IJSEAT International Journal of Science Engineering and Advance Technology

A Novel Approach for O (1) Parallel Sorting Algorithm

Lamesginew Andargie[1]
M.Sc. (Computer Science),
School of Com& Ele Engineering,
IOT, Bahir Dar University,
Ethiopia.

Gizachew Melkamu[2]
M.Sc. (Computer Science),
School of Com& Ele Engineering
IOT, Bahir Dar University
Ethiopia.

Dr. Vuda Sreenivasarao[3]
Professor
School of Com& Ele Engineering
IOT, Bahir Dar University
Ethiopia, INDIA

***Abstract*: Sorting is an algorithm of the most relevant operations performed on computers. In particular, it is a crucial tool when it comes to processing huge volumes of data into the memory. There are different types of sorting algorithms: simple sorting algorithms(such as insertion, selection and bubble) and parallel sorting(such as parallel merge sort, Odd-even sorting, Bitonic sort and O(1) parallel sorting) algorithm. Parallel sorting is the process of using multiple processing units to collectively sort an unordered sequence of data. In this paper is devoted to the discovery of new approach to O (1) parallel sorting algorithm, in which redundant data didn't taken into consideration yet.**

***Keywords:* Sorting, Parallel Algorithm, O(1) Sorting Algorithm.**

I. Introduction

In [1,3,6] sorting is a process of reordering a list of items in either increasing or decreasing order.It is a fundamental operation that is performed by most computers. It is a computational building block of basic importance and is one of the most widely studied algorithmic problems. Sorted data are easier to manipulate than randomly-ordered data, so many algorithms require sorted data. It is used frequently in a large variety of useful applications. All spreadsheet programs contain some kind of sorting code. Database applications used by insurance companies, banks, and other institutions all contain sorting code. Because of the importance of sorting in these applications, many sorting algorithms have been developed with varying complexity.

In order to speed up the performance of sorting operation, parallelism is applied to the execution of the sorting algorithms called parallel sorting algorithms. In designing parallel sorting algorithms, the fundamental issue is to collectively sort data owned by individual processors in such a way that it utilizes all processing units doing sorting work, while also minimizing the costs of redistribution of keys across processors. In [17], parallel algorithms can run on a multiprocessor computer that permits multiple instructions to execute concurrently. They perform more than one operation at a time.

A large number of parallel sort algorithms are available in literature. Of these parallel sorting algorithms, the main concern of this paper is O (1) parallel sorting algorithm. In this paper, we try to find problem of O (1) parallel sorting algorithm and propose new approach for it.

II. O (1) and Some Other Parallel Sorting Algorithms

Now let us have a look for basic idea of some parallel sorting algorithms. The *parallel merge sort* algorithm uses a divide and conquers strategy to sort its elements. The list is divided into 2 equally sized lists and the generated sub-lists are further divided until each number is obtained individually. The numbers are then merged together as pairs to form sorted lists of length 2. The lists are then merged subsequently until the whole list is constructed. This algorithm can parallelized by distributing n/p elements (where n is the list size and p is the number of processors) to each slave processor. The slave can sequentially sort the sub-list (e.g. using sequential merge sort) and then return the sorted sub-list to the master. Finally, the master is responsible of merging all the sorted sub-lists into one sorted list [2].

As in [2,3] stated that, the *odd-even transposition sort* algorithm starts by distributing n/p sub-lists (p is the number of processors) to all the processors. Each processor then sequentially sorts its sub-list locally. The algorithm then operates by alternating between an odd and an even phase, hence the name odd-even. In the even phase, even numbered processors (processor i) communicate with the next odd numbered processors (processor i+1). In this communication process, the two sub-lists for each 2 communicating processes are merged together. The upper half of the list is then kept in the higher number processor and the lower half is put in the lower number processor. Similarly, in the odd phase, odd number processors (processor i) communicate with the previous even number processors (i-1) in exactly the same fashion as in the even phase. It is clear that the whole list will be sorted in a maximum of p stages.

