisting of a finite number, \( p \), of processors (RAMs) operating synchronously on an infinite global memory consisting of cells numbered 0, 1, 2, ..., \( p \). In each step, each processor may carry out local computation, may access the global memory, or may write to one global memory cell. Various PRAM models have been introduced. They differ in the conventions regarding to concurrent reading and writing [1, 2]. One of them is the concurrent read concurrent write (CRCW) PRAM, where both simultaneously reading and writing of the same cell are allowed.

In case of concurrent writing, different assumption are made about which processor’s value is written into the memory location to resolve write conflicts. In Common CRCW, all processors must attempt to write the same value. In Arbitrary CRCW, one of the processors succeeds. In Priority CRCW, the smallest numbered among the processors succeeds. In Tolerant CRCW, the contents of the cell do not change. In Collision CRCW, a special collision symbol appears in the cell. In Min CRCW, the processor trying to write the minimum value succeeds. A CRCW PRAM is called a Sum CRCW PRAM if the sum of the values processors attempt to write is actually written in case of write conflict.

Hagerup [10] has given an algorithm that sorts \( n \) integers in the range \([1, n^c]\) in time \( O(\log n) \) and \( O(n^{1+\varepsilon}) \) space using \( \frac{n \log \log n}{\log n} \) Priority CRCW processors, for any fixed \( \varepsilon > 0 \) and \( c \) is a constant. Hagerup’s algorithm is modified by Bhatt [6] to run on an Arbitrary CRCW PRAM in time \( O\left(\frac{\log n}{\log \log n} + \log \log m\right) \) with a time--processor product \( O(n \log \log m) \). Rajasekaram and Reif [14] have given a randomized optimal algorithm for sorting \( n \) integers in the range \([1, n(\log n)^{O(1)}]\). Their algorithm runs in time \( O(\log n) \) using \( \frac{n}{\log n} \) CRCW processors. Chlebus [7, 8] has given an optimal randomized algorithm when the \( n \) integers are drawn...
randomly from a polynomial range. The algorithm runs in $O(\log n)$ time using $\frac{n}{\log n}$ CRCW processors. In 1990, two randomized algorithms were suggested. The first one was suggested by Raman [15]. It takes $O\left(\frac{\log n}{\log \log n} + \log m\right)$ time using $\min(n, n\log \log m \log \log n)$ Collision CRCW processors and $O(n)$ space, for any $m > n$. The other was suggested by Matias and Vishkin [13]. It runs in $O(\log n)$ time and $O(n)$ space using $\frac{n\log \log m}{\log n}$ Min CRCW processors.

Also, by using Tolerant CRCW PRAM, Saxena [16] gave an algorithm in $O\left(\frac{\log n}{\log \log n}\right)$ time with a processor-time product of $O(n(\log \log n)^2)$. The same running time was obtained by Akl and Chen [3] with linear work on Sum CRCW PRAM when the $n$ inputs are taken from the range $[1,n]$.

From the above overview on integer sorting on CRCW PRAM, the only optimal deterministic algorithm is the algorithm that introduced by Akl and Chen [3]. In [4], the author proved that Akl and Chen algorithm is not run in sublogarithmic time for every input data distribution. In this paper, we extend Akl and Chen algorithm so that it works on any input data distribution. Our modification runs in sublogarithmic time, $O\left(\frac{\log n}{\log \log n}\right)$, using $\frac{n\log \log n}{\log n}$ processors. The algorithm uses linear space, $O(n)$. Also, the algorithm is optimal and deterministic. Moreover, no previous optimal deterministic algorithm has sublogarithmic time.

Our work consists of an introduction and five sections. Akl and Chen algorithm is given in Section 2. In Section 3, we give our modified algorithm and its analysis for integer sorting on Sum CRCW PRAM. The comparison between optimal algorithms is given in Section 4. Finally, in Section 5 includes the conclusion of our work.

