The CSI Journal on
Computer Science and Engineering
Vol. 19, No. 1, 2025
Pages 7-16
doi.org/10.22034/jcse.2025.182105
Regular Paper

Performance analysis of parallel sorting algorithms Using Message Passing Interface (MPI)

Yahya Kord Tamandani[1]
yahya.kord@cs.usb.ac.ir

[1]Department of Computer Science, University of Sistan and Baluchestan, Iran, Zahedan

Abstract

A large number of complex applications which can include database, data mining, as well as image processing need to involve a number of computations, simulations, iterations and accuracy. These types of high-end applications may well be solved making use of parallel processing a situation where the problem is divided into a set of independent sub-problems and solving each one of the sub-problems simultaneously. Message Passing Interface (MPI) is a powerful and standard interface with a library of subroutines and functions to write parallel programs. Sorting plays a very important role in many applications. In this paper some parallel algorithms namely parallel quick sort, parallel merge sort and parallel hybrid Merge-Quicksort are discussed. The algorithms are implemented using MPI with C programming language on multi-core system with Linux platform. Parallel programs are tested with large data sets. The overall performance with respect to computing time of each parallel algorithm is analyzed. The comparison of parallel and corresponding sequential algorithms is also performed.

Keywords: Parallel processing, Message passing interface, Parallel sorting algorithms, Performance analysis, Computing time.

1. Introduction

The previous decade has experienced significant advances in computer architecture as well as microprocessor technology [1], [4]. Clock rates of processors have increased noticeably. At the same instant, processors are currently very effective at executing a number of instructions in the same cycle.

Desktop computers, servers and engineering workstations with several processors which are connected and work together have become normal platforms to be used by various type of applications. Large scale applications such as database and data mining, image processing relies on parallel computers, quite often comprising many processors as they require a huge number of computations, simulations, iterations with high accuracy. Enhanced hardware design and production in addition to an increasing knowledge of how to handle challenges of parallel programming has re-established parallel processing at the leading edge of computer modern technology. Parallel programming is programming of multiple independent computers to work cooperatively or computers with multiple internal processors in order to utilize the full computing power of the computers [5], [8]. It is believed to be very hard and tedious, overcoming and solving complex problems without using high computing power. Basically, in parallel environment a problem is broken down into number of independent sub-problems and then each sub-problem is solved simultaneously. The recent improvements in computer modern advances have introduced parallel processing computers; and so, a range of languages intending for parallel programming have been designed and improved. Recent advances in computer technology have developed parallel processing computers; accordingly, a number of parallel programming languages have been designed and developed. Message Passing Interface (MPI) [9], [11] is basically a specification for an Application Programming Interface which is a powerful and standard interface with a library of subroutines and functions which allows many computers to

communicate with one another. In this paper some parallel algorithms namely parallel quick sort, parallel merge sort and parallel Hybrid Merge-Quicksort are discussed. The algorithms are implemented using MPI with C programming language on multi-core system with Linux platform. Parallel programs are tested with large data sets and their performance with respect to computing time of the parallel algorithms are analyzed.

1.1. Message Passing Interface (MPI)

MPI is basically used for programming parallel computers. It is a protocol for communication which is language-independent. There exist several available implementations of MPI each focuses on different elements of high-performance computation. The Open MPI [12], [14] is one of the implementations which is maintained and developed by a group consisting of educational researchers, and industrial partners. Open MPI provide benefits to software and system dealers, computer researchers as well as application designer and developers. It offers a high-performance, vigorous, environment for parallel execution. Open MPI is implemented as a library, which is available for nearly all computer platforms such as Linux, Windows and OS X. It also provides interfaces for many popular languages like C, C++, FORTRAN and Python. Open MPI contains compiler wrappers that add all of the necessary includes and linking flags for compiling MPI code.

2. Parallel sorting algorithms

Sorting [15] plays a significant role in computer programming. Sort is typically an ordering procedure by a type of keyword for any given records or data. Sorting is essential to enhance the efficiency of number of operations such as record searching, modification, insertion and deletion. In this paper the experiment has been carried out with the help of some popular sorting algorithms which are discussed as follow:

2.1. Parallel Quick Sort

Quick sort [16] also uses the divide and conquer strategy in order to sort an unsorted list of items by repeatedly dividing the list into sub-lists. Algorithm picks an element from the given list to be sorted as a pivot element. Then the elements smaller than pivot will be located to its left side and elements greater will be positioned to its right side. The same process will be recursively carried out on the right and left sub-lists. Same as parallel merge sort there will be a master process which will broadcast the original list size and distribute the unsorted list to all the other processes. Further the quick sort algorithm will be carried out by each process on its own sub-list as shown if Figure 1. The process of parallel quick sort is given below.