In [9], if a sequence increases or decreases from left to right, then it is a monotonic sequence. If $a_k < a_{k+1}$ for all $k < n$, then the sequence a_1, a_2, a_n is monotonic. A bitonic sequence is one that monotonically increases (decreases), reaches a single maximum (minimum), then monotonically decreases (increases). A *bitonic sequence* is obtained by concatenating two monotonic sequences, one ascending and the other descending. A cyclic shift of this concatenated sequence is also a bitonic sequence. The bitonic iterative rule is based on the observation that a bitonic sequence can be split into two bitonic sequences by performing a single step of comparison- exchanges [3]. Bitonic sorting takes advantage of this property of the bitonic split. By repeated applications of the bitonic split, a bitonic sequence can be converted to a monotonic sequence (i.e., sorted) [9].

[17] A parallel computer is a set of processors that are able to work cooperatively to solve a computational problem. This definition is broad enough to include parallel supercomputers that have hundreds or thousands of processors, networks of workstations, multiple-processor workstations, and embedded systems. The parallel computers can be represented with the help of various kinds of models such as random access machine (RAM), parallel random access machine (PRAM) etc.There are different models based on PRAM.In *Concurrent-Read Concurrent-Write (CRCW)* model, the processors access the memory location concurrently for reading as well as for writing operation. In the algorithm which uses CRCW model of computation, n^3 numbers of processors are employed. The complexity of CRCW based algorithm is O(1). In a situation when more than one processor tries to write on the same memory location, the value stored in the memory location is always the sum of the values computed by the various processors. In *Concurrent-Read Exclusive-Write (CREW)* model, the processors access the memory location concurrently for reading while exclusively for writing operation. Here, only one processor is allowed to write to one particular memory cell at any one step. In the algorithm which uses CREW model of computation, n^2 numbers of processors have been attached in the form of a two dimensional array of size n x n. The complexity of CREW based algorithm is O (n). But, one important thing here is that an algorithm that works correctly for CREW will also work correctly for CRCW but not vice versa.

For the current O (1) parallel sorting algorithm, a CRCW PRAM model where concurrent write is handled with addition is taken as assumption, and has the following pseudo code.

Algorithm:

```
For (int i=1; i<=n; i++)
{for (int j=1; j<=n; j++)
 {   if(X[i] > X[j])
       Processor Pij stores 1 in memory location mi
     else
       Processor Pij stores 0 in memory location mi
  }
 }
```

Let us see O (1) sorting algorithm using example.

Given unsorted list {**14, 12, 11, 13**}

P_{11} (14, 14)	P_{12} (14, 12)	P_{13} (14, 11)	P_{14} (14, 13)
		0+1+1+1 = 3	when sorted**14** is in position **3**
P_{21} (12, 14)	P_{22} (12, 12)	P_{23} (12, 11)	P_{24} (12, 13)
		0+0+1+0 = 1	when sorted**12** is in position **1**
P_{31} (11, 14)	P_{32} (11, 12)	P_{33} (11, 11)	P_{34} (11, 13)
		0+0+0+0 = 0	when sorted**11** is in position **0**
P_{41} (13, 14)	P_{42} (13, 12)	P_{43} (13, 11)	P_{44} (13, 13)
		0+1+1+0 = 2	when sorted**13** is in position **2**

Positions obtained from

Hence, the sorted list will be:-

0	**1**	**2**	**3**
11	**12**	**13**	**14**

III. PROPOSED ALGORITHM:

The headache for the above O (1) parallel sorting algorithm is that, it doesn't sort lists with having two or more redundant item values. It only does properly when the array list has distinct item values. Let us see this problem with example.

