SORTING OF REAL NUMBERS INTO A LINKED LIST ON THE PRAM MODEL

A Thesis in

Computer Science

Presented to Faculty of University of
Missouri - Kansas City in partial fulfillment of
requirements for the degree

MASTER OF SCIENCE

By

Pruthvi Kasani

B. Tech in Computer Engineering
Indian Institute of Information Technology, Chennai, India, 2015

Kansas City, Missouri, United States of America

2022

SORTING OF REAL NUMBERS INTO A LINKED LIST ON THE PRAM MODEL

Pruthvi Kasani, Candidate for the Master of Science Degree

University of Missouri – Kansas City

ABSTRACT

We study the sorting of real numbers into a linked list on the PRAM (Parallel Random Access Machine) model. The research work consists of two parts. First part talks about the various techniques involved in sorting the real numbers on the linked list in terms of number of processors and time complexity. We have examined on how to sort the real numbers in the linked list using n^3, n^2 processors which has the time complexity of constant time and O(loglogn) time respectively. We have done good research in that area to come up with an algorithm to sort n real numbers into the linked list using n^2 processors in constant time. In second part, we talk about the time processor trade off for sorting the real numbers in the linked list.

APPROVAL PAGE

The faculty listed below, appointed by Dean of School of Computing and Engineering, must examine the thesis titled as “Sorting of Real Numbers into a Linked List on the PRAM Model” presented by Pruthvi Kasani, candidate for the Master of Science degree, and certify that in their opinion it is worthy of acceptance.

Supervisory Committee

Yijie Han, Ph. D., Chair
School of Computing and Engineering

Deb Chatterjee, Ph. D.
School of Computing and Engineering

Sejun Song, Ph. D.
School of Computing and Engineering

CONTENTS

ABSTRACT iii

Chapter

1. INTRODUCTION 1
2. METHODOLOGY 4
3. THEOREM 8
4. CONCLUSIONS 9

REFERENCES 10

VITA 12

CHAPTER 1

INTRODUCTION

In this thesis report, 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) [12]. There are EREW (Exclusive Read Exclusive Write) PRAM, the CREW (Concurrent Read Exclusive Write) PRAM and the CRCW (Concurrent Read Concurrent Write) PRAM [12]. Each of these sub models 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 cell 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 report, we would use the Common CRCW PRAM. Because our algorithm runs on the Common CRCW PRAM and thus they could 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 [3], 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 report, we would 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 [5,7,8].

It is known that sorting n real numbers into an array takes at least W(logn/loglogn) time on the CRCW PRAM with polynomial number of processors [1]. It takes at least W(loglogn) 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 could be done in constant time. Thus, the lower bound of W(logn/loglogn) [1] and the lower bound of W(loglogn) [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 [2, 6]. It is known there that n integers in {0, 1, …, m-1} could 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 [6,10,11], 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.

In the later part of the research report, our algorithm is a time processor tradeoff for sorting real numbers into the linked list. We show that we can spent a little more time but reduce the number of processors to less than n^2

CHAPTER 2

METHODOLOGY

Sorting of N Real numbers into 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 [10,11] 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, 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 could be

done in O(loglogn) time [10,11]. 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 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 $A_i[j]$ =MIN. Then we will find the smallest element $A_i[k]$ in A_i. This takes constant time using n^{1+e} 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+e} 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 numbers larger than it. Each number then uses n processors to find the minimum among these $\sqrt{n}-1$ smallest numbers in constant time [10,11]. So, in total the proposed algorithm uses n^2 processors to sort the n real numbers in a linked list in constant time.

Time Process Trade off for Sorting Real Numbers in Linked List

Finally, let us discuss about the algorithm which is used to sort the real numbers in the linked list using less then 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 [13][11], 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 logm time. If we let $m^2=t$ then we used n^2/t processors and logt 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 logn time.

CHAPTER 3

THEOREM

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

We have been able to optimize the existing algorithms with lesser number processors and with relaxed time. Earlier, we had algorithms like sorting of n real numbers into a linked list in constant time using n^3 and sorting of n real numbers in O(loglogn) time using n^2 processors [8]. We also came up with an algorithm to sort the n real numbers in linked list using less than n^2 processors [8].

CHAPTER 4

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 n^3 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 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 using advantage integers bring to sorting. We have not been achieved further results along this direction.

REFERENCES

[1]. 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).

[2]. 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).

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

[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]. 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.

[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”, Proc. of the 2021 Int. Conf. on List Science, Engineering and Technology, 45-49(2021).

[9]. 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.

[10]. Y. Han, “Uniform linked list contraction”, Paper 2002.05034 in arXiv.org.

[11]. R. Anderson, G. Miller, “Deterministic parallel list ranking”, Algorithmica, Vol. 6, 859-868(1991).

[12]. 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).

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

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

VITA

Pruthvi Kasani was born in Vijayawada, Andhra Pradesh, India on August 6th 1997. He attended elementary school named as Nirmala High School and graduated as the top scorer of that academic year 2013. Later, he got admitted into one of the top colleges in the nation and finished his bachelor's from "Indian Institute of Information Technology, Chennai". In the year 2015, he was placed in a MNC called ADP and worked for a year and half. During Jan 2021, he entered University of Missouri Kansas City to complete his Master of Science Degree in Computer Science in May 2022.