1. *A random element is chosen as pivot and broadcasted to all the other processes*
2. *Each process splits its unsorted list by placing elements smaller than pivot in one sub-list and elements greater in another one.*
3. *The "low list" of each process in upper half is sent to a process located in lower half and given a "high list" in return.*
4. *Processes in upper-half will have elements which are greater than pivot and processes in lower half will only have elements which are smaller than pivot.*
5. *Afterward the processes will be divided into couple of sets and algorithm recourses.*
6. *Then each process will have a separate and disjoint unsorted list of values from other processes after log P recursions.*
7. *The least value in process j+1 will be greater than the main value in process j.*
8. *Sequential quicksort is used by each process to sort its own list*

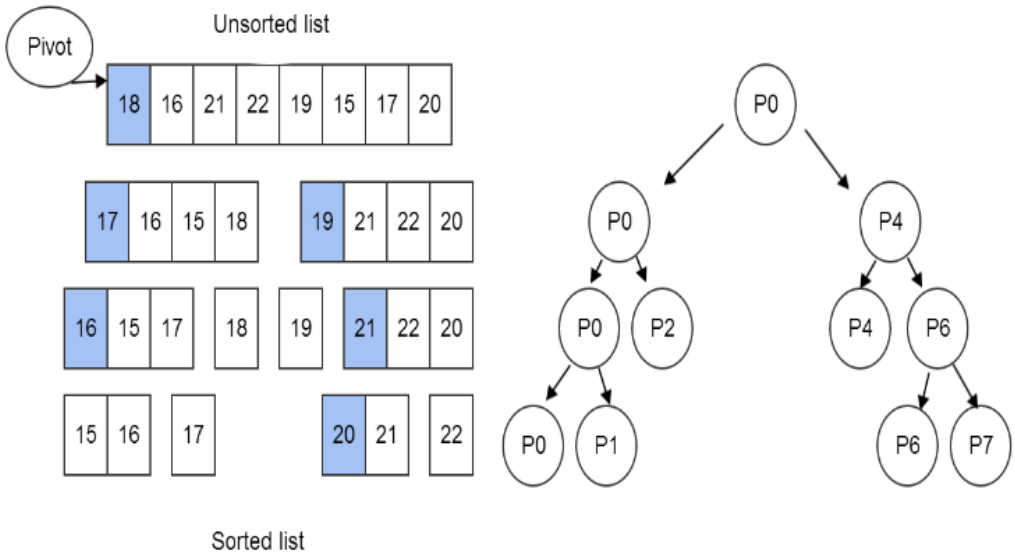

Fig. 1. Allocation of Processes in Parallel Quick Sort

2.2. Parallel Merge Sort

Divide and conquer strategy is being used in merge sort which is as comparison-based sorting algorithm. It is a recurrent procedure in which each time the list to be sorted is divided in two sub-list of same size, the procedure goes on until a single data element is remained in each sub-list [17]. In parallel version of merge sort implemented by MPI one process is selected as master [10] which allots the data to other processes. Each process sorts its own given sub-list. Finally, the master process receives all the sorted sub-list from other processes and merge them to generate the final sorted list having all the original data as shown in Figure2. The process of parallel merge sort algorithm is given below.

```
parallel_mergesort (Array-data, data-size)
Start
Data-left=LeftHalf[Array-data]
Data-right=RightHalf[Array-data]
Send(Data-right)
Data-left = Mergesort(Data-left,i,j)
Receive(Data-right)
Array-data=MergeResult(Data-left, Data-right)
End
Mergesort(Data-left,i,i)
Begin
If(j-i>16)
{
MergeSort(Data-left,i,(i+j)/2)
MergeSort(Data-left,(i+j)/2,j)
}
Else
Sequential-InsertionSort(Data-left,i,j)
End
```