Given unsorted list {**14, 12, 11, 14, 13,11**}

P_{11} (14, 14) P_{12} (14, 12) P_{13} (14, 11) P_{14} (14, 14) P_{15} (14, 13) P_{16} (14, 11)

0+1+1+0+1+1 = 4**14** is in position **4** when sorted

P_{21} (12, 14) P_{22} (12, 12) P_{23} (12, 11) P_{24} (12, 14) P_{25}(12, 13) P_{26}(12, 11)

0+0+1+0+0+1 = 2**12** is in position **2**when sorted

P_{31} (11, 14) P_{32} (11, 12) P_{33} (11, 11) P_{34} (11, 14) P_{35} (11, 13) P_{36}(11, 11)

0+0+0+0+0+0 = 0 **11** is in position **0** when sorted

P_{41} (14, 14) P_{42} (14, 12) P_{43} (14, 11) P_{44} (14, 14) P_{45} (14, 13) P_{46} (14, 11)

0+1+1+0+1+1 = 4 **14** is in position **4** when sorted

P_{51} (13, 14) P_{52} (13, 12) P_{53} (13, 11) P_{54} (13, 14) P_{55} (13, 13) P_{56} (13, 11)

0+1+1+0+0+1 = 3 **13** is in position **3** when sorted

P_{61} (11, 14) P_{62} (11, 12) P_{63} (11, 11) P_{64} (11, 14) P_{65} (11, 13) P_{66} (11, 11)

0+0+0+0+0+0 = 0 **11** is in position **0** when sorted

The list looks like the following:

0	**1**	**2**	**3**	**4**	**5**
11		**12**	**13**	**14**	

From above scenario, in the given unsorted list, there are two 11 and two 14 redundant values. For both 11 values of the list, a position of 0 is assigned, and again for both 14 items of the list, a position with value of 4 is obtained. Hence, two positions (position 1 and position 5) are not assigned for items of the list. Therefore, one can observe that O (1) parallel sorting algorithm doesn't sort lists having 2 or more redundant values.

Our proposed algorithm addresses this problem of O (1) parallel sorting algorithm. The new proposed algorithm works, unlike as the current O (1) sorting algorithm, with the essence that CRCW and CREW PRAM model are taken in to consideration.The CRCW PRAM will be used when assigning a value of 0 or 1 for a given compared pair of arrays (for our case when comparing a[i] and a[j]). And, the CREW PRAM model will be used while a correct index for item of unsorted list is assigned for index array(for our case b[n] in the coming algorithm) to be used for sorting in the final ordered list. Furthermore, we consider CRCW and CREW because they are the most popular models of PRAM. CREW is popular because it maps to physical architecture well, and CRCW is used when the details of the current write must be specified.This algorithm does properly for lists of both distinct and redundant values. The pseudo code for our new proposed algorithm looks like the following.

Algorithm:

→ Given unsorted list with n size: **a[n]**

→ Have a list of n size for holding sorted items of the list: **x[n]**

→ Have an array with n size to hold index of sorted list for unsorted list: **b[n]**

→ Have a variable for arithmetic operation: **sum=0**;

```
for (int i=0; i < n ; i++)
  {for (int j = 0 ; i < n ; j++)
      {if ( a[i] > a [j])
          {
             Processor P (i+1)(j+1) stores 1 memory
             location mi+1
             sum ← sum+1
          }
        else
          Processor P (i+1)(j+1) stores 0 memory
          location mi+1
      }
    if ( i = =1)
      {b[i] ← sum
       x [ b[i] ] ← a [i]
      }
    else
      {for (k =0; k < i ; k ++ )
         {
            if (sum = = b [k])
            sum ← sum + 1
         }
       b[i] ← sum
       x [ b[i] ] ← a [i]
      }
    sum ← 0
  }
```

IV. **DISCUSSION AND COMPARISON OF THE PROPOSED ALGORITHM:** The above proposed algorithm works properly for both a list having distinct or redundant values. Let us see by taking the above scenario for O (1) parallel sorting algorithm and check for our proposed algorithm.

