Journal of Theoretical and Applied Information Technology
30th April 2022. Vol.100. No 8

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

IMPROVEMENT OF PARALLEL AVERAGE PARTITIONING AND SORTING ALGORITHM WITH DECREASING SINGLE POINT OF FAILURE AND BOTTLENECK

Ahmed Hammad Helal[1], Masoud E Shaheen[2]

[1] Computer Science department, Faculty of computer science and information system, Fayoum University, Egypt.

[2] Computer Science department, Faculty of computer science and information system, Fayoum University, Egypt.

E-mail: [1]ah1348@fayoum.edu.eg

ABSTRACT

Sorting algorithms are algorithms which put elements of a list in a certain order. The first sorting algorithms were sequential and there are many contributions to make these algorithms parallel, these contributions have been attracted many researchers may be due to the complexity of solving it efficiently. Parallel sorting algorithms have conquered many problems like decreasing processing time and resource utilization. This paper presents some enhancements on a parallel partitioning and sorting algorithm to overcome single point of failure and bottleneck problems.

Keywords: *Bottleneck, Single Point of Failure, Resource Utilization, Distributed Shared Memory System, Many to Many Communication.*

1. INTRODUCTION

Parallel algorithms have many advantages such as overcoming the long time which consumed using sequential algorithms like quick sort, merge sort, and bubble sort [1], to achieve resource utilization.

Whereas sorting plays a vital role between computer operations and facilitates some other computer operations like searching and reading or writing from particular position [2-3]. So many attempts have been done to turn the sequential sorting algorithms to parallel algorithms to achieve the advantages of parallel algorithms like Performing operations efficiently in a short time, but the parallel algorithms face many obstacles and challenges like resource utilization, bottleneck, fault tolerance and single point of failure.

Some of these attempts which tried to perform quicksort algorithm in parallel are:

- "Hyper Quicksort algorithm".

Hyper quicksort is a parallel algorithm which implements quicksort algorithm on a hypercube [4].

- "Fast Parallel Sorting Algorithm Using Subsets and Quick Sort (FPSAUSQS)".

The main idea of " FPSAUSQS" is division of large set of variables to subsets and sort parallelly every subset [5-6].

- "Hams Algorithm"

Hams algorithm is a parallel partitioning and sorting algorithm aims to divide data equally [7].

The difference between these algorithms was in partitioning, it's known that the best number which can be used in partitioning is the median [8], but the main obstacle is that the median can't be known unless the numbers are sorted, however these algorithms found alternative methods which can be used in partitioning, for example "Hyper Quicksort" uses the median of a sorted sub list of numbers, "FPSAUSQS" uses the maximum number in the list of numbers and "Hams algorithm" uses the using arithmetic mean of the list of numbers.

2. RELATED WORK

2.1 Parallel Sorting of Arrays with OpenMP and the Quick-Sort Algorithm

In this algorithm the pivot element is chosen randomly from the unsorted array and broadcasted to all working processors. The next step is dividing each array into two subarrays less and greater than the pivot element. the following steps are exchanging then sorting these subarrays recursively so that every processor has a sorted subarray greater than its pervious processor and less than next one [9].

2.2 Dual Parallel Partition Sorting Algorithm

Dual Parallel Partition Sorting Algorithm uses two given indices the first one for left partition and the second one for right partition and has a given pivot element the values in the left and right are compared with the pivot element and swapped until the right values be less than the pivot and the left values be great than the pivot. Then (STLSort or qsort) function is used to sort the values in parallel [10].

2.3 Hams, Parallel Average Partitioning and Sorting Algorithm

" Hams, parallel average partitioning and sorting algorithm" is an effective algorithm used in division a given list of values in parallel into almost equal partitions using the arithmetic mean of this list, then it uses a quick sort algorithm to sort partitioned sub lists in parallel. As shown in figure 1 the algorithm uses message passing interface to communicate between processes and uses master /slave architecture, so they perform single read and single write [7]. In this algorithm there is a bottleneck in reading the data from input file using the master process only and this also would lead to single point of failure if the master process was down for any reason and the same problem is found in calculating the arithmetic mean of related processes [11].

3. RESEARCH METHOD

In this paper an enhanced algorithm of Hams algorithm called *i*Hams will be presented, some modifications are implemented to enhance the performance of Hams algorithm.

