

Sorting Real Numbers in Constant Time using $n^2/\log^c n$ Processors

Yijie Han

School of Computing and Engineering
University of Missouri-Kansas City
Kansas City, MO 64110
Email: hanyij [AT] umkc.edu

Pruthvi Kasani

School of Computing and Engineering
University of Missouri-Kansas City
Kansas City, MO 64110
Email: pruthvikasani [AT] mail.umkc.edu

Sai Swathi Kunapuli

School of Computing and Engineering
University of Missouri-Kansas City
Kansas City, MO 64110
Email: skmqw [AT] umkc.edu

Abstract ---- We study the sorting of real numbers into a linked list on the PRAM (Parallel Random-Access Machine) model. We show that n real numbers can be sorted into a linked list in constant time using n^2 processors. Previously n numbers can be sorted into a linked list using n^2 processors in $O(\log \log n)$ time. We also study the time processor trade-off for sorting real numbers into a linked list on the PRAM (Parallel Random Access Machine) model. We show that n real numbers can be sorted into a linked list with n^2/t processors in $O(\log t)$ time. Previously n real numbers can be sorted into a linked list using n^3 processors in constant time and n^2 processors in $O(\log \log n)$. And then we show that input array of n real numbers can be sorted into linked list in constant time using $n^2/\log^c n$ processors for any positive constant c . We believe that further reduction on the number of processors for sorting real numbers in constant time will be very difficult if not impossible.

Keywords – Constant time sorting, sorting real numbers into a linked list, lower bounds for sorting, PRAM (Parallel Random Access Machine), EREW, CREW, CRCW.

1. INTRODUCTION

The requirement for parallel algorithms has become critical in these days and age. Our algorithm has a series of stages that takes many inputs from the input and can execute various instructions at the same time, combining all of the separate outputs to produce the final result. In this study, we use the PRAM (Parallel Random Access Machine) model [14,15] to sort n real numbers into a linked list. A Parallel Random Access Machine is a model that is used for the design of many parallel algorithms. In this model, n processors can conduct independent operations on n data sets in a unit of time. This may result in many CPU's accessing the same memory cells at the same time. This issue is resolved in various ways on the PRAM model: On the EREW (Exclusive Read Exclusive Write) PRAM [14,15], no two processors are allowed to read from or write to the same memory location at the same time, on the CREW (Concurrent Read Exclusive Write) PRAM [14,15], multiple processors are permitted to read from the same memory location at the same time but are not permitted to write to the same memory location at the same time, and in the CRCW (Concurrent Read Concurrent Write) PRAM [14,15], in which multiple processors are permitted to read from or write to the same memory

location at the same time. Since, the CRCW PRAM allows multiple processors to read and write simultaneously into a memory cell arbitration schemes are used to resolve concurrent write conflict. On the Priority CRCW PRAM, the processor with the highest priority wins the write on the memory cell among the processors writing to the same memory cell. The processor's index can be used as the priority. On the Arbitrary CRCW PRAM, an arbitrary processor is chosen to win the write from among the processors to write on the same memory cell. On the Common CRCW PRAM, processors write to the same memory cell in a step must write the same value, which is then written into the memory cell. Priority CRCW PRAM is the strongest of the three CRCW PRAM models; Arbitrary CRCW PRAM is weaker than Priority CRCW PRAM; and Common CRCW PRAM is the weakest of the three. In this study, we shall design algorithm on the Common CRCW PRAM. Because Common CRCW PRAM is weaker than Arbitrary and Priority CRCW PRAM and therefore our algorithm also runs on the Arbitrary and Priority CRCW PRAM.

Let T_p denote the complexity of a parallel algorithm with p processors. Let T_1 be the time complexity of the best serial algorithm for the same problem. Then $pT_p \geq T_1$. When $pT_p = T_1$, this parallel algorithm is an optimal parallel algorithm.

When we have a T_p time algorithm that uses P processors, we can represent or translate the time as $T_p P/p + T_p$ when we employ p processors.

A parallel algorithm for a problem of size n that uses polynomial number processors (i.e., n^c processors for a constant c) and runs in polylog time (i.e., $O(\log^c n)$ time for a constant c) is considered to belong to the NC class [4], where NC is Nick's class.

NC algorithms, as well as fast and efficient parallel algorithms, are being developed by researchers in the field of parallel algorithms.