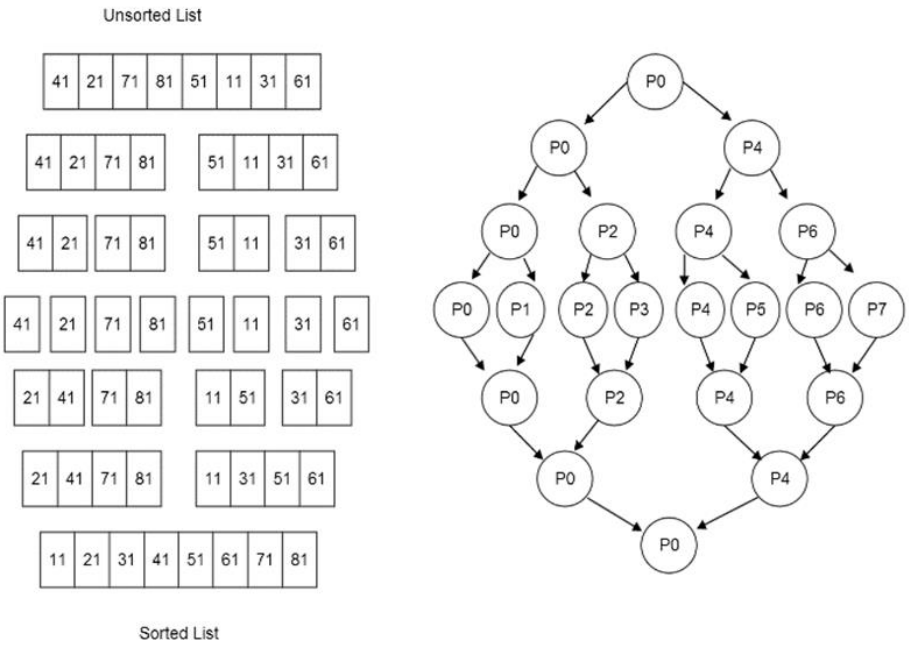

Fig. 2. Allocation of Processes in Parallel Merge Sort

2.3. Parallel hybrid Merge-Quicksort

Parallel hybrid Merge-Quicksort algorithm combines both Merge and Quicksort in order to sort the data. For dividing the data among processes, it utilizes the parallel Merge sort and then using the parallel quicksort algorithm in order to sort data locally at each process. The parallel analysis of hybrid Merge-Quicksort algorithm is identical to merge sort considering that in the best case non-parallel quicksort has the same complexity as the non-parallel Merge sort.

3. Performance analysis

In the Figure 3 and Table 1 the performance of sequential and parallel quick sort algorithms with different numbers of elements to be sorted with respect to computing time is given. The parallel programs were tested with 2 ,4 and 8 processes at a time. It's observed that increasing the number of elements and processes improves the performance and efficiency of parallel quicksort algorithm.

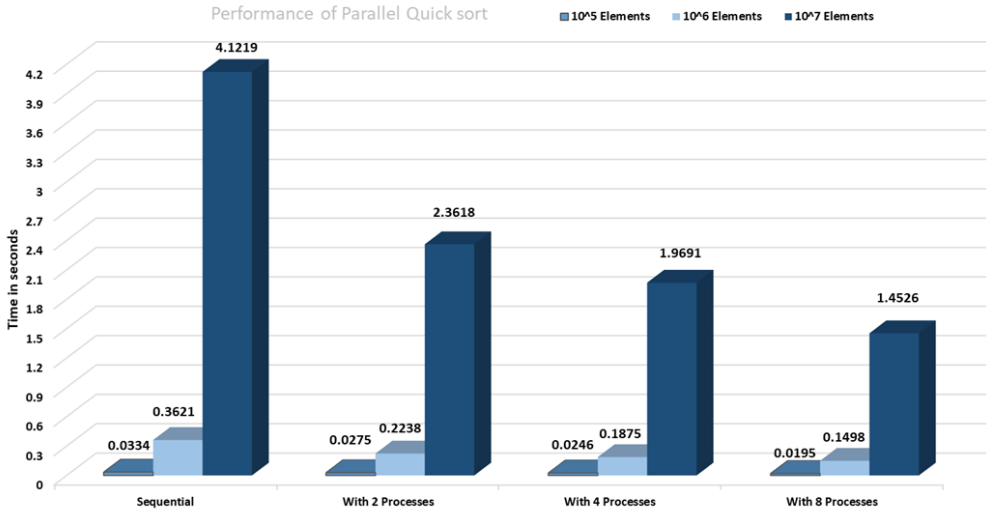

Fig. 3. Performance of Parallel sort and its sequential one

No. of Elements	*Sequential sort*	*Parallel (2 processes)*	*Parallel (4 processes)*	*Parallel (8 processes)*
10^5	0.0334	0.0275	0.0246	0.0195
10^6	0.3621	0.2238	0.1875	0.1498
10^7	4.1219	2.3618	1.9691	1.4526

TABLE 1. Required Computing Time for Quick Sorting

The Figure 4 and Table 2 show the speed up achieved by the parallel quicksort algorithm. It's observed that increasing the number of processes for relatively small data size shows no effective change in the speed up. However, for larger data set it improves the speed up effectively.