2 Akl and Chen Algorithm

Akl and Chen [3] proposed a simple deterministic algorithm for the integer sorting problem on Sum CRCW PRAM. The algorithm runs in sublogarithmic time, $O\left(\frac{\log n}{\log \log n}\right)$, with linear work, $O(n)$, using $n$ Sum CRCW processors. The algorithm is based on counting technique. The algorithm takes $n$ inputs integer $A = (a_1, a_2, ..., a_n)$ from the integer range $[1,n]$ and consists of the following steps.

Algorithm: Akl and Chen

**Input:** Given an array $A = (a_1, a_2, ..., a_n)$ of $n$ integers from the range $[1,n]$.

**Output:** Sorted array $A$.

**Begin**

1. Initialize the elements of an auxiliary array $B$ (of length $n+1$) with 0, i.e. $b_i = 0$, $\forall 0 \leq i \leq n$.
2. Processor $p_i$ write 1 into $b_n$, $\forall 0 < i \leq n$.
3. Apply prefix sum computation on the array $B$ and store the result in the array $C$.
4. For $0 < i \leq n$, do the following in parallel:
   
   if $c_{i-1} < c_i$, then write $i$ into $a_{c_i}, ..., a_{c_i-b_i+1}$.

**End.**

In [4], the author proved that Akl and Chen algorithm is not run in sublogarithmic time for every input data distribution. The proof is depending on the following observation. If the value of $b_i$ is greater than $\frac{\log n}{\log \log n}$, then the processor, $p_i$, takes $O(b_i)$ time in step 4. By setting $b_i = \frac{\log n}{\log \log n}$, say, the running time of the algorithm is $O(\log n)$ to write the integer $i$ into the locations from $c_i$ to $c_i - b_i + 1$. Therefore, the running time of the algorithm is not sublogarithmic for every input data distribution.

For example, suppose that $n = 16$ and the array $A$ consists of the following elements, where the repetition of the integer 7 is $\frac{n}{2}$.

$$A = (7,2,11,7,2,7,5,4,7,7,11,7,6,7,7,5)$$

The processor $p_7$ takes $O(n)$ time to write the integer 7 in $n/2$ locations, $a_7, a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13},$ and $a_{14}$, in step 4. Therefore, the algorithm does not run in sublogarithmic time, $O\left(\frac{\log n}{\log \log n}\right)$, in general case.

3 Modification

In this section, we propose an extension to Akl and Chen algorithm. The proposed algorithm uses
\[ p = \frac{n \log \log n}{\log n} \] processors on Sum CRCW PRAM. The \( n \) input numbers are taken from the integer domain \([1,n]\) and there is no restriction on the repetition of any element in the array. Our work consists of three parts. In the first part, we present the steps of our modified algorithm. Also, we give an example to illustrate how the steps of the algorithm are work. The input data of our example is an array \( A \) of \( n=16 \) elements.

\[ A = (7,2,11,7,2,7,5,4,7,7,14,7,6,7,7,16) \]

In the second part, we study the correctness of the modification algorithm. Finally, we study the analysis of our algorithm.

### 3.1 Steps of the algorithm

Suppose that \( m = \frac{\log n}{\log \log n} \). Our modification consists of the following steps:

**Step 1:** Initialize an auxiliary array \( R \) with value zero. The array \( R \) consists of \( n \) elements. The \( i \)th element represents the total number of repetitions for the integer \( i \), where \( i \in [1, n] \).

For our example, the array \( R \) is given as follows.

\[ R = (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) \]

**Step 2:** Compute the total number of repetitions for each element in the array \( A \).

To compute the total number of repetitions for each element we divide the array \( A \) into \( m \) subarrays, \( A_i \), of \( p \) length each. Then we compute the number of repetitions for each element \( a_j \) in the subarray \( A_i \) using the property of CW and save it in the auxiliary array \( R' \). The array \( R' \) consists of \( n \) elements. Finally, we update the number of repetition of the element \( a_j \) in \( R \) by the value computed in \( R' \).

Step 2 consists of \( m \) iterations, \( 1 \leq i \leq m \). In each iteration we do the following.

