AN ARBITRARY CRCW PRAM ALGORITHM FOR SORTING INTEGERS INTO A LINKEDLIST AND CHAINING ON A TRIE

A THESIS IN
Computer Science

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

MASTER OF SCIENCE

By
NIKITA GOYAL

Bachelor of Technology, Chitkara University,
Punjab, India, 2018

Kansas City, Missouri
2020

AN ARBITRARY CRCW PRAM ALGORITHM FOR SORTING INTEGERS INTO A LINKEDLIST AND CHAINING ON A TRIE

Nikita Goyal, Candidate for the Master of Science Degree
University of Missouri – Kansas City, 2018

ABSTRACT

The research work comprises of two parts. Part one is using an Arbitrary CRCW PRAM algorithm for sorting integers into a linked list. There are various algorithms and techniques to sort the integers in LinkedList. Arbitrary CRCW PRAM model, being the weakest model is able to sort n integers in a LinkedList in "constant time" using nlogm processors and if we use nt processors, then it can be sorted in O(loglogm/logt) time by converting Arbitrary CRCW PRAM model to Priority CRCW PRAM model.

Part two is Chaining on a Trie. This research paper solves the problem of chaining on a Trie by providing more efficient complexity. This Algorithm takes "constant time" using n(logm+1) processors to chain the nodes on a Trie for n input integers on the Arbitrary CRCW PRAM model.

APPROVAL PAGE

The faculty listed below, appointed by the Dean of School of Computing and Engineering, have to examine the thesis titled “An Arbitrary CRCW PRAM Algorithm for Sorting Integers into a LinkedList and Chaining on a Trie” presented by Nikita Goyal, 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

Kendal Bingham
Department of Computer Science Electrical Engineering

Sejun Song, Ph.D.
Department of Computer Science Electrical Engineering

CONTENTS

ILLUSTRATIONS

Figure Page

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank following people who have directly or indirectly helped me in academic achievements. Firstly, I would like to thank Dr. Yijie Han for the constant and endearing support which has helped me in fulfilling my thesis. He has provided me with an opportunity to realize my potential in the field of my thesis. His encouragement and inputs were elements of vital guidance in my thesis. He has been a constant source of motivation and challenged me with algorithms, deadlines, that have contributed to me acquiring inspiration and ideology. His expertise and innovative insights have been phenomenal in completing my thesis.

I sincerely thank Prof. Kendal Bingham and Dr. Sejun Song for accepting to be a part of my thesis committee and making time for me from their busy schedule. I would like to thank the University of Missouri- Kansas City for providing me with an opportunity to continue my research and supporting me in this regard.

I would like to dedicate my thesis to my parents who constantly inspired me to pursue higher studies. I would like to thank my family who stood behind me all these years during my degree. Finally, I would like to thank all my teachers, educational administrators, present, and past and all who helped me achieve this academic goal.

CHAPTER 1

INTRODUCTION

Algorithms and their complexities play a vital role in determining the efficiency of software to perform day-to day task on computer. The serial processing takes more time to perform the same task than Parallel processing because in serial processing, one task is completed at a time and the remaining tasks are executed in a sequence by the processor whereas in Parallel processing, multiple tasks are completed at the same time by using different processors. This research work is based on PRAM (Parallel Random-Access Machine). This is used by parallel-algorithm designers to model parallel algorithm performances which are their time and space complexities. There are four PRAM models that is EREW (Exclusive read exclusive write), CREW (Concurrent read exclusive write), ERCW (Exclusive read concurrent write) and CRCW (concurrent read concurrent write). CRCW is the strongest model among all of them. Arbitrary PRAM CRCW is used in both the papers to increase the efficiency of existing algorithms. Concurrent write is further divided into Common, Arbitrary and Priority. Common concurrent write allows all processors to write the same value, Arbitrary concurrent write allows only one arbitrary attempt to be successful to write in memory and Priority concurrent write decide on processors rank who will write in memory cell. This research includes Arbitrary CRCW which have been used in the both the papers indicating that it assigns any processor to write randomly in the memory cell. We have used a binary trie to show the results.

For any sequential algorithms to sort the integers in a linked list, take O(nlogn) time. Ω(logn/loglogn) is a time lower bound for sorting integers [1]. This time lower bounds does not need to hold to sort integers into the linked list. Sorting always allows smaller integers to come earlier followed by the larger integers in the LinkedList. The n integers can be sorted into a linked list in constant time using nlogm processors on the priority CRCW PRAM model and they can be sorted into a linked list in O(loglogm/logt) time using nt processors [12]. This [12] paper was published as a research paper to "International Conference on Foundations of Computer Science (FCS'19), Las Vegas, USA." The Paper was accepted and presented in the conference with Paper Id FCS2478. This paper was published in Computer and Information Science, Vol 13 journal.

In Paper 1, to solve the same problem [12], sorting the integers in linked list has been done using Arbitrary CRCW PRAM model. We have converted the Arbitrary CRCW PRAM model to Priority CRCW PRAM model. The time complexity of the problem remains the same. As Arbitrary CRCW is the weakest model, it is better than Priority CRCW model.

In Paper 2, a binary trie, a data structure which has been used for the chaining problem to chain 1's in the input of n 0-1 bits following the orders of the input. In a binary trie, root node is empty. Radge has proven that chaining algorithms can be solved in α(n) time with linear operations using PRAM models [14]. Hagerup's algorithm [10] shows chaining in O(loglogm/logt) time using nt processors on the priority CRCW PRAM model. In this paper, it is shown that it is possible to do it in a constant time using n(logm+1) processors on the Arbitrary CRCW PRAM. We have used a trie of height logm which induces a tree of integers and chain each node in the trie with its two children plus the leaves of the tree to their ancestor with two children or the root on the trie.

CHAPTER 2

AN ARBITRARY CRCW PRAM ALGORITHM FOR SORTING INTEGERS INTO A LINKED LIST

2.1 Introduction

This chapter explains about the algorithm which we have worked upon under the guidance of Dr. Yijie Han. I have submitted the paper [12] to "International Conference on Foundations of Computer Science (FCS'19), Las Vegas, USA". The Paper was accepted and presented in the conference with Paper Id FCS2478. In coming few sections, we will discuss more about my research work algorithm.

Several algorithms are there to sort the integers into the linked list. It is difficult to find the most efficient algorithm. Sorting Integers in the linked list usually takes $\Omega(n\log n)$ time if we are considering linear processing of algorithm. If we consider the parallel computation, it takes $\Omega(\log n/\log\log n)$ time which is a lower bound for sorting integers [1]. However, this lower bound need not hold, if we sort integers into a linked list. Sorting integers into a linked list is to let smaller integers precede larger integers. In this paper, we are using Arbitrary PRAM model to sort the integers into the linked list by converting Arbitrary CRCW PRAM model into the Priority CRCW PRAM model.