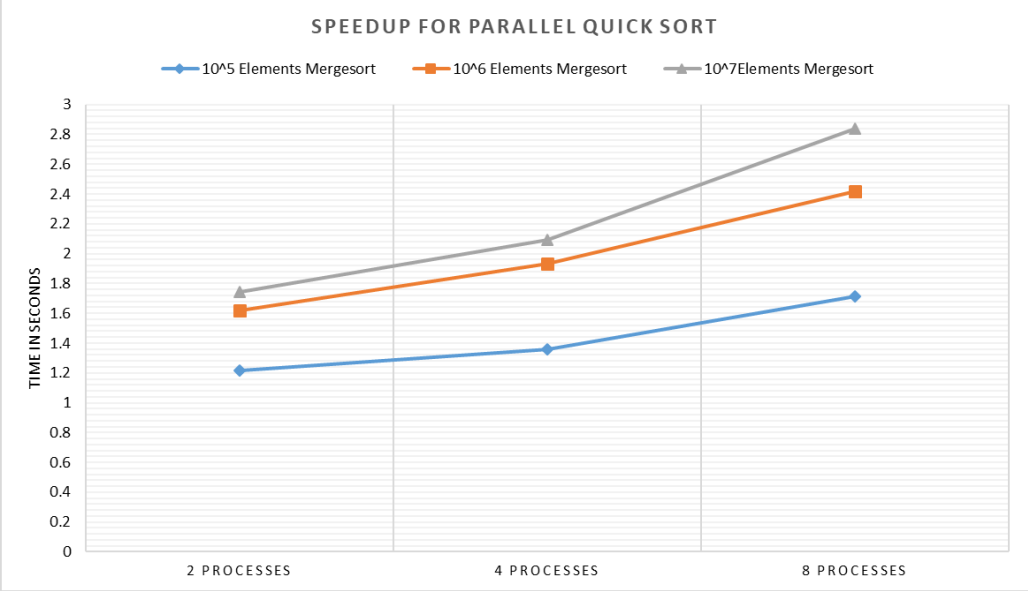

Fig. 4. Speedup for parallel Quick Sort

No. of Elements	*Parallel (2 processes)*	*Parallel (4 processes)*	*Parallel (8 processes)*
10^5	1.2145	1.3577	1.7128
10^6	1.6179	1.9312	2.4172
10^7	1.7452	2.0932	2.8376

TABLE 2. Speedup for parallel Quick Sort

Figure 5 and Table 3 Indicate the performance of sequential and parallel merge sort algorithms with respect to computing time. It is observed that the computation time improvement for 10^7 elements is not effectively improved using 8 processes compared with the one using 4 processes.

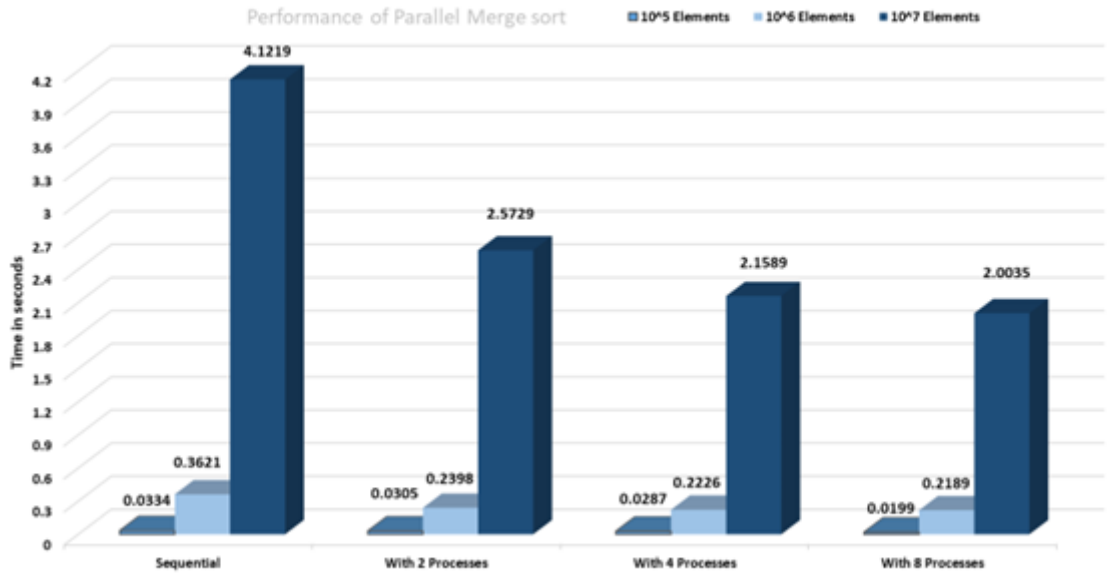

Fig. 5. Performance of Parallel Merge sort and its sequential one

No. of Elements	*Sequential sort*	*Parallel (2 processes)*	*Parallel (4 processes)*	*Parallel (8 processes)*
10^5	0.0334	0.0305	0.0287	0.0199
10^6	0.3621	0.2398	0.2226	0.2189
10^7	4.1219	2.5729	2.1589	2.0035

TABLE 3. Required Computing Time for Merge Sorting

Overall Speedup for parallel Merge Sort is shown in Figure 6 and Table 4. It is detected that the computation time improvement for 10^6 elements is not efficiently enhanced using 8 processes.

