Scientific Research Publishing

Open Journal of Applied Sciences, 2019, 9, 403-408
http://www.scirp.org/journal/ojapps
ISSN Online: 2165-3925
ISSN Print: 2165-3917

An NC Algorithm for Sorting Real Numbers in $O\left(\frac{n\log n}{\sqrt{\log\log n}}\right)$ Operations

Yijie Han, Sneha Mishra, Md Usman Gani Syed

School of Computing and Engineering, University of Missouri at Kansas City, Kansas City, MO, USA
Email: hanyij@umkc.edu, smccr@mail.umkc.edu, gmsc8c@mail.umkc.edu

How to cite this paper: Han, Y.J., Mishra, S. and Syed, M.U.G. (2019) An NC Algorithm for Sorting Real Numbers in $O\left(\frac{n\log n}{\sqrt{\log\log n}}\right)$ Operations. *Open Journal of Applied Sciences*, **9**, 403-408.
https://doi.org/10.4236/ojapps.2019.95034

Received: April 8, 2019
Accepted: May 27, 2019
Published: May 30, 2019

Abstract

We apply the recent important result of serial sorting of n real numbers in $O\left(n\sqrt{\log n}\right)$ time to the design of a parallel algorithm for sorting real numbers in $O\left(\log^{1+\varepsilon} n\right)$ time and $O\left(\frac{n\log n}{\sqrt{\log\log n}}\right)$ operations. This is the first NC algorithm known to take $o\left(n\log n\right)$ operations for sorting real numbers.

Keywords

Parallel Algorithms, Sorting, Sort Real Numbers, Complexity

1. Introduction

It is known widely that serial comparison sorting takes $\theta\left(n\log n\right)$ time for sorting n numbers [1]. Although integer sorting can outperform the $\Omega\left(n\log n\right)$ lower bound for sorting n integers [2] [3], these algorithms generally do not apply to the problem of sorting real numbers. It has been known that n integers can be sorted in $O\left(n\log\log n\right)$ time and linear space [2] [3]. The $O\left(n\log n\right)$ time bound remains for sorting real numbers ever since. Only very recently Han showed that real numbers can be converted to integers for the sorting purpose in $O\left(n\sqrt{\log n}\right)$ time [4], thus enabling the serial sorting of real numbers in $O\left(n\sqrt{\log n}\right)$ time.

Parallel sorting algorithms for sorting real numbers run on the PRAM (Parallel Random Access Machine) model are known [5] [6]. The AKS sorting network [5] can be transformed into an EREW (Exclusive Read Exclusive Write) PRAM

algorithm with $O(\log n)$ time and $O(n \log n)$ operations. Cole's parallel merge sort [6] sorts n numbers in $O(\log n)$ time using n processors on the EREW PRAM. On the CRCW (Concurrent Read Concurrent Write) PRAM Cole showed [7] that his parallel merge sort can run in $O(\log n / \log\log(2p/n))$ time using p processors. Also see [7].

There are also parallel algorithms for integer sorting [8] [9] [10] [11]. In the case of integer sorting, the operation bound can be improved to below $O(n \log n)$. In particular, [10] presents a CRCW PRAM integer sorting algorithm with $O(\log n)$ time and $O(n \log\log n)$ operations and [11] presents an EREW PRAM integer sorting algorithm with $O(\log n)$ time and $O\left(n\sqrt{\log n}\right)$ operations.

For sorting real numbers, the previous best serial algorithm sorts in $O(n \log n)$ time. It was also known that for comparison sorting, $\Omega(n \log n)$ is the tight lower bound. Thus if we use comparison sorting to sort real numbers, then in serial algorithms, we cannot avoid the $\Omega(n \log n)$ time bound and in parallel algorithms, we cannot avoid the $\Omega(n \log n)$ operation bound. In the past, no other sorting methods are known to sort real number in less than $O(n \log n)$ time and comparison sorting remained the norm for sorting real numbers.

However, the situation is recently changed completely as Han found a way to convert real numbers to integers for sorting purpose and he showed that real numbers can be sorted in $O\left(n\sqrt{\log n}\right)$ time [4]. This result enables us to move further to improve the operation bound of parallel algorithms for sorting real numbers to below $O(n \log n)$, as in the past, all parallel algorithms for sorting real numbers have an operation bound at least $O(n \log n)$.