Priority CRCW model is a strictly stronger model than the Arbitrary CRCW PRAM. Algorithms run on the Arbitrary CRCW PRAM cannot run on the Priority CRCW PRAM. If we have a Priority CRCW algorithm and an Arbitrary CRCW PRAM algorithm for the same problem and both algorithms have the same complexity, then the arbitrary CRCW PRAM algorithm is better than the Priority CRCW PRAM algorithm.

We know a Priority CRCW PRAM algorithm for sorting n integers in {0,1,...,m-1} in O(loglogm/logt) time using nt processors [12]. In this thesis, we show that this algorithm can be converted into an Arbitrary CRCW PRAM algorithm and therefore achieve an Arbitrary CRCW PRAM algorithm for the same problem. Because Arbitrary CRCW PRAM algorithm is better than the Priority CRCW PRAM algorithm, therefore our algorithm shown here is better than the previous algorithm [12].

There are many computation models like CRCW, EREW, CREW and ERCW. CRCW (Concurrent Read and Concurrent Write) computation model has been used for my research work which is the PRAM (Parallel Random Access Machine) model[8]. CRCW PRAM memory is a strongest model out of all the above. When we consider, Arbitrary CRCW PRAM model it indicates that the multiple processors try to write in the same memory cell, any random processor can win the write and write the data in the memory cell.

Arbitrary CRCW is a weaker model than the Priority CRCW PRAM model. We show an Arbitrary CRCW PRAM algorithm for sorting integers into a linked list using nt processors in O(loglogm/logt) time. We have used a trie of height logm to sort the n integers in the range {0,1,2...,m-1} into a linked list. A trie of logm height is a full binary tree with m leaves labeled by binary numbers from 0 to m-1 from left and right. A trie of 16 leaves have been shown in the Fig 1.

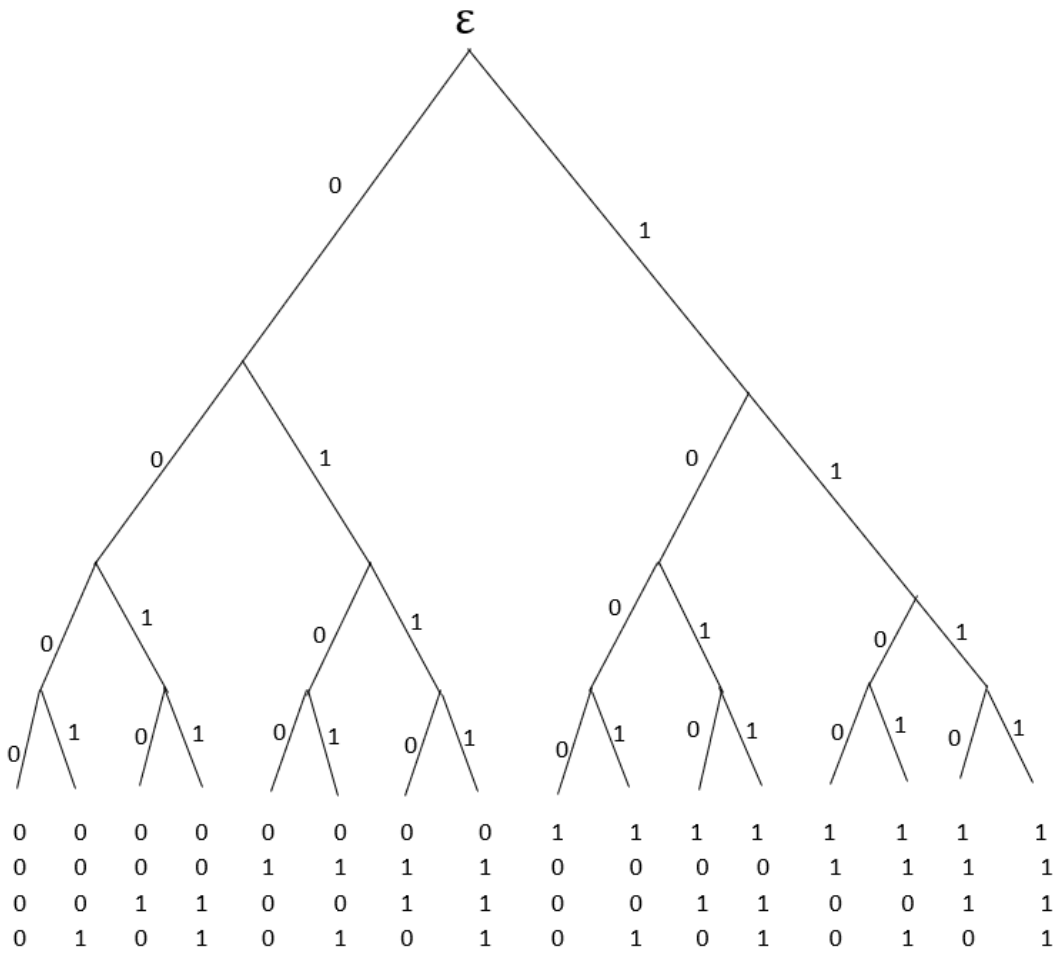

Fig1: Example of a Trie

We have a binary trie in the Fig 1. which is similar to binary tree[12]. It has left and right child. We take child as a node in the trie. The left is labeled with 0 and right is labeled with 1. The path from root to leaf node indicates the path that integer can follow. A trie has empty string on the top.

2.2 Forming a Trie

We have taken an input array of n integers in {0,1,....,n-1}. I have taken random distinct integers, for instance, 0,5,3,9,4,8 unsorted integers with their index number in an array. We cannot take duplicate integers. This figure has been taken in the reference of paper[12]. I have mentioned index number as a Processor number in an array,

Integer	0	5	3	9	4	8
Processor	0	1	2	3	4	5

Fig 2: Integers with their index numbers

I have marked the input integers and form the trie in the Fig3. In arbitrary CRCW model, an arbitrary processor among the processors writing into the same memory cell at a step can win the write in memory cell. In Fig 3, we used 3 processors for each integer and for an integer, integers are dropped in the bucket at a leaf “a” in the trie, used a k-th processor to write at the ancestor of “a” at level k [6].

