ISSN:0975-9646

An Experimental Study of Performance Comparison for Various Parallel Sorting Algorithms

Jagdeep Singh*, Alka Singh**, Garima Singh***

*M.tech. (Computer Science & Engineering), KNIT, Sultanpur, India

**Astt.Professor – Deptt. Of Computer Science & Engineering, KNIT, Sultanpur, India

***B.tech. (Computer Science & Engineering), BBDU, Lucknow, India

***Abstract*- We all know that the most important procedure in managing data is its sorting. To perform sorting, one can choose different sorting algorithmic methods. But, all the various sorting algorithms do not result the same speed, execution time and efficiency at a given set of inputs. Hence, it is necessary to know which algorithm can give better result for a given platform and pre-defined data sets. This paper presents reader an experimental study of performance comparison for various parallel sorting algorithms.**

***Keywords*- Bitonic sort, odd-even merge sort, parallel merge sort, parallel rank sort, complexity.**

I. INTRODUCTION

The process of rearranging data in a particular order that may be in a increasing or decreasing manner is termed as sorting. The data set may be alphabetical or numerical in nature. Every existing matter in this world has its few advantages as well as disadvantages. Similarly different sorting algorithms have their advantages and disadvantages. Sorting is used for performing search operation in an easier way saving time. But, sorting algorithms can't be used with same efficiency on all data-sets. Every method depending upon the data-set performs differently in different aspects with respect to processing time, speed and efficiency; few perform poorly whereas few give results in a short time. To understand the above phenomena in a better descriptive way, we in this paper will carry out an experimental study for an assumed data-set of various parallel sorting algorithms on the basis of performance and complexity.

II. PARALLEL SORTING METHODS

Parallel sorting methods are classified on the following basis:

1. On the basis of memory usage:

Sorting method	Memory occupied
Odd – even sort	1
Parallel merge sort	n
Bitonic sort	$n(\log(logn))$
Parallel rank sort	$n(\log(logn))$
Parallel quick sort	$logn$

2. On the computational complexity basis:

Sorting method	Best case	Average case	Worst case
Odd – even sort	n	n^2	n^2
Parallel merge sort	$nlogn$	$nlogn$	$nlogn$
Bitonic sort	$(\log(logn))$	$(\log(logn))$	$(\log(logn))$
Parallel rank sort	$(\log(logn))$	$(\log(logn))$	$(\log(logn))$
Parallel quick sort	$nlogn$	$nlogn$	n^2

3. On the basis of recursion: Algorithms may be recursive or non-recursive in nature. Few may be both viz. parallel merge sort.

In this paper, we will discuss in detail about Parallel merge sort, Parallel rank sort, Odd – even sort algorithms.

1. Parallel Merge Sort

(Splits data set in half → sorts each half recursively → merges them back together to a sorted list) is the main motto of merge sorting. The merge sort algorithm can be parallelized by distributing (n/p) processors to each processor. Each processor sequentially sorts the sub list and then return to final sorted list.

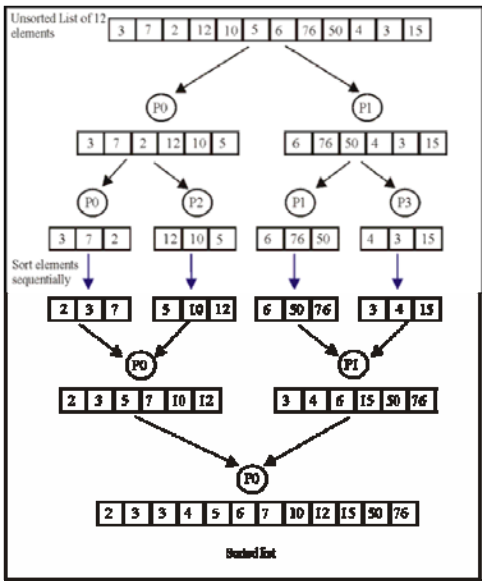

Fig 1: Parallel merge sort algorithm e.g.