In this paper, we will study sorting real numbers into a linked list in constant time using $n^2/\log^c n$ processors for any large

constant c . Further reducing the number of processors seems to be very difficult if not impossible.

On the CRCW PRAM with polynomial number of processors, it is known that sorting n real values into an array takes at least $(\log n / \log \log n)$ time [2]. If we want to sort them into a padded array, we need at least $(\log \log n)$ time [5]. There are fast merging and sorting algorithms [15] but they do not achieve constant time. However, if we arrange them into a linked list, we can demonstrate that it is possible to do so in constant time. Thus, the lower bounds of $(\log n / \log \log n)$ [2] and $(\log \log n)$ [5] are the bottom bounds for arranging integers in an array rather than "sorting" them.

There have been previous results for sorting integers into a linked list [3, 6]. It is known there that n numbers in the range of $\{0, 1, \dots, m-1\}$ may be sorted into a linked list in constant time using $n \log m$ processors. Parameter n here is not related m . Except for our prior results for sorting real numbers into a linked list [7, 8, 9, 10], we do not know any other results for parallel sorting real numbers into a linked list, nor do we know any previous results for sorting real numbers in constant time.

In [7, 8, 9] sorting integers and real numbers into a linked list is considered. The best result to sort real numbers into a linked list in constant time used n^2 processors [8]. Although in [9] the number of processors is reduced to less than n^2 by using linked list contraction [1,12,13] but the time is not constant.

II. SORTING REAL NUMBERS INTO A LINKED LIST USING n^2 PROCESSORS IN CONSTANT TIME.

We assume that the n input real numbers are distinct. This can be achieved by replacing every real number a by a pair (a, i) where i is the index of the number a in the input array.

Firstly, let us discuss about the algorithm on how to sort the real numbers into linked list using constant time using n^3 processors. Let us say, $A[0 \dots n-1]$ be the input array of n real numbers and we have n^3 processors to achieve constant time.

Assign n processors to each element of the array to compare it with the other elements in the array. It will write as 1 for the elements that it greater than the given element and 0 for the elements if it is less than it. For example, we have the given input array elements as 4,2,5,1,6,3,9. Let us pick an element 5 from the array. As said above, it marks 1 to the elements greater than 5 and 0 for the ones lesser than 5. So, the output is 0,0,0,0,1,0,1. We use the n^2 processors to the elements marked as 1 and find the smallest number among them (i.e., 6) in constant time [16,17] and link it to the element 5. So, here we have 6 and 9 out of which 6 is the minimum. So, 6 is linked to 5. This process is executed in parallel to all the elements in the array, and we get the final sorted linked list of elements. This algorithm can be done in constant time using n^3 processors.

Now, we let us show the algorithm on sorting the real numbers into a linked list using n^2 processors in $O(\log \log n)$ time on the Common CRCW PRAM. This algorithm is like the above

algorithm where we assign n processors to compare a number to the rest of the elements in the array. Now, we need to compute the minimum of n numbers using n processors. This can be done in $O(\log \log n)$ time [16,17]. Let us say $A[0 \dots n-1]$ be the input array of n real numbers. As above, the comparison task of comparing one element $A[i]$ to other elements takes constant time. Now, we need to find the minimum of elements in A that are larger than $A[i]$. Let us say m is the smallest element. Now, for each element in $A[i]$ we will copy it into a new array A_i . This usually take constant time. We now compare $A[i]$ with every element $A_i[j]$ in A_i . If $A[i] \geq A_i[j]$ then we will do $A_i[j] = \text{MIN}$. Then we will find the smallest element $A_i[k]$ in A_i . This takes constant time using $n^{1+\epsilon}$ processors (or $O(\log \log n)$ time with n processors) for A_i [16,17]. For all $i=0, 1 \dots n-1$, this takes constant time with $n^{2+\epsilon}$ processors (or $O(\log \log n)$ time with n^2 processors). $A_i[k]$ is the smallest element larger than $A[i]$. Thus, we can make a link from $A[k]$ to $A[i]$.