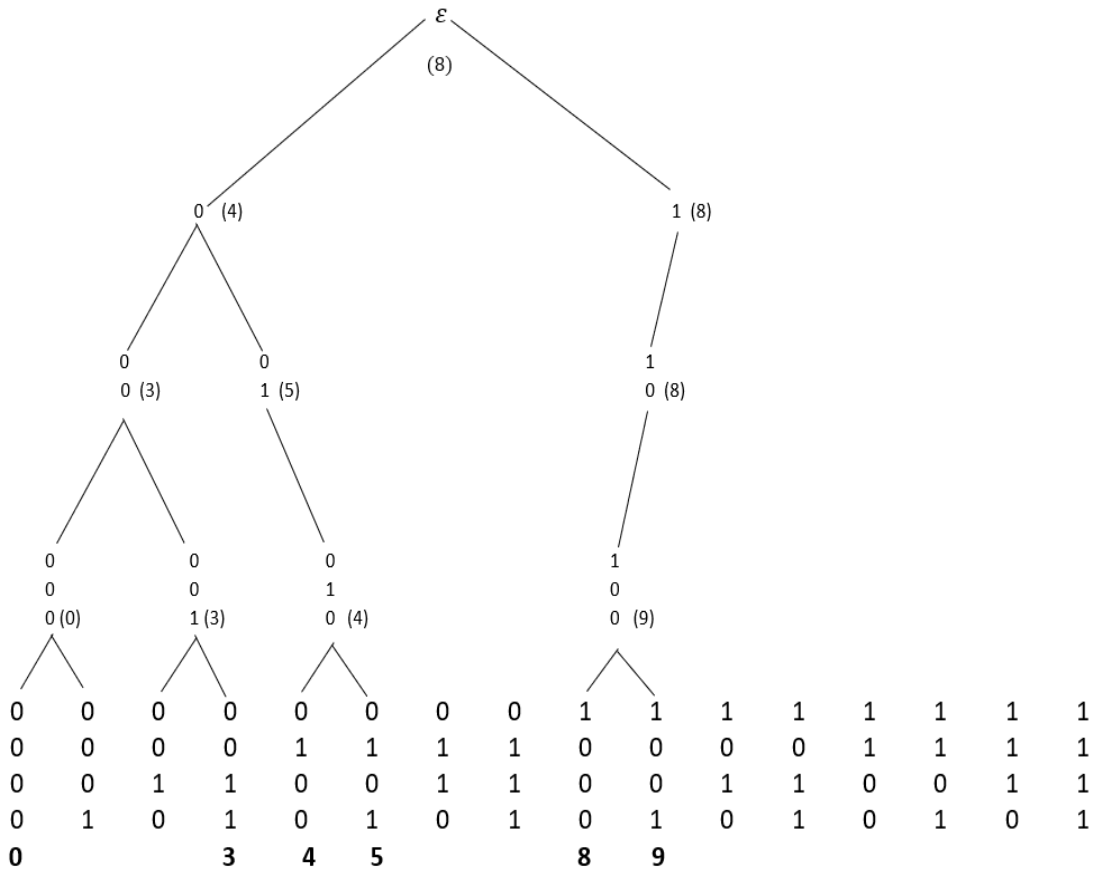

Fig 3: Integers are marked on a trie

Fig 3 depicts an arbitrary model in which at the bottom level any random integer from 4 and 5 can win at that node. We are assuming that 4 wins at that node at the bottom level. Similarly, for 8 and 9 integers, any integer can win and write in memory cell.

We assumed that 9 wins in the race of 8 and 9. If we go one level up in the trie, from 0 and 3, we assume 3 wins. This is how a trie is built from bottom to top and each integer wins randomly.

2.3 Finding the lowest winning node

To find the lowest winning node in the trie, we will take the lowest wining node of each integer. In the Fig4, bold highlighted wins show that the lowest winning nodes for each of the node in the trie. The wining ones is represented as **(a) wins** where a is an integer.

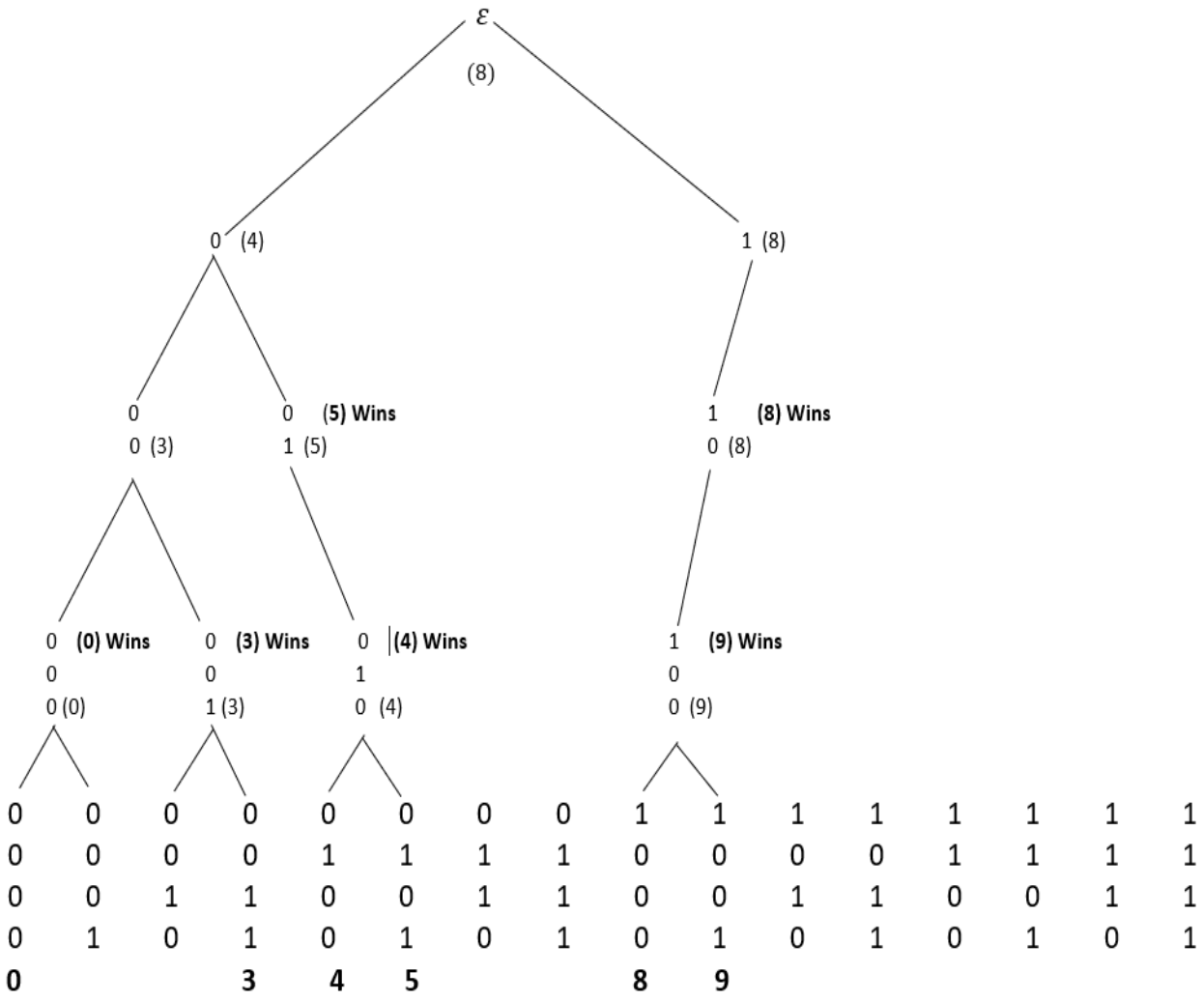

Fig 4: Lowest wining node

2.4 Finding the lowest Branch

Each lowest winning node will compare itself with all its ancestors and will find the lowest branching point. In Fig 4, 0 is compared with 3,4 and 8. The lowest branching point of 0 is at 3. Similarly, 3 is compared with all its ancestors which are 4 and 8. The lowest branching point of 3 is at node 4.