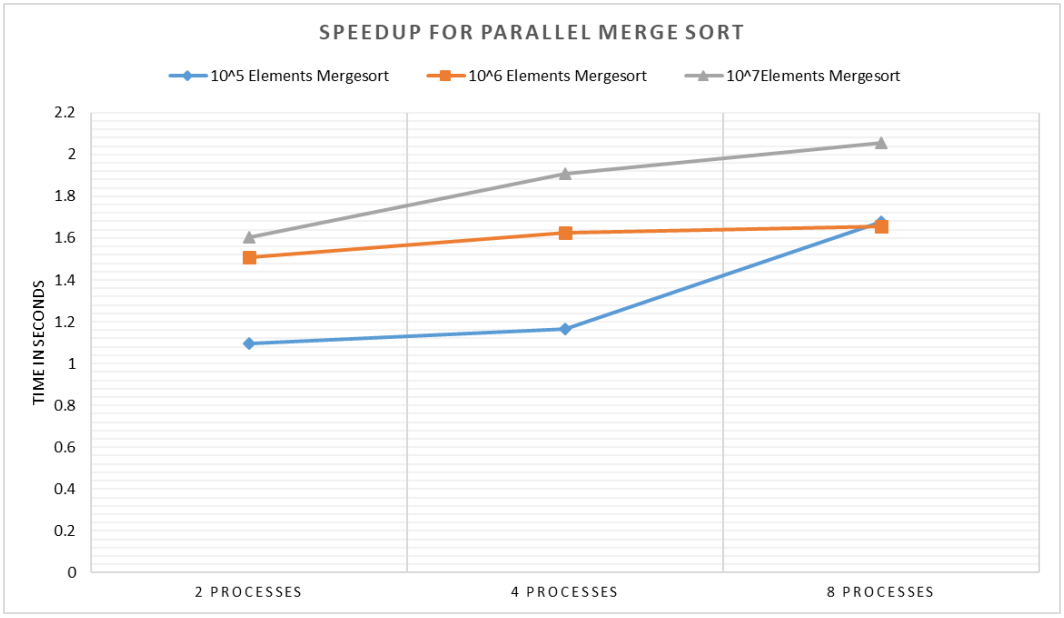

Fig. 6. Speedup for parallel Merge Sort

No. of Elements	*Parallel (2 processes)*	*Parallel (4 processes)*	*Parallel (8 processes)*
10^5	1.0950	1.1637	1.6783
10^6	1.510	1.6266	1.6541
10^7	1.6020	1.9092	2.0573

TABLE 4 Speedup for parallel Merge Sort

Fig. 7. and Table 5 shows the performance of sequential and parallel merge-quicksort algorithms with respect to computation time. It is observed that merge-quicksort algorithm performs better than merge sort algorithm in case of computation time. The experiment revealed that parallel quick sort algorithm is faster, comparing with other mentioned algorithms.

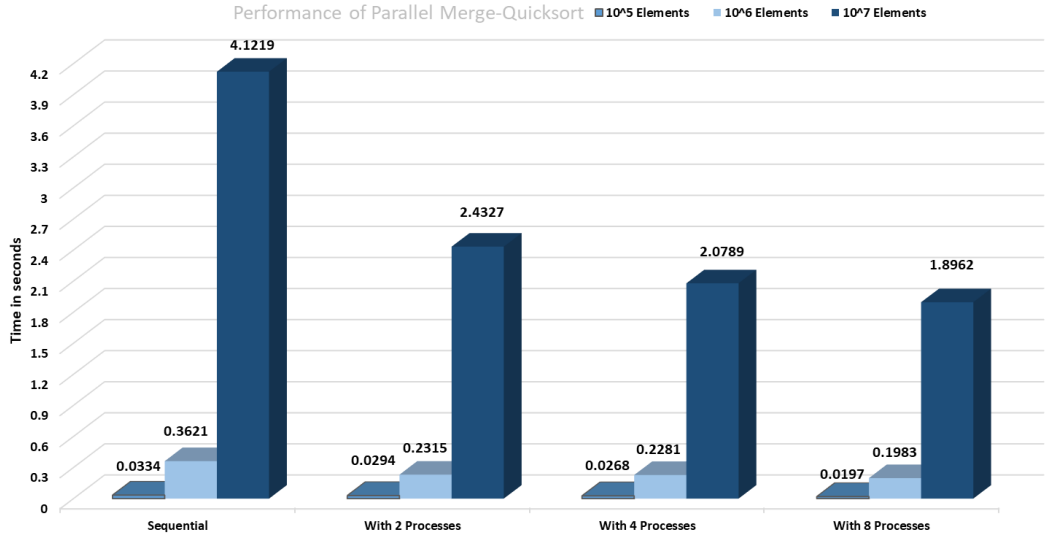

Fig. 7. Performance of Parallel Merge-Quicksort and its sequential one

No. of Elements	*Sequential sort*	*Parallel (2 processes)*	*Parallel (4 processes)*	*Parallel (8 processes)*
10^5	0.0334	0.0294	0.0268	0.0197
10^6	0.3621	0.2315	0.2281	0.1983
10^7	4.1219	2.4327	2.0789	1.8962