Now we show our new algorithm which allows to sort n real numbers into a linked list in constant time with n^2 processors. We divide the input numbers into \sqrt{n} groups. So, now each group has \sqrt{n} numbers. Assign $n^{3/2}$ processors for each group. So now the total number of processors to do this will be $\sqrt{n} \times n^{3/2}$ processors which is n^2 processors. We already know that building a sorted linked list with $n^{3/2}$ processors of \sqrt{n} numbers take constant time. Now we have \sqrt{n} groups with sorted linked lists. Since we have \sqrt{n} groups there will be $O(n)$ pairs of groups in total. Let us assign n processors for every pair of groups. So, we require n processors $\times O(n)$ pairs which is $O(n^2)$ processors total. So, for every number in the group, we can use \sqrt{n} processors. So, we require n processors for each group. Now, let us say we have a number A in Group 1. It finds the smallest number B larger than it in Group 2 by comparing with every number in group 2 and using the sorted linked list already built for group 2. This process is repeated for all the pairs of groups like Group 1, Group 3 and Group 1, Group 4 etc. We find $\sqrt{n} - 1$ smallest numbers larger than A . In general, if we do it in parallel each number find $\sqrt{n} - 1$ smallest numbers larger than it. Each number then uses n processors to find the minimum among these $\sqrt{n} - 1$ smallest numbers in constant time [16,17]. So, in total the proposed algorithm uses n^2 processors to sort the n real numbers in a linked list in constant time.

Theorem 1. n real numbers can be sorted into a linked list in constant time using n^2 processors on the Common CRCW PRAM.

Finally, let us discuss about the algorithm which is used to sort the real numbers in the linked list using less than n^2 processors. Divide n numbers into n/t groups with t numbers in each group. First sort the t numbers in each group into a linked list in constant time using $(n/t)t^2$ processors. Now for about every m nodes (between m and $2m$ nodes), we build a supernode. Initially we have n/t linked lists. Each linked list has t nodes. Combine about every consecutive m nodes to form a supernode. We have t nodes in linked list so we have $O(t/m)$ super nodes. This can be down in $O(n/p + \log^{(c)} n \log t)$ time [1,8,9], where

$\log^{(1)}n = \log n$ and $\log^{(c)}n = \log \log^{(c-1)}n$. The t/m supernodes for each sorted link of t nodes forms a sorted supernode linked list. Two supernode sorted linked lists with t/m nodes each can be merged into one lined list in constant time using $(t/m)^2$ processors. Let us say supernode s in one supernode linked list is to be inserted between supernode s_1 and supernode s_2 of the other supernode linked list. Then s uses $O(m)$ processors to compare it with every nodes in s_1 and s_2 to find the exact position it needs to be inserted. Now merge every pair of about m nodes using m^2 processors in constant time.

Finally, let us discuss about the algorithm which is used to sort the real numbers in the linked list using less than n^2 processors. Divide n numbers into n/t groups with t numbers in each group. First sort the t numbers in each group into a linked list in constant time using $(n/t)t^2$ processors. Now for about every m nodes (between m and $2m$ nodes), we build a supernode. Initially we have n/t linked lists. Each linked list has t nodes. Combine about every consecutive m nodes to form a supernode. We have t nodes in linked list so we have $O(t/m)$ super nodes. This can be down in $O(n/p + \log^{(c)}n \log t)$ time [1,8,9], where $\log^{(1)}n = \log n$ and $\log^{(c)}n = \log \log^{(c-1)}n$. The t/m supernodes for each sorted link of t nodes forms a sorted supernode linked list. Two supernode sorted linked lists with t/m nodes each can be merged into one lined list in constant time using $(t/m)^2$ processors. Let us say supernode s in one supernode linked list is to be inserted between supernode s_1 and supernode s_2 of the other supernode linked list. Then s uses $O(m)$ processors to compare it with every nodes in s_1 and s_2 to find the exact position it needs to be inserted. Now merge every pair of about m nodes using m^2 processors in constant time.

Therefore there are $(n/t)^2$ pairs of linked lists. For every pair, we use $(t/m)^2$ processors to merge supernode linked lists. So, we use $(n/m)^2$ processors for merging the supernodes. For each supernode s we used nm/t processors (m processors for each of the n/t pairs) for comparing it with the nodes in other supernodes. Because we have n/m supernodes, therefore the process used is n^2/t processors. For merging the m nodes in one supernode list with m nodes in other supernodes list we used $(n/t)^2(t/m)m = (n/m)(n/t)m = n^2/t$ processors and $\log m$ time. If we let $m^2 = t$ then we used n^2/t processors and $\log t$ time.