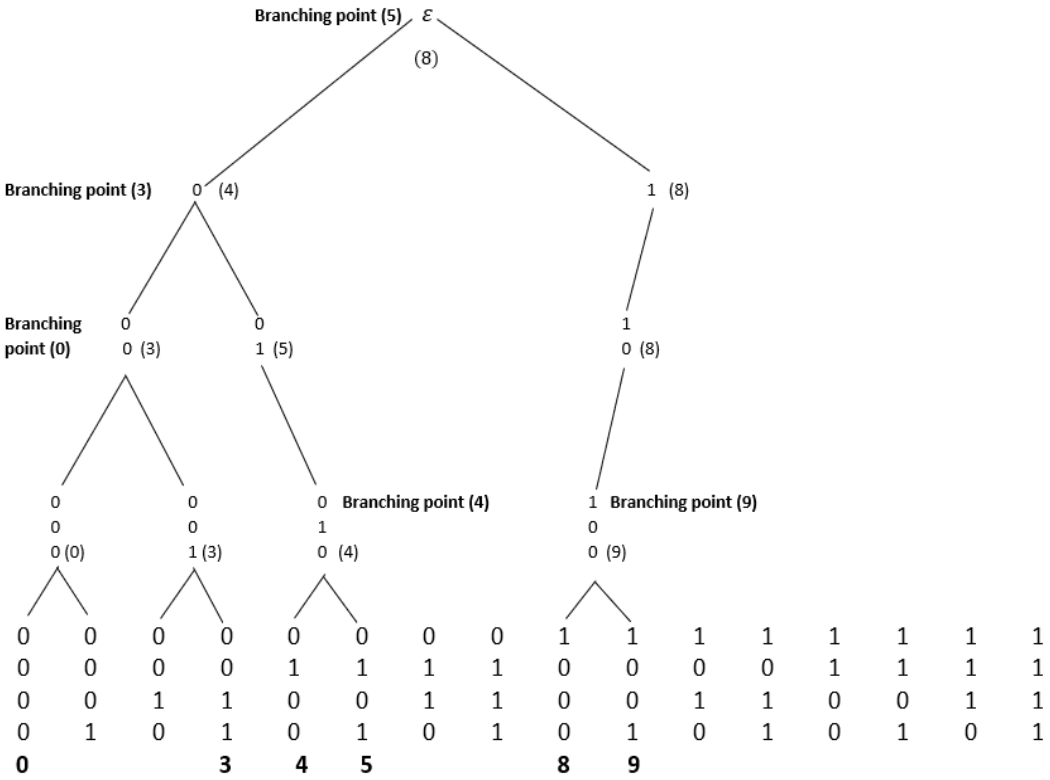

Fig 5: Finding the lowest branch

In Fig 5 all the branching points of each integer at winning node is shown. The branching point of 5 is at the top of the trie. To find the lowest branch for each winning node, we have to compare with all the ancestors above it.

2.5 Draw path to leaf node

The lowest branch point of each node is connected to its leaf. To show this, we have drawn a path from each branching point to its integers leaf. In Fig 6, red line is drawn to show the connection of branching point (5) to its integer at leaf node 5. Similarly, the branching points (3),(0),(4),(9) have been connected to their respective integers at leaf nodes.

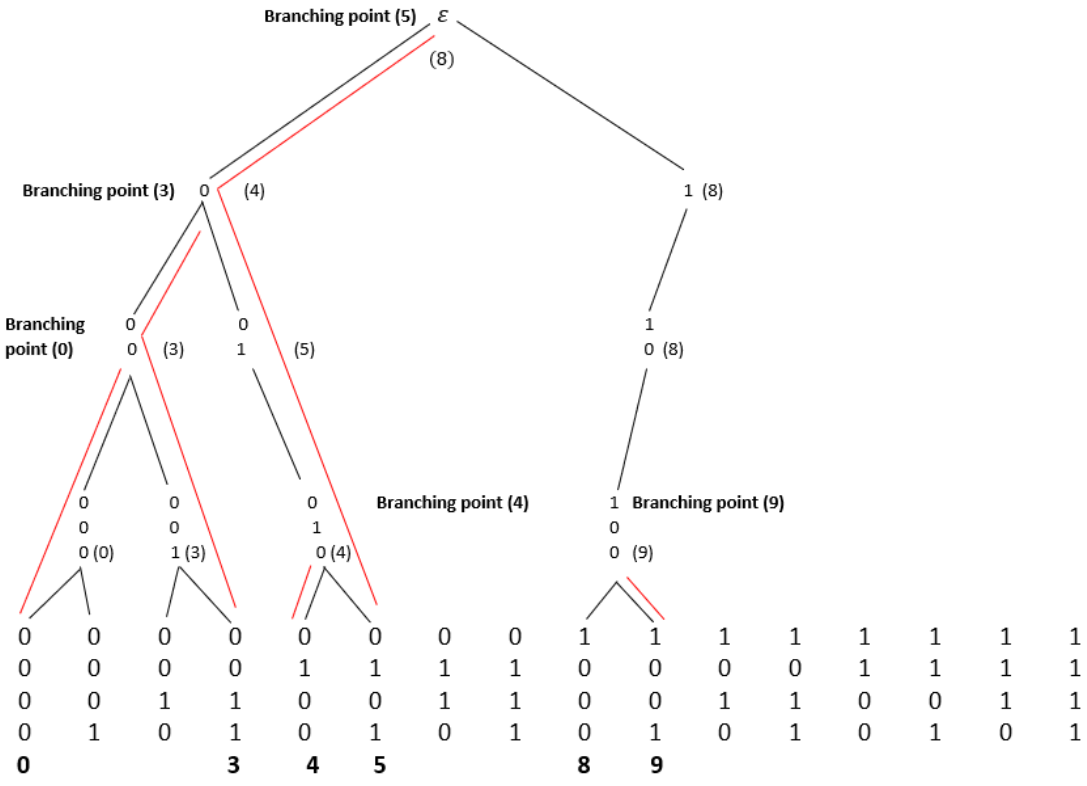

Fig 6: Connecting branching point integers to leaf

2.6 Finding the Highest Priority

The highest Prority was given to the node which is at the top of trie. The priority of each node was determined by starting from the top of the trie going to the bottom. So highest priority is of 5 as you can see in the Fig 7. So, it goes in the order 5,3,0,4 and 9.