1. **Step 2.1** The processor \( p_j \) reads the element \( a_{(i-1)p+j} \), \( \forall 1 \leq j \leq p \).
2. **Step 2.2** The processor \( p_j \) reads the current value of the number of repetition of the integer \( a_{(i-1)p+j} \) from the array of repetition \( R \), i.e. \( r_{a_{(i-1)p+j}} \), and then store it into an auxiliary array \( X \) at position \( j \), i.e. \( x_j \). The initial value of \( X \) is zero.
3. **Step 2.3** Count the number of the integer \( a_{(i-1)p+j} \) in the subarray \( A_i \) by writing 1 at position \( a_{(i-1)p+j} \) in the auxiliary array \( R' \).

\( (2.4) \) Write \( 1 + \frac{x_j}{r_{a_{(i-1)p+j}}} \) at position \( a_{(i-1)p+j} \) in the array of repetitions \( R \).

By applying this step we obtain the following:

- For \( i = 1 \):
  \[ X = (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) \]
  \[ R' = (2,0,1,1,0,3,0,0,0,0,1,0,0,0,0,0) \]

- For \( i = 2 \):
  \[ X = (3,3,0,3,0,3,0) \]
  \[ R' = (0,2,0,1,1,8,0,0,0,1,0,0,1,0,1) \]

**Step 3:** Update the value of each element in the array \( R \), if the value of the element is not perfect integer, by taking the ceiling of \( r_i \), \( \forall 1 \leq i \leq n \).

\[ R = (0,2,0,1,1,8,0,0,0,1,0,0,1,0,1) \]

**Step 4:** Compute the prefix sum, \( S \), for the array of repetition that obtained from step 3.

The prefix sum problem is defined as: given an array \( S = (s_1, s_2, ..., s_n) \) and an associated binary operator \( + \). The prefix sum problem is to compute the \( n \) prefix sums \( s_i = x_1 + x_2 + ... + x_i \), \( \forall 1 \leq i \leq n \). The algorithm takes \( O\left( \frac{\log n}{\log \log n} \right) \) time using \( \frac{n \log \log n}{\log n} \) Common CRCW processors [9].

By using the following theorem we can simulate the prefix sum algorithm on Sum CRCW without increasing the running time.

**Theorem 1** [3] A *Sum CRCW PRAM* can simulate a Common CRCW PRAM within the same time and processor bounds, but not vice versa.

We assume that the array \( S \) consists of \( n+1 \) elements, \( S = (s_0, s_1, ..., s_n) \) where \( s_0 = 0 \).

By applying this step, we obtain

\[ S = (0,0,2,2,3,4,5,13,13,13,13,14,14,14,15,16) \]

**Step 5:** Initialize the elements of the input array with zero.

By applying this step, we obtain

\[ A = (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) \]

**Step 6:** Determine the first position of each distinct elements in the sorted array \( A \) by the following relation. The first position of the integer \( (i-1)p+j \) in the sorted array \( A \) is equal to the total number of integers less than the integer \( (i-1)p+j \) plus one.

By applying this step, we obtain

\[ A = (2,0,4,5,6,7,0,0,0,0,0,0,0,0,11,14,16) \]
Step 7: Distribute the repetition of each distinct element, if exist, into the sorted array $A$.
In this step, we use the prefix maxima algorithm to distribute the repetition of each distinct element into $A$. The prefix maxima problem is defined as: given an array $X=\langle x_1, x_2, \ldots, x_n \rangle$ and an associated binary operator $\geq$. The prefix maxima problem is to compute the $n$ prefix maxima $m_i= \max \{x_1, x_2, \ldots, x_i\}, \forall 1 \leq i \leq n$. The algorithm takes $O\left(\frac{\log n}{\log \log n}\right)$ time using $\frac{n \log \log n}{\log n}$ Common CRCW processors [9].
By applying this step we obtain
$$A = (2,2,4,5,6,7,7,7,7,7,7,7,7,11,14,16)$$
The pseudocode of the algorithm is as follows.

Algorithm: Modified Akl and Chen Algorithm  
Input: Given an array $A=\langle a_1, a_2, \ldots, a_n \rangle$ of $n$ integers from the integer range $[1,n]$.
Output: Sorted array $A$.