The two extremes are $t=1$ which we use n^2 processors and sort real numbers into a linked list in constant time and when $t=n$ where we use n processors and sort real numbers into a linked list in $\log n$ time.

Theorem 2: n real numbers can be sorted into a linked list in $O(\log t)$ time with n^2/t processors, where t can range from 1 to n .

1. Prepare for Sorting Real Numbers into a Linked List Using $n^2/\log^c n$ Processors.

A parallel algorithm for sorting n input real numbers into a linked list in constant time is described. This algorithm works by grouping input real numbers, let us say, splitting $A[0 \dots n-1]$ real numbers into $n/\sqrt{\log n}$ groups. We enumerate all permutations of the $\sqrt{\log n}$ numbers in every group. Among all these $\sqrt{\log n}!$ permutations there is only one permutation in

which these $\sqrt{\log n}$ numbers are in sorted order (assuming that all input numbers are different). For each permutation of the numbers in a group we use $\sqrt{\log n}$ processor (one processor for each number) and therefore we used $\sqrt{\log n}! * \sqrt{\log n}$ processors for each group and for the n input real numbers we used $(n/\sqrt{\log n}) * \sqrt{\log n}! * \sqrt{\log n} = n\sqrt{\log n}!$ processors. For each group the permutation with the sorted order of numbers is selected in constant time by verifying the $\sqrt{\log n}$ numbers are in sorted order using $\sqrt{\log n}$ processors. This is how internal sorting is carried out.

To continue the sorting process, each element e in a group G is compared to the elements in the next group G_i , $0 \leq i < \sqrt{\log n}$, and fitted in a suitable position by determining its rank in G_i . This is done by using $\sqrt{\log n}$ processors to compare it to every number in the (sorted) group G_i . e then enumerates $(1 + \sqrt{\log n})^{\sqrt{\log n}}$ possibilities using $\sqrt{\log n}$ base $\sqrt{\log n}$ digits. There are $(1 + \sqrt{\log n})^{\sqrt{\log n}}$ patterns in these digits. The pattern $a_0 a_1 \dots a_{\sqrt{n}-1}$ denotes that e has rank a_i in G_i . Associated with pattern $a_0 a_1 \dots a_{\sqrt{n}-1}$ is the pre-computed value $a_0 + a_1 + \dots + a_{\sqrt{n}-1}$ which is the rank of e in $G_0 \cup G_1 \cup \dots \cup G_{\sqrt{\log n}-1}$. For each permutation e then uses $\sqrt{\log n}$ processors with the i -th processor p_i to verify whether the rank of e in group G_i is a_i . If the rank is not a_i then p_i will cancel this permutation by (concurrent) write to a predefined memory cell for this permutation. Thus only one permutation is not cancelled and the rank precomputed for this permutation is fetched. This determines the rank of e in $G_0 \cup G_1 \cup \dots \cup G_{\sqrt{\log n}-1}$. e used $(1 + \sqrt{\log n})^{\sqrt{\log n}} * \sqrt{\log n}$ processors. Thus for n real numbers the total number of processors used is $n * (1 + \sqrt{\log n})^{\sqrt{\log n}} * \sqrt{\log n}$. The time complexity is constant time.

In the next step we again combine $\sqrt{\log n}$ groups into one group. This time we have, for each number e , $(1 + \log n)^{\sqrt{\log n}}$ patterns because the rank of e in each group of $\log n$ numbers can be from 0 to $\log n$. Thus we will use $n * (1 + \log n)^{\sqrt{\log n}} * \log n$ processors.

For a positive integer c we will run the above process $2c$ times. Thus we will use $O(c)$ steps and use $n * (1 + (\log n)^c)^{\sqrt{\log n}} * (\log n)^c$ processors. We have sorted $(\log n)^c$ numbers in each of the $n/(\log n)^c$ groups.

EXAMPLE:

Let us now demonstrate our above approach using different numbers as an example. Assume $A[0, \dots, n-1]$ is the input array of n real values. Using $n * (1 + \log n)^{\sqrt{\log n}} * \log n$ processors, we achieve this in constant time. For example,

consider an input array of 2,3,8,6,12,19,5,4,1,0,9,7,10,18,16,13 where $n=16$.