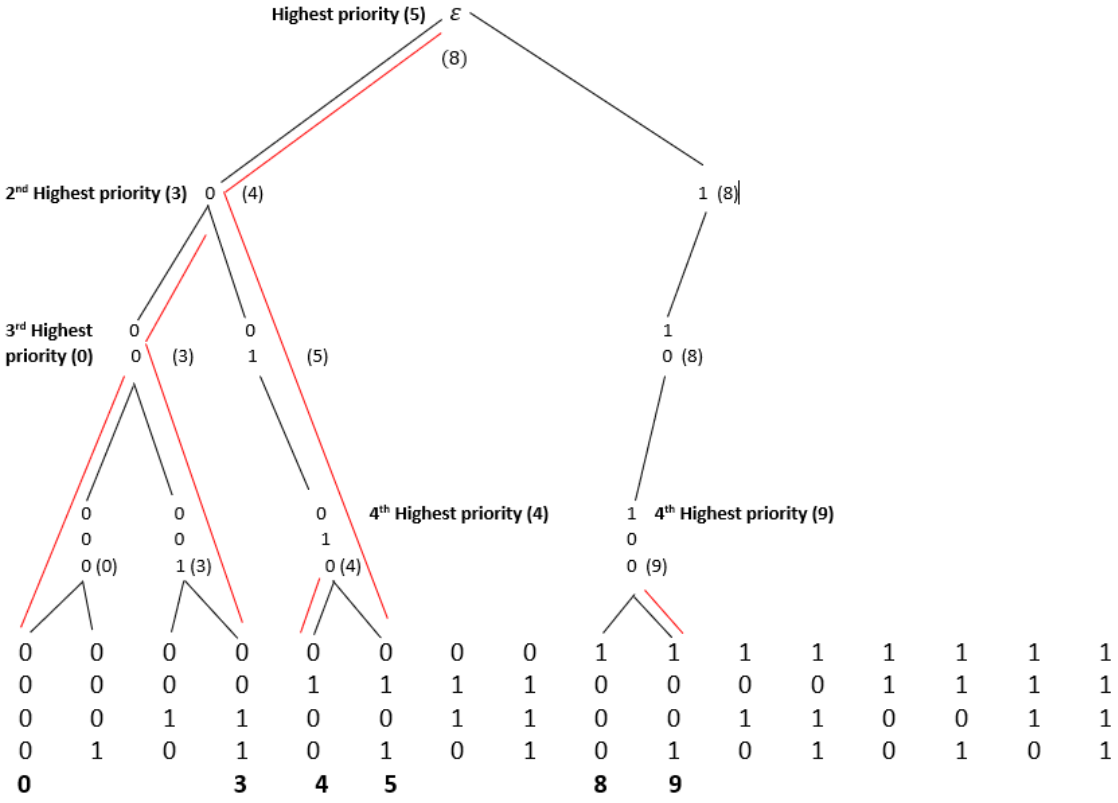

Fig 7: Highest priority of integers

We have converted the Arbitrary CRCW PRAM model into the Priority CRCW PRAM model. Now, the sorting of integers can be possible in the linked list using nt processors in O(loglogm/logt) [12].

CHAPTER 3

CHAINING ON A TRIE

3.1 Introduction

The 0-1 chaining problem is to chain 1's in the input of n 0-1 bits following the orders of the input. We study the parallel algorithm for this problem on the CRCW (Concurrent Read Concurrent Write) PRAM (Parallel Random-Access Machine) model [13].

On the CRCW PRAM model memory is shared among all processors and any processor can read or write to any memory cell in a step. When multiple processors write to the same memory cell in a step an arbitration mechanism is needed to decide what is written in the memory cell. On the Arbitrary CRCW PRAM model when multiple processors write to the same memory cell in a step an arbitrary one of them wins the write and writes its data into the memory cell. On the Priority CRCW PRAM when multiple processors write to the same memory cell the highest indexed processor wins the write. The Priority CRCW PRAM is a strictly stronger model than the Arbitrary CRCW PRAM model and therefore algorithm designed on the Arbitrary CRCW PRAM with the same complexity is better than the algorithm designed on the Priority CRCW PRAM.

Parallel algorithms can be measured by their time complexity T_P and processor complexity P. They can also be measured by their time complexity and operation complexity O which is the time processor product, i.e. $O=PT_P$. A parallel algorithm is called an optimal algorithm if $PT_P=O(T_1)$, where T_1 is the time of the best serial algorithm.

Ragde has shown that the chaining problem can be solved in $\alpha(n)$ time with linear operations [14] on the CRCW PRAM model, where $\alpha(n)$ is the inverse Ackermann function. In this paper we will study the chaining problem on the trie. The input in this problem is a set of n integers in {0, 1, …, m-1} placed at the leaves of the trie of height logm. The trie induces a tree of these integers and we want to chain each node in the trie on this tree with two children plus the leaves of the tree to their ancestor with two children or the root on the trie. This problem is intimately related to integer sorting. Hagerup's algorithm [10] will enable such chaining in O(loglogm) time with n processors on the ARBITRARY CRCW PRAM. Our recent results [11][12] shows how to do it in O(loglogm/logt) time using nt processors on the Priority CRCW PRAM. In this paper we show that it can be done in constant time using n(logm+1) processors on the Arbitrary CRCW PRAM.

3.2 The Algorithm

A trie is a full binary tree in which the root node is labeled with the empty string ε. A trie T with $m=2^k$ leaves can be represented by array A. A[0][0..m-1] are the m leaves of the trie. For $i>0$ and $0\leq j<m/2^i$, A[i][j] is an internal node of the trie with A[i-1][2j] as its left child and A[i-1][2j+1] as its right child. The edge from a parent to its left child is labeled with 0 and the edge from the parent to its right child is labeled with 1. We label node A[i][j] with a binary number j of $\log m/2^i$ bits. The root is labeled with ε. The height of a trie with m leaves will be logm.

The leaf nodes of the tris are labeled with 0, 1, ….m-1. We assume that logm is an integer. So, each leaf is represented by logm bits and the trie T has height logm and has logm+1 level. For example, suppose we have 3-bit integer so, logm=3, $m=2^3 = 8$. The trie will contain 8 leaf nodes. Fig. 1 shows such a trie with 8 leaf nodes, having height 3 and 4 levels.