TABLE 5. Required Computing Time for Merge-Quicksort

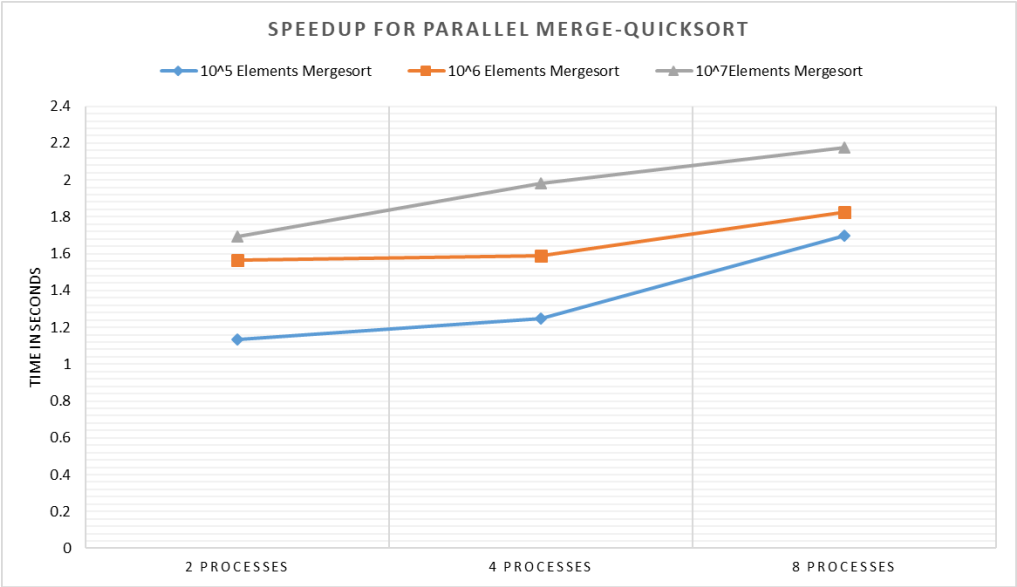

Fig. 8. Speedup for parallel Merge-Quicksort

No. of Elements	***Parallel (2 processes)***	***Parallel (4 processes)***	***Parallel (8 processes)***
10^5	1.1360	1.2462	1.6954
10^6	1.5641	1.5874	1.8260
10^7	1.6943	1.9827	2.1737

TABLE 6 Speedup for parallel Merge-Quicksort

Performance with respect to computing time of previously mentioned parallel sorting algorithms were analyzed as shown Fig.9. The result revealed that parallel quick sort algorithm works faster, however, it was observed that, its communication overhead cost is higher and suffers from load imbalance. Overall Speedup for parallel sorting algorithms and their efficiency of data size of 10^7 are shown in figure 10 and 11 respectively.