Given unsorted list {**14, 12, 11, 14, 13, 11**}

P_{11} (14, 14) P_{12} (14, 12) P_{13} (14, 11) P_{14} (14, 14) P_{15} (14, 13) P_{16} (14, 11)

0+1+1+0+1+1 = 4 **14** is in position **4** when sorted

P_{21} (12, 14) P_{22} (12, 12) P_{23} (12, 11) P_{24} (12, 14) P_{25} (12, 13) P_{26} (12, 11)

0+0+1+0+0+1 = 2 **12** is in position **2** when sorted

P_{31} (11, 14) P_{32} (11, 12) P_{33} (11, 11) P_{34} (11, 14) P_{35} (11, 13) P_{36} (11, 11)

0+0+0+0+0+0 = 0 **11** is in position **0** when sorted

P_{41} (14, 14) P_{42} (14, 12) P_{43} (14, 11) P_{44} (14, 14) P_{45} (14, 13) P_{46} (14, 11)

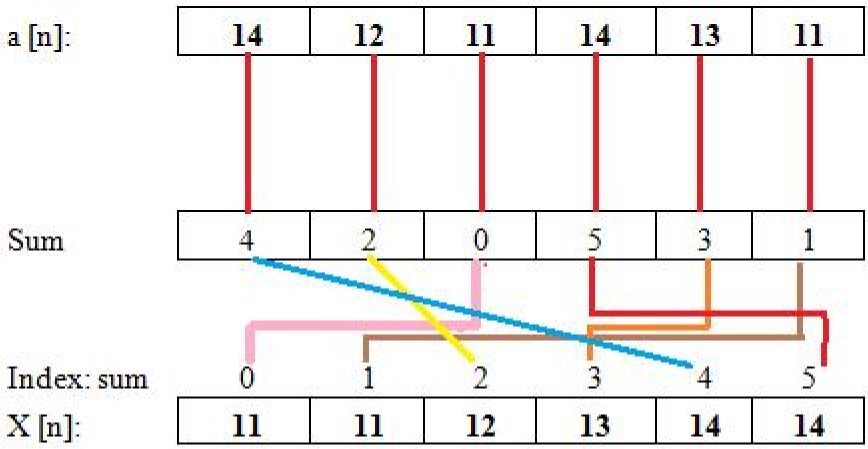
0+1+1+0+1+1 = 4 (+1)**14** is in position **5** when sorted

P_{51} (13, 14) P_{52} (13, 12) P_{53} (13, 11) P_{54} (13, 14) P_{55} (13, 13) P_{56} (13, 11)

0+1+1+0+0+1 = 3 **13** is in position **3** when sorted

P_{61} (11, 14) P_{62} (11, 12) P_{63} (11, 11) P_{64} (11, 14) P_{65} (11, 13) P_{66} (11, 11)

0+0+0+0+0+0 = 0 (+1)**11** is in position **1** when sorted



As stated in the algorithm, the algorithm checks arithmetic variable sum as the current position of the item before assigning it with the previous positions whether an equivalent/equal position exists in index list. If there exits equal value (s), them it adds 1 to the current value for each equal appearance of index, and continues like this until the end of the list. So that, by so doing like this, the above problem of O (1) parallel sorting algorithm is solved in our new proposed algorithm. We assumed that this comparison of the current position of item with the previous index values is done by the last processor for each row of P_{ij}.

V. Comparison Results:

Now compare our proposed algorithm with the existing O (1) parallel sorting algorithm.

Criteria	Current O (1) sorting algorithm	Proposed O (1) sorting algorithm
PRAM model used	CRCW	CRCW+CREW
Complexity	O (1)	O (n)
Sorts distinct items of list	Yes	Yes
Sorts lists with redundant items	No	Yes

VI. Conclusion

In this paper tries to show the problem of the current O (1) parallel sorting algorithm. It finds that O (1) sorting algorithm is inefficient for lists having two or more redundant values. The paper also proposes a new approach to handle and solve the current problem and shows the comparison between the proposed algorithm and the current algorithm of O (1) sorting algorithm.