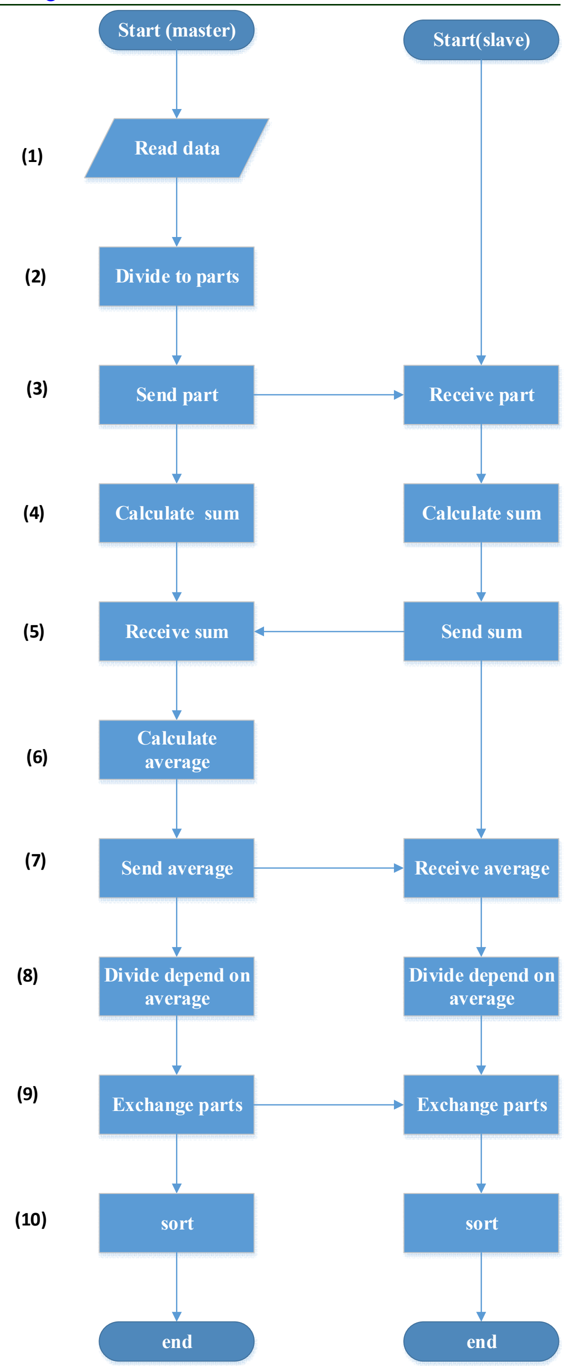

Figure 1: Hams Algorithm Steps

3.1 The Proposed Algorithm

The proposed algorithm *i*Hams aims to conquer some problems like single point of failure and bottleneck and achieve resource utilization and load balance that will may be led to reduce the execution time and fault tolerance [12-13]. As shown in figure 2 many to many communication

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

and distributed shared memory (DSM) mechanism is used in this algorithm, all process will communicate between each other and data source [14-15].

First modification is to replace single reading of the input data file using master process by parallel reading using all available processes by partitioning the data into chunks and read it using index, data reading index is calculated depending on the following equation.

$$I = ((CD / CP) * N) - (CD / CP) \qquad (1)$$

where I is the data reading index, CD count of upcoming data, CP count of working processes, and N is process index.

Second modification is to replace calculating the arithmetic mean in one process by calculating it in all related processes by broadcast the sums of chunks in all related processes.

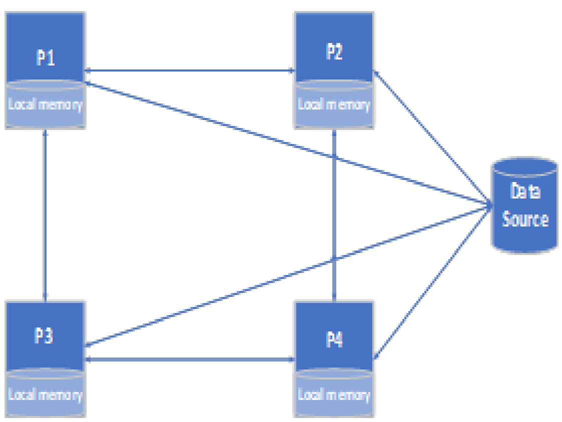

Figure 2: Communication Between Processes

3.2 Algorithm Steps

As shown in figure 3 the algorithm steps are as following:

1. Setting number to every process.
2. Calculating data reading index (I) depending on its number using equation (1).
3. Every process reads its data chunk
4. Every process calculates a sum of its data chunk
5. Every process broadcasts the sum of its data chunk
6. Every process calculates the arithmetic mean
7. Every process divides its chunk into two partitions, one of them is less or equal to the arithmetic mean and the second is greater than the arithmetic mean.
8. Swapping the partitions between each two-coupled processes.
9. Repeating the steps from 5 to 8 for lower half and upper half process log CP times
where CP is count of working processes.
10. Finally, every process sorts its data chunk using quick sort.

3.2 Case Study

As shown in figure 4, suppose there are four processes which can work in parallel together and an input data file that has twenty numbers.

1. Every process reads the data in an array and calculates the index of its data chunk.
2. Every process calculates a sum of its chunk.
3. Every process sends and receives the sum.
4. Every calculates the general arithmetic mean locally.
5. Every process splits its data lower and greater than the arithmetic mean.
6. Every process swaps splitted parts.
7. Every process calculates the sum of its chunk.
8. Every process sends and receives the sum.
9. Every calculates the secondary arithmetic mean locally.
10. Every process splits its data lower and greater than the arithmetic mean.
11. Every process swaps splitted parts.
12. Finally, every process sorts its chunk using quick sort algorithm.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