Parallel merge sort time complexity:

- In sequential environment → O $(nlogn)$
- In parallel environment → O $(\frac{n}{p} log \frac{n}{p})$

2. Odd – even sort:

(Splits datasets in n/p sub lists → sequentially sort sub lists → operates data sets according to even phase or odd phase → merges them back together to a sorted list). In even phase, even numbered processor communicates with next odd numbered processor i.e. P_i communicates to P_{i+1}. In this communication 2 sub-lists for each communication are merged together. In odd phase, odd numbered processor communicates with even numbered processor i.e. P_i communicates to P_{i-1}.

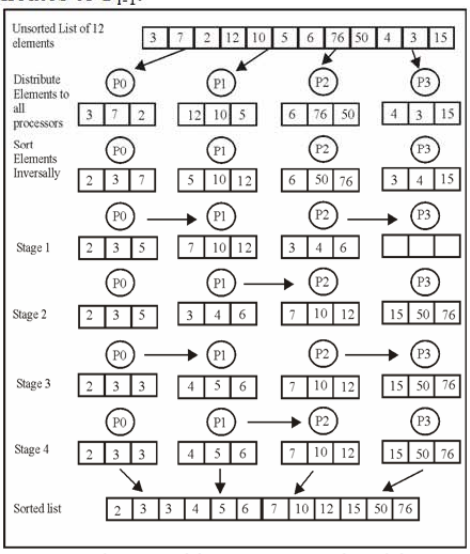

Fig 2: Odd - even sort algorithm e.g.

Odd - even sort time complexity:

- In parallel environment → O (n^2)

3. Parallel rank sort:

(Splits the data list to all the processor → each processor compute the rank → construct the sorted list according to rank). Data list is distributed among all the processors and each processor contains n/p elements.

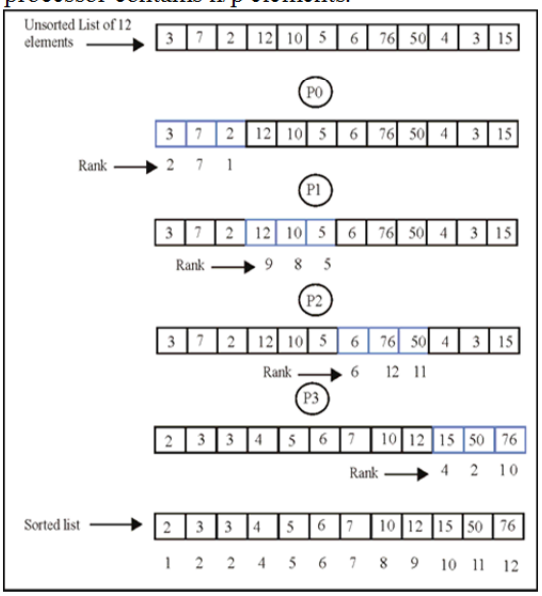

Fig 3: Parallel rank sort algorithm e.g.

Parallel rank sort time complexity:

- In sequential environment → O (n^2)
- In parallel environment → O (n^2)

Note: n is the list size whereas p is the number of processors.

III. Testings And Methodology

How much an algorithm's efficiency and performance can be improved by Parallelism? To answer this question, we have developed and executed three parallel sorting algorithms. These algorithms are parallel merge sort, parallel rank sort, and odd-even sort respectively.

The MPI library has been adopted to establish the communication between processors. We have compared three different parallel sorting algorithms for 10,000 integers (data set) on 2, 4, 6, 8, 10 and 12 work stations respectively. Each algorithm has supported parallelization. Finally, results in the form of combined graphs for three different parameters viz. execution time, speed up and efficiency respectively have been drawn below.

Execution time is the time required to execute an algorithm whether in sequential environment or parallel environment; Speed up is the ratio of the total time taken in the execution of an algorithm in sequential environment to that in a parallel environment, $s(n,p) = \frac{T(n,1)}{T(n,p)}$.