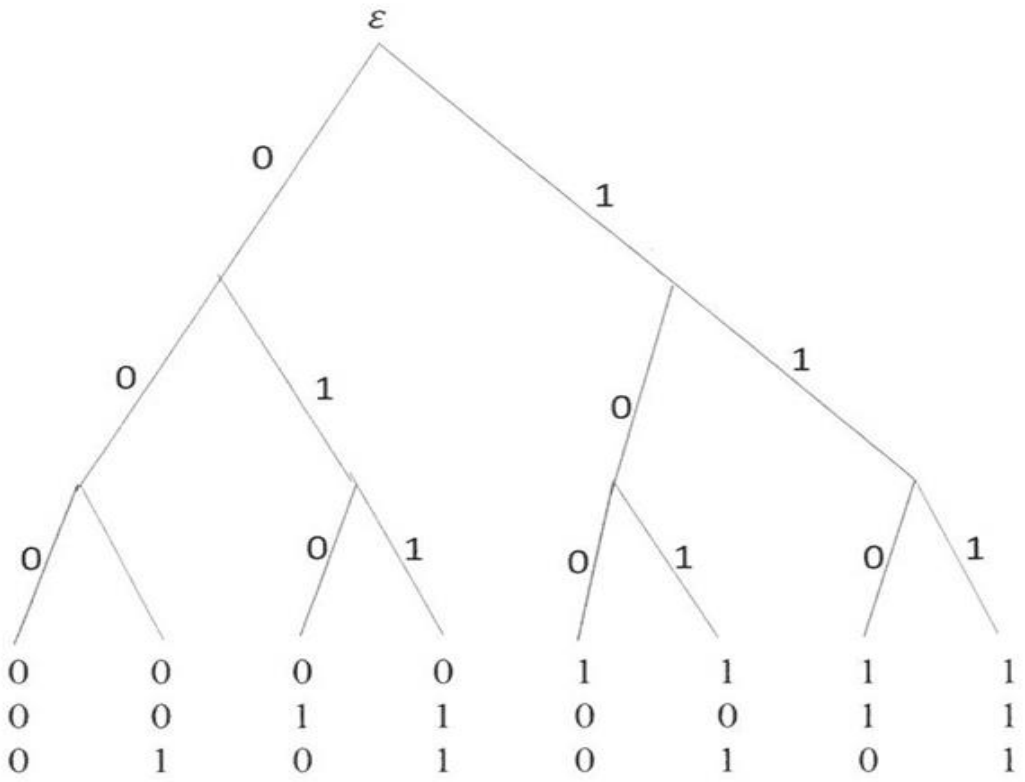

Fig 1.

Fig 1: Example of Trie

We assume that n out m leaves in the trie have objects in them and other leaves are empty. We use the Arbitrary CRCW PRAM model with n(logm+1) processors with logm+1 processor allocated to each nonempty leaf of the trie. Fig. 2. gives 4 integers and assumed that they are placed at the leaves of a trie with 8 leaves. 4 processors are allocated to each of them. This is shown in Fig. 3.

Integers	5	3	1	7
Processors assigned	0,1,2,3	4,5,6,7	8,9,10,11	12,13,14,15

Fig. 2

Fig 2: Assigning random Processors to integers

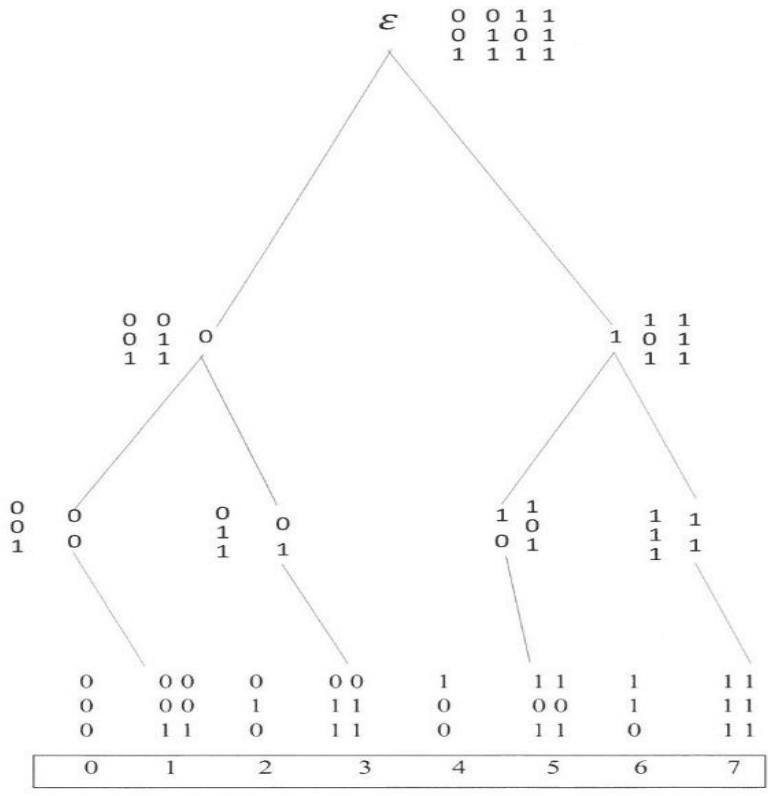

Fig. 3.

Fig 3: Processor allocation shown in trie

In Fig. 3. Integer 001 uses a processor at 001, a processor at 00, a processor at 0 and a processor at the root. Integer 011 uses a processor at 011, a processor at 01, a processor at 0 and a processor at the root. Integer 101 uses a processor at 101, a processor at 10, a processor at 1 and a processor at the root. Integer 111 uses a processor at 111, a processor at 11, a processor at 1 and a processor at the root. A tree T on the trie for these nonempty leaves can be envisioned as shown in Fig. 4. Each internal node of the tree has two children (possibly except the root). Our task is to let every node in the tree link to its parent in the tree provided that in the input the tree has not been built yet.