In this paper, we will apply the $O\left(n\sqrt{\log n}\right)$ time serial real number sorting algorithm [4] to the design of an NC algorithm with $O\left(\log^{1+\varepsilon} n\right)$ time and $O\left(\frac{n \log n}{\sqrt{\log\log n}}\right)$ operations on the CREW (Concurrent Read Exclusive Write) PRAM. NC stands for Nick's Class [12]. NC algorithms are parallel algorithms with polylog time and polynomial operations. Algorithm in [4] is an inherently serial algorithm without much parallelism within it. Here, we use it in the design of an NC algorithm with $O\left(\log^{1+\varepsilon} n\right)$ time and $O\left(\frac{n \log n}{\sqrt{\log\log n}}\right)$ operations. This is the first NC algorithm for sorting real numbers with $o(n \log n)$ operations.

The computation model used for designing our algorithm is the CREW PRAM. On this model, in one step, any processor can read/write any memory cell. Concurrent read of one memory cell by multiple processors in one step is allowed and concurrent write of one memory cell by multiple processors in one step is prohibited. Parallel algorithms can be measured with their time complexity and the number of processors used. They can also be measured with time complexity and operation complexity which is the time processor product. The operation complexity ($T_p p$, with T_p time using p processors) of a parallel algorithm is often compared with the time T_1 of the best serial algorithm. In general,

$T_p p \geq T_1$. When $T_p p = T_1$, the parallel algorithm is said to be an operation optimal algorithm.

The main contribution of this paper is the demonstration of the existence of an NC parallel algorithm with $o(n \log n)$ operations for sorting real numbers. All previous parallel algorithms for sorting real numbers have at least $O(n \log n)$ operations. The algorithm presented in this paper is derived from Han's $O\left(n\sqrt{\log n}\right)$ time serial algorithm [4] for sorting real numbers by applying parallel algorithm design techniques. These parallel algorithm design techniques are specially tuned for the derivation of our parallel algorithm.

The remaining part of this paper is organized as follows. In Section 2 we present our NC algorithm for sorting real numbers with $O\left(\log^{1+\varepsilon} n\right)$ time and $O\left(\frac{n \log n}{\sqrt{\log \log n}}\right)$ operations. In Section 3, we present a running example of our algorithm and conclude our paper with the Main Theorem.

2. The Algorithm

Consider an algorithm for sorting n real numbers. Suppose each of the n/m lists with m real numbers in each list has already been sorted, we are to merge these n/m lists into one sorted list. We will do k-way merge in each pass to merge every k-lists into 1 sorted list and there are $\log(n/m)/\log k$ passes to have all n/m lists merged into 1 sorted list.

For simplicity, let us break down n elements into lists with m elements in each list. We can do parallel sort on the individual list of m elements recursively. Now we pick every k-lists and have them merged together.

The k-way merging of sorted lists $L_0, L_1, \cdots, L_{k-1}$ is done as follows. For each sorted list of m real numbers we pick every k^2-th real number, *i.e.* we pick the 0th real number, the k^2-th real number, the $2k^2$-th real number, the $3k^2$-th real number, and so on. Thus from each list L_i we picked m/k^2 real numbers and these m/k^2 real numbers form a sorted list L_i'. and from these k lists we picked m/k real numbers they form sorted lists $L_0', L_1', \cdots, L_{k-1}'$. We merge $L_0', L_1', \cdots, L_{k-1}'$ into one sorted list L'using Valiant's merging algorithm [13] (its improved version is given by Kruskal in [14] with time complexity of $O(\log \log m)$ and linear operations for merging two sorted lists of m elements each) in log$k$ passes and $O(\log \log m)$ time and $O(m/k)$ operations in each pass. Thus the total time for merging $L_0', L_1', \cdots, L_{k-1}'$ is $O(\log k \log \log m)$ and the total operation is $O(m \log k/k)$.