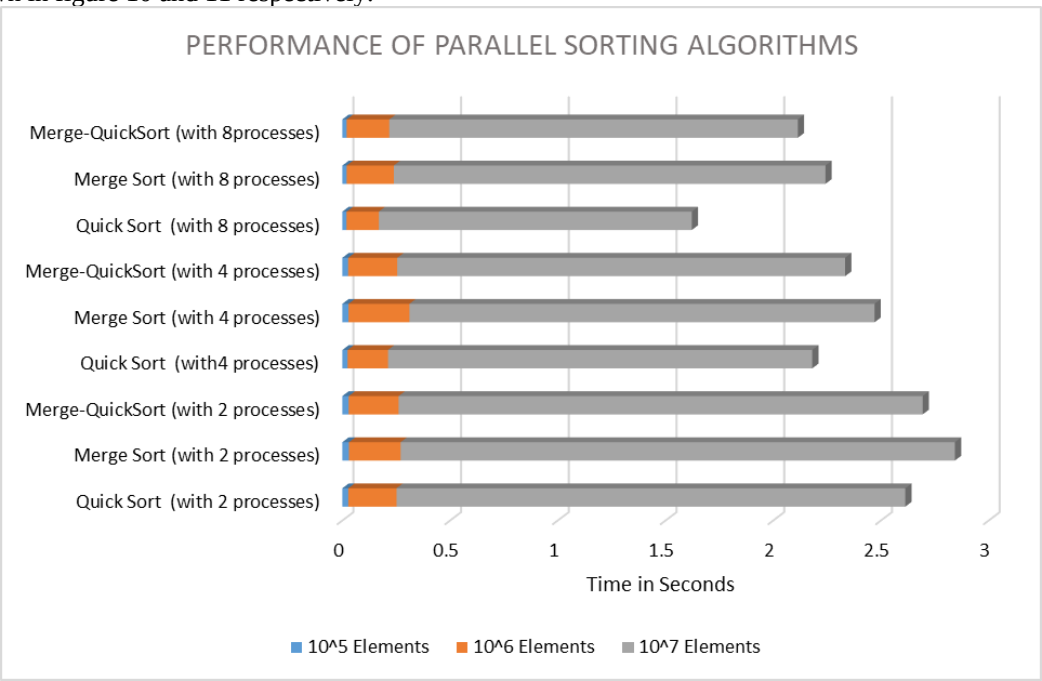

Fig. 9. Computation time of parallel sorting algorithms

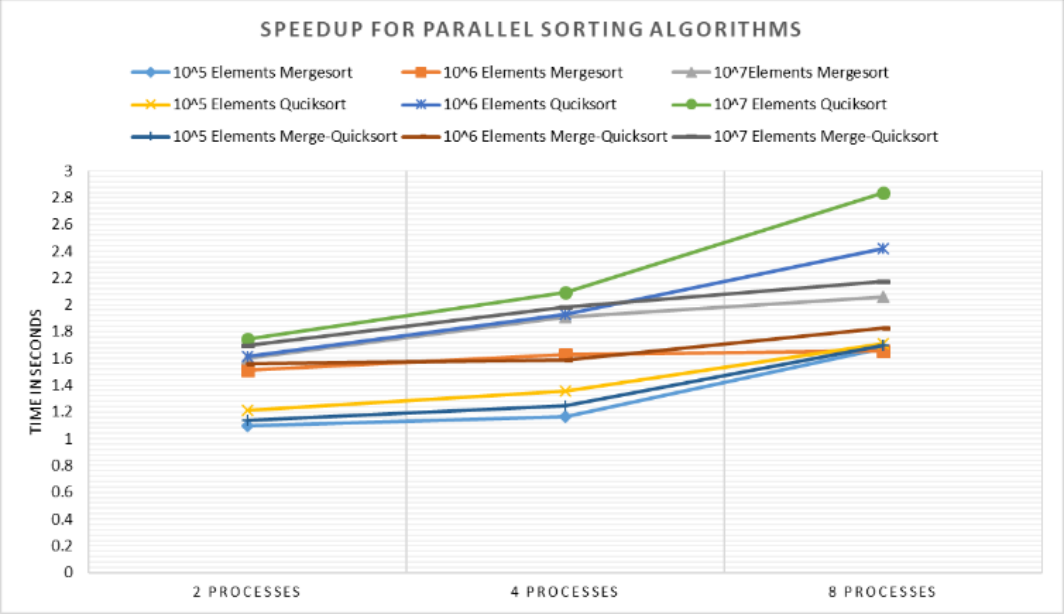

Fig. 10. The speed up achieved of parallel sorting algorithms

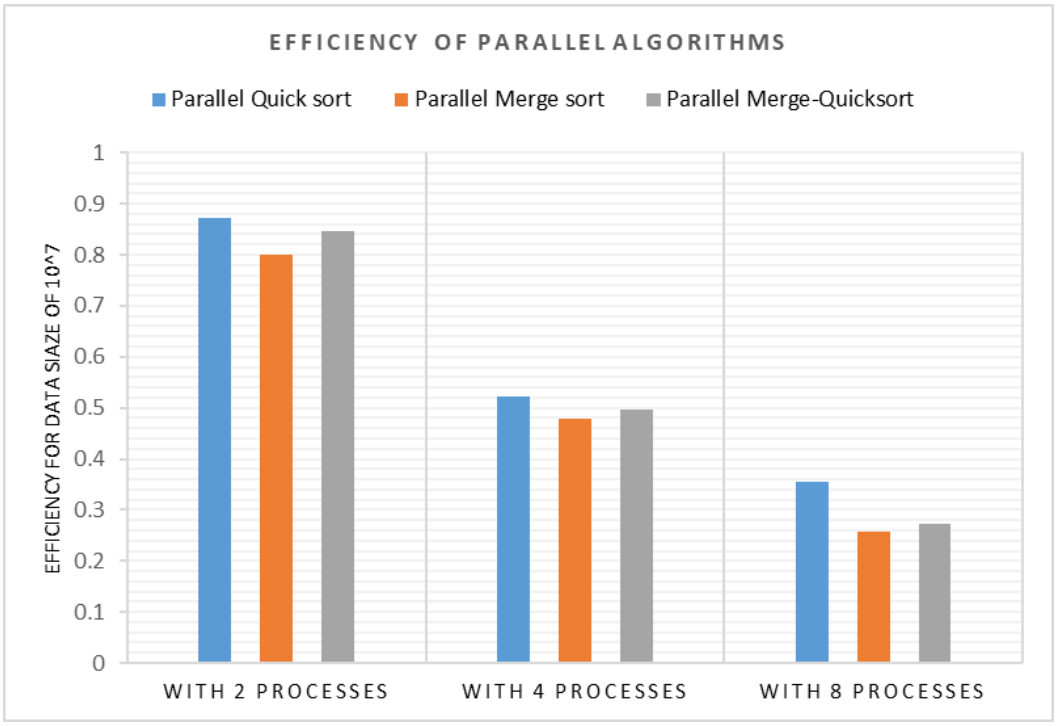

Fig. 10. The efficiency of parallel sorting algorithms for data size of 10^7

4. Conclusion

In this experimental work MPI was used in order to write the parallel code for quick sort, merge sort and hybrid Merge-Quicksort algorithms on Linux platforms.
Performance evaluation was presented in terms of computation time, speed up, and efficiency of parallel algorithms. The evaluation s is based on varying number of processes and size of input. Results indicates that parallel Quicksort algorithm has a better computing time, comparing with other two parallel algorithms. However, using the parallel algorithms for small data set shows no effective change in the computation time. Based on the efficiency of parallel algorithms, the efficiency of parallel Quicksort improved if it is applied on large number of data set and processors.

References

[1] J. Wu, H. Chen, and X. Wang, *Advanced Computer Architecture*, 10th Annual Conference, ACA 2014.

[2] Y. Solihin, *Fundamentals of Parallel Multicore Architecture*. Boca Raton, FL, USA: Chapman & Hall/CRC, 2015.
[3] K. Uchiyama, *Heterogeneous Multicore Processor Technologies for Embedded Systems*. New York, NY, USA: Springer, 2012.
[4] *Algorithms and Architectures for Parallel Processing: 17th International Conference, ICA3PP 2017, Helsinki, Finland, August 21–23, 2017, Proceedings*. Springer.
[5] *Parallel Computing Technologies: 13th International Conference, PaCT 2015, Petrozavodsk, Russia*. Springer, 2015.