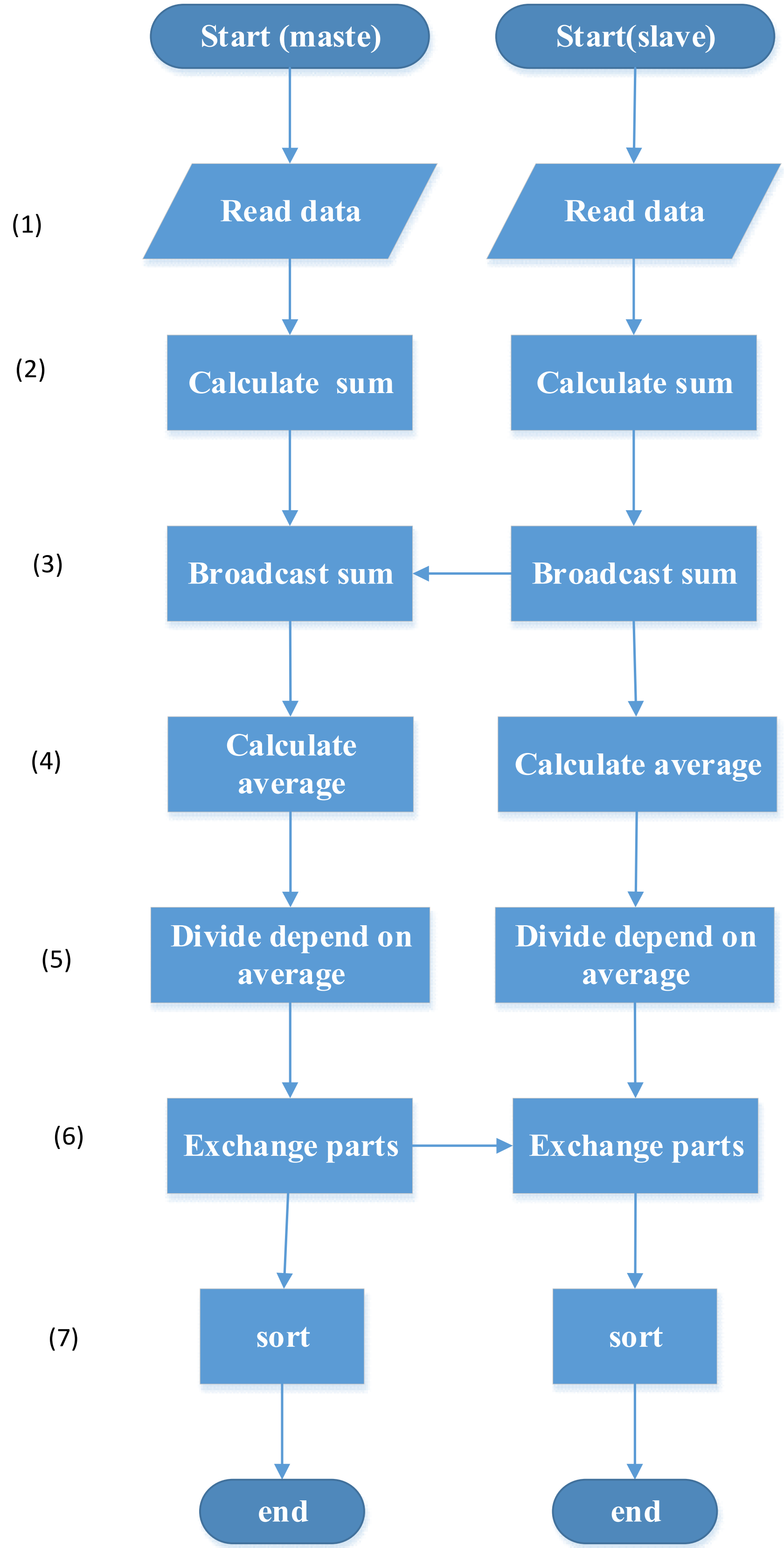

Figure 3: iHams Algorithm Steps

74	56	63	32	42	81	93	5	22	12	98	34	19	24	0	88	52	17	56	79

1

P1	74	56	63	32	42
P2	81	93	5	22	12
P3	98	34	19	24	0
P4	88	52	17	56	79

2

P1	74	56	63	32	42	267
P2	81	93	5	22	12	213
P3	98	34	19	24	0	175
P4	88	52	17	56	79	292

3

P1	74	56	63	32	42	(276,5
P2	81	93	5	22	12	(213,5
P3	98	34	19	24	0	(175,5
P4	88	52	17	56	79	(292.5

4

P1	74	56	63	32	42	47
P2	81	93	5	22	12	47
P3	98	34	19	24	0	47
P4	88	52	17	56	79	47

5

P1	32	42	74	56	63	47
P2	5	22	12	81	93	47
P3	34	19	24	0	98	47
P4	17	88	52	56	79	47

6

P1	32	42	34	19	24	0
P2	5	22	12	17		
P3	98	74	56	63		
P4	88	52	56	79	81	93

7

P1	32	42	34	19	24	0	151
P2	5	22	12	17			56
P3	98	74	56	63			291
P4	88	52	56	79	81	93	449

8

P1	32	42	34	19	24	0	(151,6
P2	5	22	12	17			(56,4)
P3	98	74	56	63			(291,4
P4	88	52	56	79	81	93	(449,6)

9

P1	32	42	34	19	24	0	20
P2	5	22	12	17			20
P3	98	74	56	63			74
P4	88	52	56	79	81	93	74

10

P1	19	0	32	42	34	24	20
P2	5	12	17	22			20
P3	74	56	63	98			74
P4	52	56	88	79	81	93	74

11

P1	19	0	5	12	17
P2	22	32	42	34	24
P3	74	56	63	52	56
P4	98	88	79	81	93

12

P1	0	5	12	17	19
P2	22	24	32	34	42
P3	52	56	56	63	74
P4	79	81	88	93	98

Figure 4: Case Study of the Proposed Algorithm

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

4. RESULTS AND DISCUSSION

IHams and Hams algorithms are made by C++ programming language with MPI [16], this experiment has been done on a virtual machine with a four cores processor with 2.1 GHz and five-gigabyte RAM.

The experiment was made using random numbers generated by the uniform_int_distribution class. The range of generated numbers from 0 to 100000 and the counts of numbers are 1000,10000 and 100000.

Moreover, the experiment was applied to sort Total Compensation column of a real data of the benefits and salary paid to The San Francisco employees with the file version uploaded on 09 May 2019, which has 831308 rows [17].

The experiment was made using 4 processes, table 1 and figure 4 show the affected processes in every step if process number one (P1) is down. It is noticed that affected processes decrease in *i*Hams algorithm than Hams algorithm and this leads to decrease the dependency on master process and increasing the dependency on other working processes in *i*Hams algorithm than Hams algorithm, so this will lead by its role to overcome resource utilization, bottleneck, single point of failure and fault tolerance issues.

The execution time of the four experiments in microsecond unit is as shown in table 2 and figure 4. It is noticed that the execution time of *i*Hams algorithm decreases compared with the execution time of Hams algorithm, The difference between them becomes clear whenever the data set becomes larger.