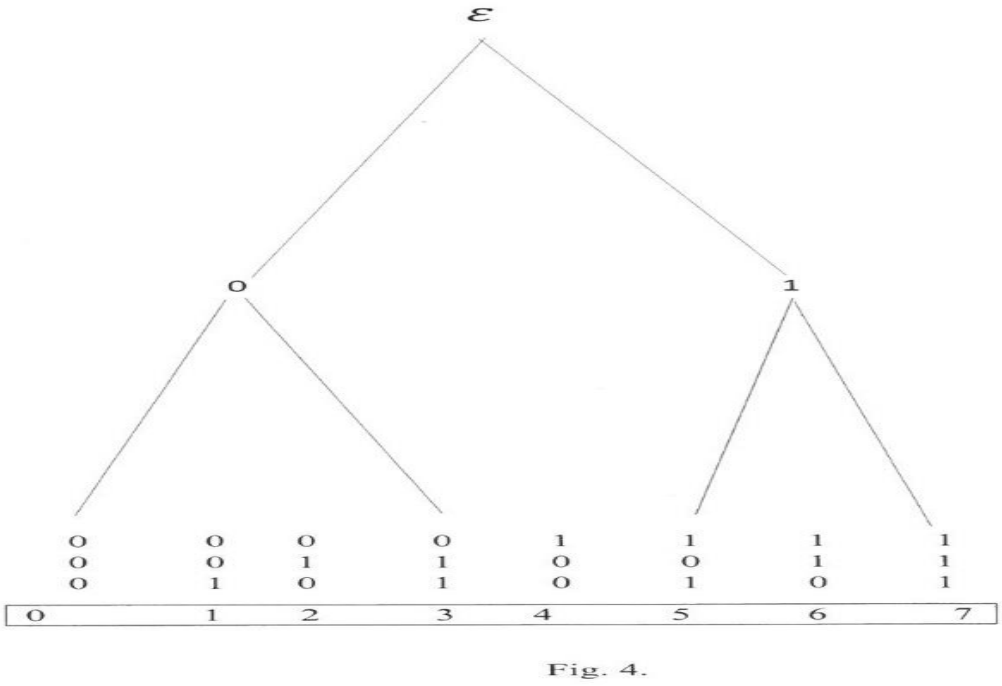

Fig. 4.

Fig 4: Two children shown on trie

Using Arbitrary CRCW PRAM, each integer at a leaf of the trie will use one of its processors allocated to write at an ancestor in the trie of this leaf node. Fig. 5 shows a situation where processors win the write at the node of the trie.

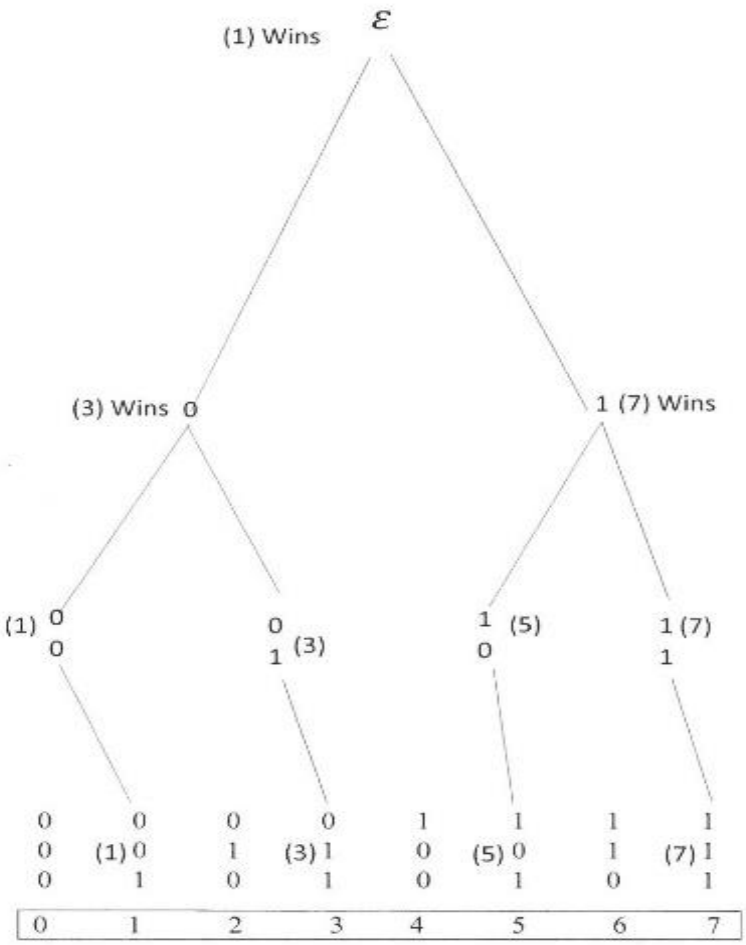

Fig. 5.

Fig5: Processors wins the write

Let $\varepsilon \rightarrow b_1 \rightarrow b_1b_2 \rightarrow b_1b_2b_3 \rightarrow \ldots \rightarrow b_1b_2b_3..b_{logm}$ be a path from the root of the trie to a leaf, where b_i's are binary bits. Because we use Arbitrary CRCW PRAM and therefore it is possible that processors for an integer wins at nonconsecutive nodes on b_1, b_1b_2, $b_1b_2b_3$,…, $b_1b_2b_3..b_{logm}$. For each integer at a leaf of the trie we will find the node at the highest level of the trie that it wins. For integer i let h(i) be the node in the trie at the highest level that processor for i wins. For example, in Fig. 5., h(3)=0, h(1)=ε, h(5)=10 and h(7)=1. Next integer i will do exclusive-or between i and each integer j with h(j) being an ancestor of h(i) in the trie. This exclusive-or will give the branch level between i and j.

This information is obtained from the most significant bit that is 1 of the exclusive-or. For example, h(3) has 1 at its ancestors. 001 xor 011 = 010. This indicates that 3 and 1 branch at at level 2. h(5) has 7 and 1 at its ancestors. 101 xor 111=010 so 5 and 7 branch at level 2. 101 xor 001 = 100 so 5 and 1 branch at level 3.

Next each integer finds, among these exclusive or operations with its ancestors, the lowest branch level. So, 3 finds its lowest branch level at level 2, 5 finds its lowest branch level at level 2 (minimum of level 2 and level 3) and 7 finds its lowest branch level at level 3. Because 1 wins the write at the root and therefore it needs not to find its lowest branch level as it has no ancestors. This situation is shown in Fig. 6.

Now, the root will use the four processors of 7 (because 7 finds its lowest branch branches out of the root ε) to find the node of the tree at the highest level along the path of 7 (i.e. ε, 1, 11, 111) to be 1 and therefore link 1 to ε. Integer 1 at the root will also use the four processors of 7 to find the node of the tree at the highest level along the path of 1 (i.e. ε, 0, 00, 001) because 1 is at the root.

Here, 1 finds that 0 is the node at the highest level and therefore it links 0 to ε. Because integer 3 founds its lowest branch level at level 2 and it branches out of node 0

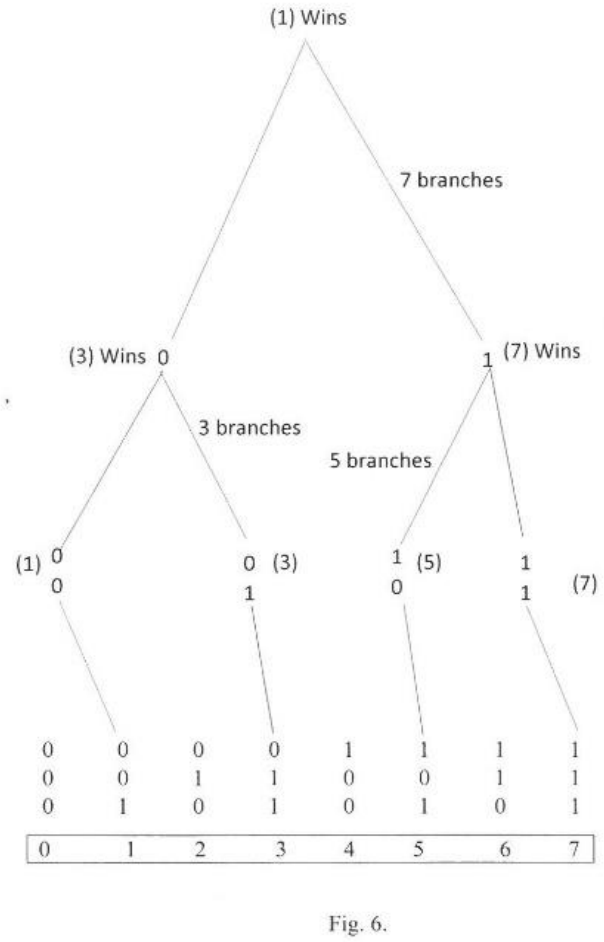

Fig. 6.