VII. References

[1] Ishwari S. R., Bhawnesh K., Tinku S.; "*Performance Comparison of Sequential Quick Sort and Parallel Quick Sort Algorithms*", International Journal of Computer Applications (0975 – 8887); Volume 57– No.9, November 2012
[2] Kalim Qureshi, A *Practical Performance Comparison of Parallel Sorting Algorithms on Homogeneous Network of Workstations*, Kuwait University, Kuwait
[3] NA Bitton, Avid J. Dewitt, and Avid k. Hsiao AND Jaishankar Menon, Taxonomy of Parallel Sorting, puting Surveys, Vol. 16, No. 3, September 1984.
[4]Nancym. Amato, Ravishankariyer, Sharadsundaresan, Yanwu A Comparison of Parallel Sorting Algorithms on Different Architectures. Department of Computer Science, Texas A&M University, College Station, TX 77843-3112, January 1996.
[5]Friedhelm Meyer auf der Hede Avi Wigderson, The Complexity of Parallel Sorting,IBM Research Labratory, San Jose.
[6] David R. Helman, David A. Bader, Joseph J´ aJ´azy, A Randomized Parallel Sorting Algorithm /with an Experimental Study
[7] Davide Pasetto Albert Akhriev. A Comparative Study of Parallel Sort Algorithms IBM Dublin Research Lab, Mulhuddart, Dublin 15, Ireland
[8] B. Wilkinson & M. Allen, Parallel Programming Techniques & Applications Using Networked Workstations & Parallel Computers 2nd ed., 2004 Pearson Education Inc.
[9] Thomas Anastasio Bitonic Sorting December 4, 2003
[10] Laxmikant V. Kale, Sanjeev Krishnan, *A comparison based parallel sorting algorithm,* University of Illinois, Urbana
[11] Yingxu Wang, A *New Sort Algorithm: Self-Indexed Sort,Communications of ACM SIGPALN, Vol.31, No.3, March, 1996, ACM, pp.28-36*
[12]David R. Cheng, Viral B. Shah John R. Gilbert, Alan Edelman *A Novel Parallel Sorting Algorithm for Contemporary Architectures,* May 20, 2007
[13]G. E. Blelloch, C. E. Leiserson, B. M. Maggs, C. G. Plaxton, S. J. Smith, and M. ZaghaAn *Experimental Analysis of Parallel Sorting Algorithms*, Theory Computing Systems31, 135–167 (1998)
[14]Sanguthevar R., *On Parallel Integer Sorting,* University of Pennsylvania, Sandeep Sen
[15] Jariya Phongsai , *Research Paper on Sorting Algorithms*, October 26, 2009
[16] Pooja Adhikari, Review On Sorting Algorithms *A comparative study on two sorting algorithms*, Mississippi State, Mississippi, December 2007
[17] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivestand Clifford Stein. Introduction to Algorithms Third Edition.

AUTHORS BIBLIOGRAPHY:

Lamesginew Andargie Alamirew received his B.Sc. Degree in Computer Science & IT from Adama Science and Technology University in 2009.Currently perusing M.Sc. in Computer Science, School of Computing and Electrical Engineering, IOT, Bahir Dar University, Ethiopia. His main research interest is Algorithms, wireless computing and Data Mining. He is a member of professional society like SDIWC.

Gizachew Melkamu Mollareceived his B.Sc. Degree in Computer Science from University of Gondar in 2009.Currently perusing M.Sc. in Computer Science, School of Computing and Electrical Engineering, IOT, Bahir Dar University, Ethiopia. His main research interest is Algorithms, Cryptography and Data Mining. He is a member of professional society like SDIWC.

Dr. Vuda Sreenivasarao received his M.Tech degree in computer science and engineering from Sathyabama University from 2007.He received PhD degree in computer science and engineering from Singhania University, Rajasthan, India from 2010. Currently working as Professor in School of Computing and Electrical Engineering, IOT, Bahir Dar University, Ethiopia. His main research interests are Data mining, Fuzzy logic, Mobile communication and Network Security. He has got 13 years of teaching experience. He has published 32 research papers in various international journals and one Springer international conference paper. He has Editorial Board / Reviewers memberships in various international journals. He is a life member of various professional societies like IEEE, ACM, MAIRCC, MCSI, SMIACSIT, MIAENG, MCSTA, MAPSMS, MSDIWC, SMSCIEI, SNMUACEE and MISTE.