In stage one of the process, we divide into groups depending on $\sqrt{\log n}$. When we solve, we get $\sqrt{\log 16}=2$, which represents two processors for each group. Following the procedure, we divided numbers into 8 groups with 2 numbers in one group, as indicated in [2,3],[8,6],[12,19],[5,4],[1,0],[9,7],[10,18],[16,13]. Each group is solved by determining the proper order from the all potential $\sqrt{\log n}!$ (For our example, $2!=2$.) permutations. Thus the first group will have two permutations: 2, 3, and 3, 2 and it determined that 2, 3 is in sorted order. The second group will have two permutations: 8, 6 and 6, 8 and it determined that 6, 8 is in the sorted order. And so on. Thus for each group we used 4 processors and the total number of processors used is $16/2*4=32$. The time is constant. After this stage we get [2,3],[6,8],[12,19],[4,5],[0,1],[7,9],[10,18], [13,16].

As we move on to stage II of the process, after we have completed internal group sorting, we will combine $\sqrt{\log n}=2$ groups into one group. To combine [2,3] and [6,8] into one group each of the 2, 3, 6, 8 will use 4 processors to determine its rank in each group. For example, 3 will use 4 processors, use 2 processors to determine its rank in [2,3] as 1 and use 2 processors to determine its rank in [6,8] as 0. Then for each number we form $(\sqrt{\log n} + 1)^{\sqrt{\log n}} = 3^2 = 9$ permutations: $p_0=00, p_1=01, p_2=02, p_3=10, p_4=11, p_5=12, p_6=20, p_7=21, p_8=22$ and use $(\sqrt{\log n} + 1)^{\sqrt{\log n}} \sqrt{\log n} = 9*2=18$ processors, two processors for each permutation. Thus 3 use 2 processors to check p_0 and finds that p_0 is incorrect as it indicates that 3 has rank 1 in [2,3] and rank 0 in [6,8]. Thus p_0 will be taken out of consideration (crossed out). The only permutation that is not crossed out is $p_3=10$ as it indicates that 3 has rank 1 in [2,3] and rank 0 in [6,8]. Thus 3 picks the pre-computed rank of $0+1=1$ for p_3 . Thus at the end of this stage we got [2,3,6,8],[4,5,12,19],[0,1,7,9],[10,13,16,18].

1. Sorting Real Numbers into a Linked List with $n^2/\log^c n$ processors.

For a given total number of “n” inputs, dividing them into $n/(\log n)^c$ groups with $(\log n)^c$ numbers in each group. As described in the Section 3 we use $n * (1 + (\log n)^c)^{\sqrt{\log n}} * (\log n)^c$ processors to sorted $(\log n)^c$ numbers in each of the $n/(\log n)^c$ groups in constant time.

For each group of sorted $(\log n)^c$ numbers we sample every $(\log n)^d$ -th number (with $0 < d < c$) and thus we sampled $(\log n)^{c-d}$ numbers from each group and among the n input numbers we sampled $n/(\log n)^d$ numbers.

We now sort these $n/(\log n)^d$ numbers into a sorted linked list in constant time using $(n/(\log n)^d)^2$ processors using the algorithm we described earlier .

Now for each number a we use $n/(\log n)^d$ processors to compare it to all the numbers on the sorted linked list and finds: the largest number smaller than a and l: the smallest number larger than a among numbers in the sorted linked list. Because numbers in the linked list are sorted and therefore s and l can be found in constant time. s and l are neighboring elements on the sorted linked list.

Now between two neighboring elements s and l on the sorted linked list there can be at-most $n/(\log n)^d$ numbers fell in between. This is because for each group of sorted $(\log n)^c$ numbers there can be at most $(\log n)^d$ numbers fell between s and l. For otherwise if more than $(\log n)^d$ numbers fell in between s and l then there is at least sampled number fell in between s and l because we sampled every $(\log n)^d$ -th number from these $(\log n)^c$ numbers. But between s and l there is no another sampled number.

Thus there are at most $n/(\log n)^d$ numbers fell in between s and l.