Fig 6: Finding the lowest branch

Thus node 0 will use the four processors of 3 to find the highest level node along the path of 0, 01, 011 (this path is generated by 3) to be 011 (note that all leaves of the tree are nodes in the tree and all internal nodes except the root have two children).

Thus 011 is linked to 0. Node 0 will also use the four processors of 3 to find the highest-level node along the path of 0. 00, 001 (this path is generated by 1 as 1 wins the node 1 in the trie) to be 001. Thus 001 is linked to 0. Because the lowest branch of 5 branches out of node 1 in the trie and therefore node 1 in the trie will use the four processors of 5 to find the highest level node along the path of 1, 10, 101 (this path is generated by 5) to be 101. Thus 101 is linked to 1. Also, node 1 in the trie will use the four processors of 5 to find the highest-level node along the path of 1, 11, 111 (this path is generated by 7 because 7 won at node 1) to be 111. Therefore 111 is linked to 1. All these operations take constant time by using the constant time algorithm for finding integer maximum among n integers in {0, 1, …, n} using n processors.

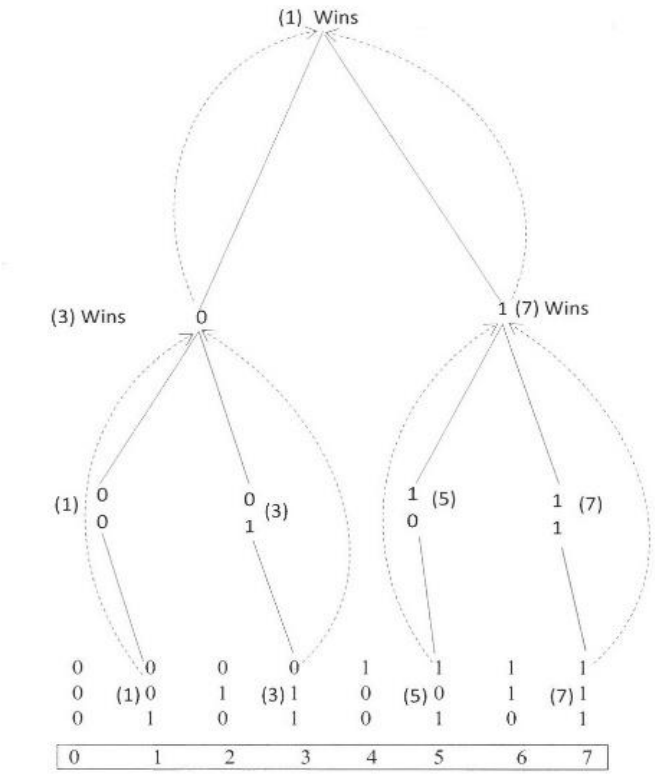

Fig. 7.

Fig 7: Chaining the nodes

CHAPTER 6

CONCLUSION

The chaining algorithm presented in this paper has some desirable characters. Both the models, the Arbitrary CRCW PRAM model and the Priority CRCW PRAM model take the same amount of time to sort the integers in the linked list. Converting Arbitrary model into the Priority model is, design being the weaker model, it is better than Priority CRCW model. On the positive side, the algorithm is simple and easy to program. It has no hidden factors and is fast in practical terms.

To understand the chaining problem in the second paper, we studied parallel algorithms. Several researchers have shown their research work and the most recent work done in this algorithm is simple and more efficient. This paper has shown that,n integers can be chained on the trie in constant time using n(logm+1) processors on the Arbitrary CRCW PRAM models. This problem was difficult to solve. The facts mentioned in paper indicates that it is feasible in a practical term.

References

[1] P. Beame and J . Histad. Optimal bounds for decision problems on the CRCW PRAM. *Journal of the ACM,* 36:643-670, 1989.

[2] F.E. Fich, P. L. Ragde, and A. Wigderson. Simulations among concurrent-write PRAMs, *Algorithimica*, 3, 43-51(1988).

[3] T. Goldberg, U. Zwick. Optimal deterministic approximate parallel prefix sums and their applications. *Proceedings 1995 Israel Symposium on the Theory of Computing and Systems*, 220-228 (1995).

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

[5] T. Hagerup and R. Raman. Fast deterministic approximate and exact parallel sorting. In *Proceedings of the 5th Annual ACM Symposium on Parallel algorithms and architectures, Velen, Germany*, 346-355, 1993.

[6] Y. Han, H. Koganti. Searching in a Sorted Linked List. *In Proceedings of the 17th Int. Conf. on Information Technology (ICIT'2018).*

[7]. Y. Han, X. Shen. Parallel integer sorting is more efficient than parallel comparison sorting on exclusive write PRAMs. *SIAM J. Comput.* 31, 6, 1852-1878(2002).

[8] 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)

[9] P. Ragde. The parallel simplicity of compaction and chaining. *Journal of Algorithms*, 14(3), 371-380(1993).

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

[11] Y. Han, H. Koganti. Searching in a Sorted Linked List. In Proceedings of the 17th Int. Conf. on Information Technology (ICIT'2018).

[12] Y. Han, H. Koganti, N. Goyal. Sort Integers into a Linked List. Accepted by 2019 Int. Conf. on Foundations of Computer Science, Computer and Information Science Journal, vol.13,No1 (2019, 2020).

[13] 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)

[14] P. Ragde. The parallel simplicity of compaction and chaining. *Journal of Algorithms*, 14(3), 371-380(1993).

VITA

Nikita Goyal was born on March 18 in Punjab, India. She completed her bachelor's degree in Computer Science from Chitkara University in Punjab, India. After completing her under-graduation, to enhance her career further she started her master's in computer science at the University of Missouri-Kansas City (UMKC) in Fall 18, Majoring in Computer Science emphasis. Upon completion of her requirements for the master's Program, Nikita Goyal Plans to do a job as a Software Developer Data Platform.