Now for each real number r in L_i' and for any L_j', r knows the largest real number s in L_j' that is smaller than r and smallest real number l in L_j' that is larger than r. s and l are actually neighbors in L_j'. There are k^2 elements between s and l in L_j. r then uses binary search in $O(\log k)$ time to find the largest real number among these k^2 real numbers that is smaller than r and the smallest real number that is larger than r. That is, r finds the exact insertion point of r in L_j. Because there are k-lists and there are m/k real numbers in L'thus

the time for this binary search is $O(\log k)$ and the operation is $O(m \log k)$. The operation for all lists is $O(n \log k/k)$ because there are n/m lists and every k-lists are merged in the k-way merge, we picked n/k^2 real numbers and every one of them has to use k processors to check k-lists in the k-way merging. Because r is arbitrary picked and thus we know that every real number in L' knows its insertion point in every L_j. Let the real numbers in sorted order in L' be $r_0, r_1, \cdots, r_{m/k-1}$. To merge $L_0, L_1, \cdots, L_{k-1}$ we need now to merge or sort all real numbers between the insertion points of r_i and r_{i+1} in $L_0, L_1, \cdots, L_{k-1}$. There are no more than k^2 real numbers in L_j between the insertion points of r_i and r_{i+1} and therefore the total number $R(i, i+1)$ of real numbers (call them a block) in $L_0, L_1, \cdots, L_{k-1}$ between the insertion points of r_i and r_{i+1} is no more than k^3 (*i.e.* $R(i, i+1) \le k^3$). When $R(i, i+1) < k^3$ we will combine multiple blocks together to reach k^3 real numbers. We use the $O\left(n\sqrt{\log n}\right)$ serial sorting algorithm to sort them in $O\left(k^3\sqrt{\log k}\right)$ time. This represents $O\left(k^3\sqrt{\log k}\right)$ time and $O\left(n\sqrt{\log k}\right)$ operations in our parallel algorithm.

Thus the time for each stage is $O\left(k^3\sqrt{\log k}\right)$ and the operation for each stage is $O\left(n\sqrt{\log k}\right)$. When we start with m as a constant then there are $\log n/\log k$ stages and therefore the time of our algorithm is $O\left(\frac{k^3 \log n}{\sqrt{\log k}}\right)$ and the operation is $O\left(\frac{n \log n}{\sqrt{\log k}}\right)$.

Pick $k = \log^{\varepsilon} n$, we get $O\left(\log^{1+\varepsilon} n\right)$ time and $O\left(\frac{n \log n}{\sqrt{\log \log n}}\right)$ operations.

3. Procedure

Step 1: Let's say we have "m" sorted elements in each list, and we have a total of "n" elements to sort (Figure 1).

This implies that we have "n/m" lists to sort. To sort these blocks, we will apply k-way merging.

Step 2: Each stage of k-way merging is to merge every k sorted lists into 1 sorted list. This is repeatedly until all n/m lists are merged into one list (Figure 2).

Step 3: To merge k lists into 1 list, we need to pick the "0-th", "k^2-th", "$2k^2$-th", "$3k^2$-th", … real numbers in each list L_i to form a new list L_i' of m/k^2 elements. This is shown as Figure 3.

Step 4: We merge $L_0', L_1', \cdots, L_{k-1}'$ into one sorted list L'. The elements of this new formed list L' then use binary search to find their exact insertion point in

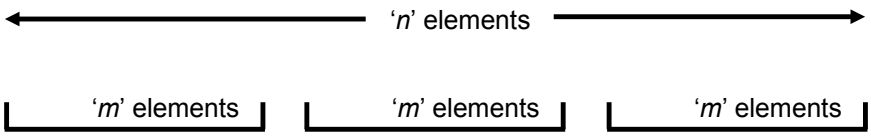

Figure 1. n/m sorted lists of m elements each.

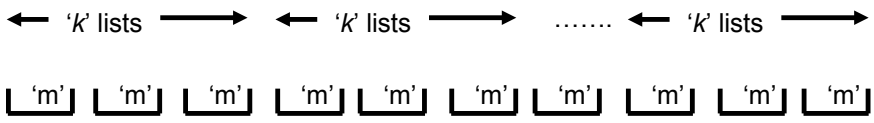

Figure 2. *k*-way merging of *k* lists.

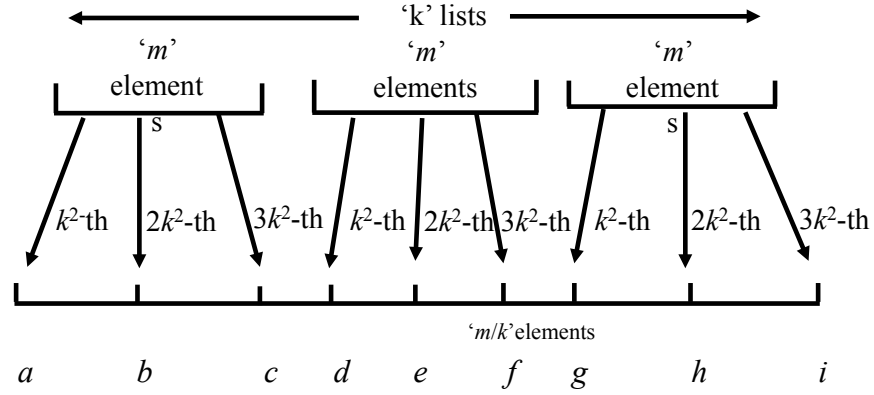

Figure 3. Pick every k^2-th element of each list.