Now for all numbers fell in between s and l we sort them into a linked list. We use $n/(\log n)^d$ processor for each number (thus we used a total of $n^2/(\log n)^d$ processors). Because there no more than $n/(\log n)^d$ numbers between s and l and therefore we have at least m^2 processors for the m numbers between s and l. Thus we can sort the numbers between s and l into a sorted linked list in constant time use the algorithm we described earlier .

After the numbers in each interval between s and l are sorted into a linked list we can connects these linked lists into one sorted linked list.

Because c is an arbitrarily large constant and $d < c$ thus d can be an arbitrarily large constant. Thus we have:

We have been able to optimize the existing algorithms with less number processors and time. Earlier, we showed algorithms like sorting of n real numbers into a linked list in constant time using n^3 processors and sorting of n real numbers into a linked list in $O(\log \log n)$ time using n^2 processors. Here we improved them to:

Theorem 3: n real numbers can be sorted into a linked list in constant time with $n^2/\log^c n$ processors on the COMMON CRCW PRAM

CONCLUSION

We discussed about sorting n real numbers into a linked list using $o(n^2)$ processors in constant time. We have followed the approach to assign the processors by dividing the given input into groups. The approaches of solving correct order from

potential permutations and finding rank made algorithm work efficiently to sort the given input of array.

Currently we do not know how to reduce the number of processors further to reach constant time for sorting real numbers into a linked list. The problem is that after we sorted real numbers into a linked list we cannot sample every k-th number in constant time because sorted numbers are on linked list and not in an array.

REFERENCES

1. R. Anderson, G. Miller. Deterministic parallel list ranking. *Algorithmic*, Vol. 6, 859-868(1991).
2. P. Beame, J. Hastad. Optimal bounds for decision problems on the CRCW PRAM. *Proc. 1987 ACM Symp. On Theory of Computing (STOC'1987)*, 83-93(1987).
3. P.C.P. Bhatt, K. Diks, T. Hagerup, V.C. Prasad, T. Radzik, S. Saxena. Improved deterministic parallel integer sorting. *Information and Computation*, 94, 29-47(1991).
4. S. A. Cook. Towards a Complexity Theory of Synchronous Parallel Computation. *L' Enseignement Mathématique*, 27, 99-124(1981).
5. T. Goldberg, U. Zwick. Optimal deterministic approximate parallel prefix sums and their applications. *Proc. 3rd. Israel Symp. On Theory and Computing Systems*, 220-228(1995).
6. T. Hagerup. Towards optimal parallel bucket sorting. *Information and Computation*. 75, 39-51(1987).
7. Y. Han, N. Goyal, H. Koganti. Sort Integers into a Linked List. *Computer and Information Science*. Vol. 13, No.1, 51-57(2020).
8. Y. Han, P. Kasani. Sorting real numbers into a linked list on the PRAM model. *{\it Proceedings of the 2021 Int. Conf. on Life Sciences, Engineering and Technology}*. 45-49(2021).
9. Y. Han, P. Kasani. Time processor trade-off for sorting real numbers into a linked list. *Proc. International Conference on Computation Structures and Algorithms*. 40-44(2021).
10. Y. Han, T. Sreevalli. Parallel merging and sorting on linked list. *International Journal of Computer and Information Technology (IJCIT)*. Vol. 10, No. 2, (March 2021), to appear.
11. Y. Han. Uniform linked list contraction. Paper 2002.05034 in arXiv.org.
12. Y. Han. Matching partition a linked list and its optimization. *Proc. 1989 ACM Symposium on Parallel Algorithms and Architectures (SPAA'89)*, 246-253 (June 1989).
13. Y. Han. Parallel algorithms for computing linked list prefix. *Journal of Parallel and Distributed Computing* 6, 537-557(1989).
14. J.J'aJ'a. *An Introduction to Parallel Algorithms*. Addison Wesley, Reading, MA, 1992.
15. R. M. Karp, V. Ramachandran, Parallel algorithms for shared-memory machines. In *Handbook of Theoretical Computer Science (Vol. A): Algorithms and Complexity*, J. van Leeuwen, Ed., New York, NY: Elsevier, 869-941(1991).
16. C. P. Kruskal. Searching, merging, and sorting in parallel computation. *IEEE Trans. Comput.*, C-32, 942-946(1983).
17. L. G. Valiant. Parallelism in comparison problems. *SIAM J. on Computing*, Vol. 4. No. 3, 348-355(1975).