Begin
1. Initialize the elements of the array $R$ with zero values.
2. Repeat the following $m$ times, $1 \leq i \leq m$:
   For each processor $p_j$, $1 \leq j \leq p$, do in parallel:
   (2.1.) Assign processor $p_j$ to the element $a_{(i-1)p+j}$.
   (2.2.) Processor $p_j$ reads $r_{a_{(i-1)p+j}}$ and store it in $x_j$.
   (2.3.) Processor $p_j$ writes $1$ at $r_{a_{(i-1)p+j}}$
   (2.4.) Processor $p_j$ writes $1$ if $r_{a_{(i-1)p+j}}$.
3. Repeat the following $m$ times, $1 \leq i \leq m$:
   For each processor $p_j$, $1 \leq j \leq p$, do in parallel:
   processor $p_j$ update $r_{a_{(i-1)p+j}}$ by $\land \{r_{a_{(i-1)p+j}}\}$.
4. Compute the prefix sum, $S$, for the array $R$ using $p$ processors.
5. Initialize the elements of the array $A$ with zero values using $p$ processors.
6. Repeat the following $m$ times, $1 \leq i \leq m$:
   For each processor $p_j$, $1 \leq j \leq p$, do in parallel:
   if $s_{(i-1)p+j-1} < s_{(i-1)p+j}$ then write $(i-1)p+j$ into $a_{s_{(i-1)p+j}}$. 
7. Compute the prefix maxima for the array $A$ using $p$ processors.
End.

3.2 Correctness

Now, we prove that the algorithm is correct. In computing the repetition of each element using the property of Sum CW, we use the steps 2.2, 2.3, and 2.4 for the following reason. If 1 is written to position $r_{a_{(i-1)p+j}}$ during several different iterations, then there 1’s are not added together. Therefore, we can’t replace step 2 as follows:

Repeat the following $m$ times, $1 \leq i \leq m$:
Processor $p_j$ writes $1$ at $r_{a_{(i-1)p+j}}$, $\forall 1 \leq i \leq p$.

Also, to overcome the possibility of decreasing the value of $r_{a_{(i-1)p+j}}$ by small fraction, less than 1, due to several divisions, we take the ceiling of $r_{a_{(i-1)p+j}}$. So, after these steps the array $R$ represents the repetition of each element in the array $A$.

The different elements in the array $A$ can be determined by comparing the two successive elements of $S$. If $s_{(i-1)p+j-1} \leq s_{(i-1)p+j}$ then the integer $(i-1)p+j$ has nonzero repetition. But the integer $(i-1)p+j-1$ may have a repetition or not. Since $\eta_{(i-1)p+j}$ represents the repetition of the integer $(i-1)p+j$ and $s_{(i-1)p+j} = \eta_{(i-1)p+j} + 1$. Therefore the algorithm is optimal in sense of cost since the product of the running time and the number of processors is $O(n)$. Also, the space of the algorithm is linear.
Theorem 2 An Array $A$ of $n$ elements taken from the integer range $[1, n]$ can be sorted in $O\left(\frac{\log n}{\log \log n}\right)$ time and $O(n \log \log n)$ space on Sum CRCW PRAM.

4 Comparison

In this section we compare our result with previous known optimal algorithms on CRCW PRAM. The comparison between them according to kind of concurrent write, domain of input data, running time, and deterministic or not. Table 1, illustrate the comparison between five optimal algorithms.

From the table, we found that:

(1) The best algorithms according to the running time (main objective of parallel computation) are our algorithm and the algorithms in [3],[5]. But our algorithm is deterministic while the algorithm in [5] is randomized and the algorithm in [3] is correct with some restrictions.

(2) The best algorithm according to the large input domain is Chelbus’s algorithm.

(3) Our algorithm and Akl and Chen algorithm are deterministic while the others are randomized.

5 Conclusion and open problem

We have extended Akl and Chen algorithm for integer sorting on Sum CRCW PRAM to work on any input data distribution. Our modified algorithm is: (1) optimal in the sense of cost, (2) deterministic, (3) runs in sublogarithmic time, (4) uses linear space, and (5) works on any distribution of the input data.

There exist an open question related to this problem which is: Can we extend the domain of input data from integer range $[1, n]$ to $[1, n^k]$, for some constant $k$?

References