$L_0, L_1, \cdots, L_{k-1}$. These insertion points then partition $L_0, L_1, \cdots, L_{k-1}$ into m/k blocks with each block containing no more than k^3 real numbers.

Step 5: When every one of these m/k blocks are sorted, we effectively merged $L_0, L_1, \cdots, L_{k-1}$ into one sorted list L.

Main Theorem: n real numbers can be sorted in $O\left(\log^{1+\varepsilon} n\right)$ time and $O\left(\frac{n \log n}{\sqrt{\log \log n}}\right)$ operations on the CREW PRAM.

Proof: The algorithm and its time complexity analysis are presented in Section 2. An example of the running of the algorithm is presented in Section 3. □

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Corman, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C. (2009) Introduction to Algorithms. Third Edition, The MIT Press.

[2] Han, Y. (2015) A Linear Time Algorithm for Ordered Partition. *International Workshop on Frontiers in Algorithmics* (*FAW*15), *LNCS*, **9130**, 89-103. https://doi.org/10.1007/978-3-319-19647-3_9

[3] Han, Y. (2004) Deterministic Sorting in $O(n\log\log n)$ Time and Linear Space. *Journal of Algorithms*, **50**, 96-105. https://doi.org/10.1016/j.jalgor.2003.09.001

[4] Han, Y. (2017) Sort Real Numbers in $O\left(n\sqrt{\log n}\right)$ Time and Linear Space. In arXiv.org with paper id 1801.00776.

[5] Ajtai, A., Komlós, J. and Szemerédi, E. (1983) An $O(n\log n)$ Sorting Network. *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, Bos-

ton, MA, 11-13 May 1983, 1-9. https://doi.org/10.1145/800061.808726

[6] Cole, R. (1988) Parallel Merge Sort. *SIAM Journal on Computing*, **17**, 770-785. https://doi.org/10.1137/0217049

[7] Saxena, S., Chandra, P., Bhatt, P. and Prasad, V.C. (1994) On Parallel Prefix Computation. *Parallel Processing Letters*, **4**, 429-436. https://doi.org/10.1142/S0129626494000399

[8] Bhatt, P.C.P., Diks, K., Hagerup, T., Prasad, V.C., Radzik, T. and Saxena, S. (1991) Improved Deterministic Parallel Integer Sorting. *Information and Computation*, **94**, 29-47. https://doi.org/10.1016/0890-5401(91)90031-V

[9] Hagerup, T. (1987) Towards Optimal Parallel Bucket Sorting. *Information and Computation*, **73**, 39-51. https://doi.org/10.1016/0890-5401(87)90062-9

[10] Han, Y. and Shen, X. (1995) Conservative Algorithms for Parallel and Sequential Integer Sorting. *International Computing and Combinatorics Conference*, *Lecture Notes in Computer Science*, **959**, 324-333. https://doi.org/10.1007/BFb0030847

[11] Han, Y. and Shen, X. (2002) Parallel Integer Sorting Is More Efficient than Parallel Comparison Sorting on Exclusive Write PRAMs. *Tenth Annual ACM-SIAM Symposium on Discrete Algorithms* (*SODA*'99), Baltimore, Maryland, January 1999, 419-428. https://doi.org/10.1137/S0097539799352449

[12] Cook, S.A. (1981) Towards a Complexity Theory of Synchronous Parallel Computation. *L'Enseignement Mathématique*, **27**, 99-124.

[13] Valiant, L.G. (1975) Parallelism in Comparison Problems. *SIAM Journal on Computing*, **4**, 348-355. https://doi.org/10.1137/0204030

[14] Kruskal, C.P. (1983) Searching, Merging, and Sorting in Parallel Computation. *IEEE Transactions on Computers*, **C-32**, 942-946. https://doi.org/10.1109/TC.1983.1676138