[6] S. P. Pacheco, *An Introduction to Parallel Programming*. Amsterdam, Netherlands: Morgan Kaufmann, 2011.
[7] P. S. Pacheco, *An Introduction to Parallel Programming*. Amsterdam, Netherlands: Morgan Kaufmann, 2011.
[8] A. Grama, *Introduction to Parallel Computing*. Harlow, England: Addison-Wesley, 2003.
[9] E. Lusk, S. Huss, B. Saphir, and M. Snir, "MPI: A message-passing interface standard," *Int. J. Supercomput. Appl.*, vol. 8, no. 3/4, pp. 623–654, 2010.
[10] T. Rolfe, "Parallel Merge," [Online]. Available: http://penguin.ewu.edu/~trolfe/ParallelMerge/ParallelMerge.doc
[11] R. Keller, *Recent Advances in the Message Passing Interface: 17th European MPI User's Group Meeting, EuroMPI 2010, Stuttgart, Germany, September 12–15, 2010, Proceedings*. Berlin, Germany: Springer, 2010.
[12] Open-mpi.org, "Open MPI: Open-Source High Performance Computing," 2015. [Online]. Available: http://www.open-mpi.org/. [Accessed: Nov. 19, 2018].
[13] H. El-Rewini and M. Abd-El-Barr, "Message passing interface (MPI)," in *Advanced Computer Architecture*, 2005, pp. 1–42.
[14] H. Brunst, *Tools for High Performance Computing 2011: Proceedings of the 5th International Workshop on Parallel Tools for High Performance Computing, September 2011, ZIH, Dresden*. Berlin, Germany: Springer, 2012.
[15] E. Joseph, *Sorting Algorithm Analysis and Comparison Performance*, Neue Ausg. Saarbrücken: LAP LAMBERT Academic Publishing, 2012.
[16] K. J. Kim, S. J. Cho, and J. W. Jeon, "Parallel quick sort algorithms analysis using OpenMP 3.0 in embedded system," in *Proc. 11th Int. Conf. Control, Automation and Systems (ICCAS)*, Gyeonggi-do, Korea, Oct. 2011, pp. 757–761.
[17] S. Roosta, *Parallel Processing and Parallel Algorithms: Theory and Computation*. New York, NY, USA: Springer-Verlag, 2000.