IV. Results

The combined graph for execution time showing time taken at every set of work stations is drawn below:

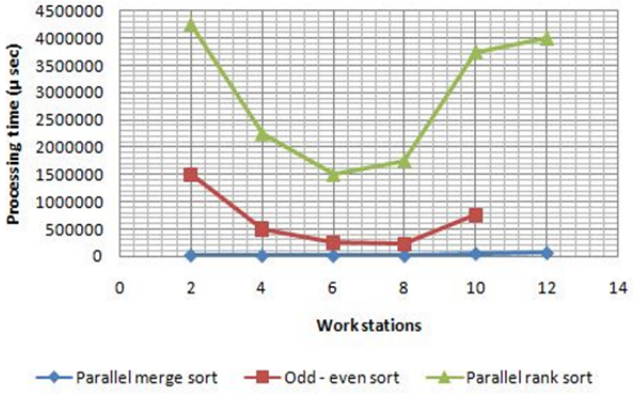

Fig 4: Combined graph for total execution time

The combined graph for speed up at every set of work stations is drawn below:

Fig 5: Combined graph for speed up

The combined graph for efficiency at every set of work stations is drawn below:

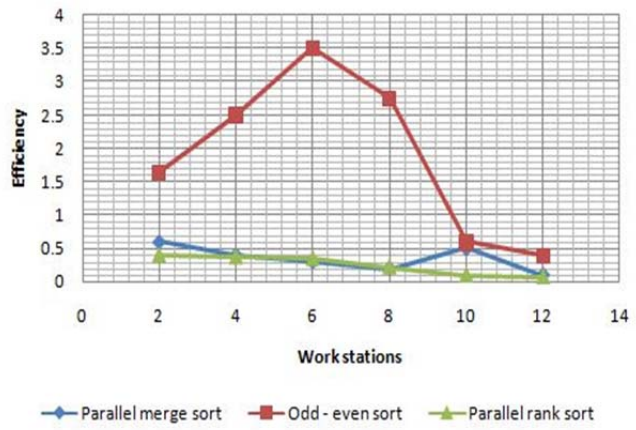

Fig 6: Combined graph for efficiency

V. CONCLUSIONS

1. If we talk about the total execution time, the least amount of time was taken by parallel merge sort whereas parallel rank sort took maximum time for the complete execution. This implies that parallel merge sort algorithm is the quickest of all for the given data set whereas the parallel rank sort algorithm is the slowest.
2. From (speed up) point of view, odd – even sort algorithm stands as the best algorithm followed by parallel merge and then parallel rank sort.
3. Talking about efficiency, again odd – even sort algorithm leads of all, worst stands the parallel rank sort algorithm.
4. Finally it can be concluded that, from the chosen three algorithms, parallel rank sort performs worst in each of the cases.
5. It is upon the user to go for parallel merge sort or odd – even sort depending upon his needs. If one wishes to have shortest execution time i.e. fastest processing algorithm, one should choose parallel merge sort whereas if someone believes in efficiency, one may choose odd – even sort algorithm.

REFERENCES:

[1] Kalim Qureshi and Haroon Rashid,"A Practical Performance Comparison of Parallel Matrix Multiplication Algorithms on Network of Workstations.", IEE Transaction Japan, Vol. 125, No. 3, 2005.

[2] Kalim Qureshi and Haroon Rashid," A Practical Performance Comparison of Two Parallel Fast Fourier Transform Algorithms on Cluster of PCS", IEE Transaction Japan, Vol. 124, No. 11, 2004.

[3] Kalim Qureshi and Masahiko Hatanaka, "A Practical Approach of Task Partitioning and Scheduling on Heterogeneous Parallel Distributed Image Computing System," Transaction of IEE Japan, Vol. 120-C, No. 1, Jan., 2000, pp. 151-157.

[4] K. Sado, Y. Igarashi, Some Parallel Sorts on a Mesh-Connected Processor Array and Their Time Efficiency, Journal of Parallel and Distributed Computing, 3, pp. 398-410, 1999.

[5] D. Bitton, D. DeWitt, D.K. Hsiao, J. Menon, A Taxonomy of Parallel Sorting, ACM Computing Surveys, 16,3,pp. 287-318, September 1984.

[6] Song, Y.D., Shirasi, B. A Parallel Exchange Sort Algorithm. *South Methodist University, IEEE 1989*.

[7] B.R. lyer, D.M. Dias, System Issues in Parallel Sorting for Database Systems, Proc. Int. Conference on Data Engineering, pp. 246-255, 2003.

[8] F. Meyer auf der Heide, A Wigderson, The Complexity of Parallel Sorting, SIAM Journal of Computing, 16, 1, pp. 100-107, February 1999.