Table 1: Count of Affected Processes

Step ID	Step	Normal affected	Hams	*i*Hams
1	read data	1	4	1
2	divide the data	1	4	0
3	send the data	1	4	0
4	calculate sum	1	1	1
5	send sum	1	0	1
6	receive sum	1	4	1
7	calculate average	1	4	1
8	send average	1	4	0
9	partitioning the data	1	1	1
10	exchange partitions	1	2	2
11	calculate sum	1	1	1
12	send sum	1	0	1
13	receive sum	1	2	1
14	calculate average	1	2	1
15	send the average	1	2	1
16	partitioning the data	1	1	1
17	exchange partitions	1	2	2
18	sorting	1	1	1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 2: Execution Time in Microseconds

Count of numbers	Hams	*i*Hams
1000	6990	5460
10000	62403	48609
100000	577210	449062
831308	4041399	3140158

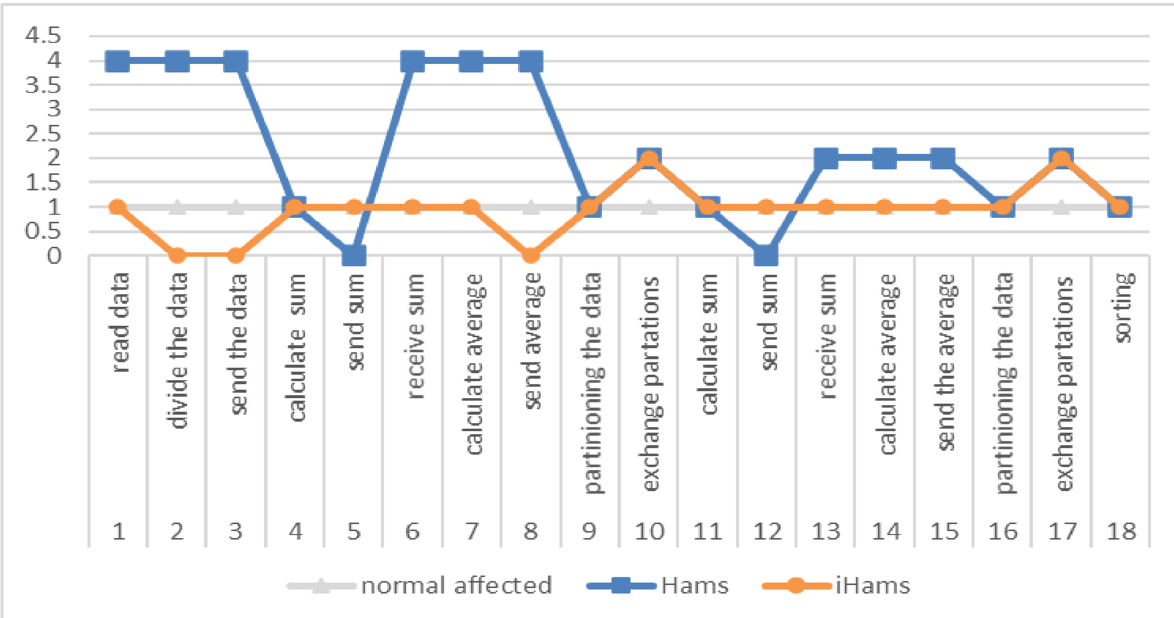

Figure 5: Count of affected processes

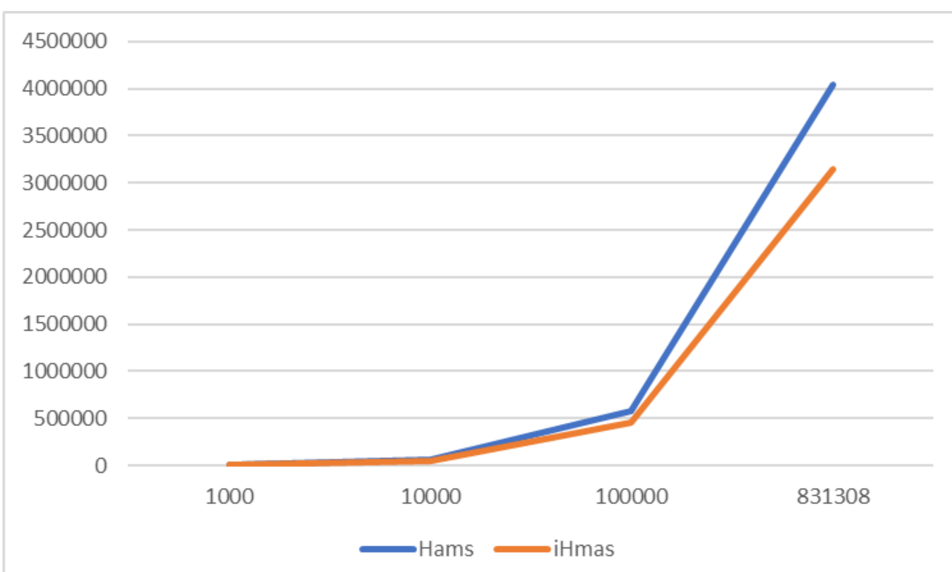

Figure 6: Execution Time in microseconds

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

5. CONCLUSION AND FUTURE WORK

Resource utilization, bottleneck, single point of failure and fault tolerance are obstacles in implementation the parallel algorithms which uses master and slaves' approach. *i*Hmas is the enhanced algorithm of Hams algorithm, the two algorithms were implemented and compared. The experiment proved that iHams decreases the dependency on

master process and increases the dependency on other working processes, which leads to overcome resource utilization, bottleneck, single point of failure and fault tolerance issues, and this also leads to less execution time than Hams algorithm.

In the future work, we will apply iHams algorithm on a larger data set and increase the count of working processes with improving the communication between processes and performance of the algorithm.

REFERENCES:

[1] Gary Pollice, Stanley Selkow, George T. Heineman, "Algorithms in a Nutshell", O'Reilly Media, Inc,2008.

[2] Knuth D., The art of computer programming, V3. "Sorting and Searching, Addison Wesley Publishing Company", 1973.

