Sorting Real Numbers into a Linked List on the PRAM Model

Yijie Han
School of Computing and Engineering
University of Missouri at Kansas City
Kansas City, MO 64110
hanyij@umkc.edu

Pruthvi Kasani
School of Computing and Engineering
University of Missouri at Kansas City
Kansas City, MO 64110
pruthvikasani@mail.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(loglogn) time.**

Keywords—parallel algorithms, parallel sorting, CRCW (Concurrent Reat Concurrent Write), EREW (Exclusive Read Exclusive Write), CREW (Concurrent Read Exclusive Write).

I. INTRODUCTION

In this paper we study parallel sorting of real numbers into a linked list. The computation model we used for our algorithm is the PRAM (Parallel Random Access Machine) [1]. There are EREW (Exclusive Read Exclusive Write) PRAM, the CREW (Concurrent Read Exclusive Write) PRAM and the CRCW (Concurrent Read Concurrent Write) PRAM [1]. Each of these submodels differentiates themselves on how the memory is shared among all the processors in PRAM. On the EREW PRAM, at any step no more than one processor can either read or write on a memory cell. On the CREW PRAM, one or more processors can simultaneously read a memory cells in a step but no more than one processor can write a memory cell in a step. Whereas in CRCW PRAM, multiple processors can read or write on a memory cell in a step. Since CRCW allows multiple processors to read or write on a single memory cell, there are some arbitrary schemes designed to perform the actions. On the Priority CRCW PRAM, the processor having the highest priority wins the write on the memory cell among the processors writing to the memory cell. The priority can be the index of the processor. On the Arbitrary CRCW PRAM an arbitrary processor wins among the processors to write on the memory cell. On the Common CRCW PRAM when multiple processors write the same memory cell in one step, they must write the same value and that value is written into the memory cell. Among all the CRCW PRAM, Priority CRCW is the strongest model, Arbitrary CRCW PRAM is weaker than the Priority CRCW PRAM, and Common CRCW PRAM is the weakest among the three. In this paper we will use the Common CRCW PRAM. Because our algorithm runs on the Common CRCW PRAM and thus they can run on the Arbitrary and Priority CRCW PRAM.

Let T_p be the time complexity of a parallel algorithm using 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 $pT_p=T_1$ then this parallel algorithm is an optimal parallel algorithm.

When we have a T_P time algorithm using P processors, then when we use p processors the time can be expressed or translated as $T_PP/p+T_P$.

A parallel algorithm for a problem of size n using polynomial number processors (i.e. n^c processors for a constant c) and running in polylog time (i.e. $O(\log^c n)$ time for a constant c) is regarded as belong to the NC class [2], where NC is Nick's class.

Researchers in parallel algorithm field are working to achieve NC algorithms and fast and efficient parallel algorithms.

In this paper, we will study sorting real numbers into a linked list in constant time using n^2 processors. Previously it is known that n real numbers can be sorted into a linked list in O(loglogn) (constant time) using n^2 (n^3) processors [9,10,11].

It is known that sorting n real numbers into an array takes at least $\Omega(\log n/\log\log n)$ time on the CRCW PRAM with polynomial number of processors [3]. It takes at least $\Omega(\log\log n)$ time if we are to sort them into a padded array [4]. However if we are going to sort them into a linked list we show here that it can be done in constant time. Thus the lower bound of $\Omega(\log n/\log\log n)$ [3] and the lower bound of $\Omega(\log\log n)$ [4] are really the lower bound for arranging numbers in an array instead of the lower bound of "sorting" them.

There are results before for sorting integers into a linked list [5, 6]. It is known there that n integers in {0, 1, …, m-1} can be sorted into a linked list in constant time using nlogm processors. m here cannot be bounded by functions of n. Except our previous results for sorting real numbers into a linked list [6,7,8] we do not know other results for parallel sorting real numbers into a linked list and we do not know previous results of sorting real numbers in constant time.

II. SORTING REAL NUMBERS INTO A LINKED LIST

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 in linked list using constant time using n^3 processors. Let us say, A[0.....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 [7,8] 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(loglogn) 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(loglogn) time [7,8]. Let us say A[0....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 minimum 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 $A_i[j]$ =MIN. Then we will find the minimum element $A_i[k]$ in A_i. This takes constant time using $n^{1+\varepsilon}$ processors (or O(loglogn) time with n processors) for A_i [10.11]. For all i=0, 1… n-1, this takes constant time with $n^{2+\varepsilon}$ processors (or O(loglogn) 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 $\sqrt{n}$ x $n^{3/2}$ 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 x 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 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 [7,8]. So, in total the proposed algorithm uses n^2 processors to sort the n real numbers in a linked list in constant time.

III. THEOREM

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

We have been able to optimize the existing algorithms with lesser number processors and time. Earlier, we had algorithms like sorting of n real numbers in constant time using n^3 processors and sorting of n real numbers in O(loglogn) time using n^2 processors [9, 10, 11].

IV. CONCLUSIONS

We discussed about sorting n real numbers into a linked list using n^2 processors in constant time. This algorithm is more effective than the ones that require n3 processors to sort into a linked list in constant time and n^2 processors to sort into a linked list in O(loglogn) time. We have followed the approach to assign the processors by dividing the given input into groups. The most interesting part of this algorithm is that we were able to sort the n real numbers in the linked list by decreasing the number of processors from n^3 to n^2 and also by achieving this in constant time.

It looks to us that reducing the number of processors further while still achieving constant time is not trivial. A plausible way of doing this is to convert real numbers into integers while utilizing advantage integers bring to sorting. We have not been achieved further results along this direction.

V. REFERENCES

[1] 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).

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

[3]. 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).

[4]. 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).

[5]. 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).

[6]. T. Hagerup. Towards optimal parallel bucket sorting. Information and Computation. 75, 39-51(1987).

[7]. C. P. Kruskal. Searching, merging, and sorting in parallel computation. IEEE Trans. Comput., C-32, 942-946(1983).

[8]. L. G. Valiant. Parallelism in comparison provlems. SIAM J. on Computing, Vol. 4. No. 3, 348-355(1975).

[9]. N. Goyal. An Arbitrary CRCW PRAM Algorithm for Sorting Integers into the Linked List and Chaining on a Trie. Master's Thesis. University of Missouri at Kansas City. 2020.

[10] Y. Han, N. Goyal, H. Koganti. Sort Integers into a Linked List. Computer and Information Science. Vol. 13, No.1, 51-57(2020).

[11]. 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.