[3] R. Kaushal, Why Sorting is So Important in Data Structures, International Journal for Scientific Research & Development| Vol. 5, Issue 07, 2017.

[4] M. J. Quinn, Parallel Programming in C with MPI andOpenMP, Tata McGraw Hill Publications, 2003, p. 338.

[5] Bosakova-Ardenska A., Vasilev N., Kostadinova-Georgieva L., Fast Parallel Sorting Algorithm using Subsets and Quick sort, Balkan Journal of Electrical & Computer Engineering, 2015, Vol.3, No.1, ISSN: 2147-284X, pp.27-29 **.**

[6] A. Bosakova-Ardenska , A.Miroslav, IMPLEMENTATION OF FAST PARALLEL SORTING ALGORITHM WITH C AND MPI, INTERNATIONAL SCIENTIFIC CONFERENCE 18 – 19 November 2016, GABROVO.

[7] A.Hammad , M.E-Shaheen HAMS, PARALLEL AVERAGE PARTITIONING AND SORTING ALGORITHM , INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH VOLUME 9, ISSUE 03, MARCH 2020.

[8] J.Medhi , Statistical Methods: An Introductory Text , New Age International. pp. 53–69,1992.

[9] G.Petkovski, D.Tosev, Parallel Sorting of Arrays with OpenMP and the Quick-Sort Algorithm , • Conference: The 12th International Conference for Informatics and Information Technology,2015

[10] A.Rattanatranurak, Dual Parallel Partition Sorting Algorithm, Conference: Proceedings of 2018 International Conference on Information Technology and Science,2019.

[11] M.Zheludkov, S.Bhuiyan ,"The Apache Ignite Book" - Page 101,2019.

[12] Designing Large-scale LANs – Page 31, K. Dooley, O'Reilly, 2002 .

[13] https:/ /www.investopedia.com/terms/b/bottleneck.asp By ADAM BARONE Updated May 8, 2019.

[14] H.Geng , N.Jamali , Supporting Many-to-Many Communication , Conference: AGERE!,2013.

[15] L. Czaja , Introduction to Distributed Computer Systems , 2018 , p. 187 - 126.

[16] M.snir,s.olto,S.Huss-lederman,D.Walker and J.Dongarra ,MPI the complete reference ,MIT press ,Cambridge ,MA,USA,1996.

[17] data.gov, ―Employee Compensation‖, Available at: https://catalog.data.gov/dataset/employee-compensation53987?fbclid=IwAR3wabi2vXamKLc7BHjFyz5Afw39yHp3 4fqe67BWHmEwiVTt6-xkhSynvrc,2019 .