PERFORMANCE ANALYSIS OF ODD-EVEN MERGE SORT BY USING OPENMP, MPI AND CONCURRENT JAVA

M Rajasekhara Babu, P venkata Krishna, Dinesh Kumar, and V.Shravan
School of Computing Sceince and Engineering, VIT University, Vellore-632014,T.N,India

Abstract: ***In recent years with the advent of programming techniques, parallel programming consumes less execution time as compared to sequential. The odd-even merge sort algorithm was developed by K.E. Batcher [1]. It takes two sorted array and merge them into a single sorted array. In this paper we have implemented this algorithm in three different modules and compared the performance of sequential with parallel. Parallelism is achieved by using #pragma parallel in openmp, using message passing in mpi and by using threads in concurrent java.***

Keyword: odd-even merge sort, parallelism, openmp, mpi, concurrent java

I. INTRODUCTION

Sorting is a method which arranges the list of elements into a particular order. sorting has two different meanings ordering and categorizing, ordering means to order the list of same items and categorizing means grouping and labeling the same type of items[2]. sorting is used in other algorithms that require sorted list to work efficiently. The odd-even merge sort algorithm was developed by K.E. Batcher [1]. Odd even merge sort algorithm can be used for the construction of a systematic sorting network. To construct a systematic sorting network, it is necessary to construct a comparison network that can sort any odd-even sequence. The main idea of this algorithm is that first it sorts the odd position list and the even position list, finally it combines the two sorted list into a single sorted sequence by using a merge algorithm.

In this paper we tried to parallelize this sorting algorithm into three different modules that is openmp, mpi and concurrent java and compared the results of sequence implementation with parallel.

II. ODD-EVEN MERGE SORT:

Algorithm for odd- even merge sort

Input: sequence $t_0, ..., t_{i-1}$ of length $i>1$ whose two halves $t_0, ..., t_{i/2-1}$ and $t_{i/2}, ..., t_{i-1}$ are sorted (i a multiple of 2)

Output: sorted sequence

Method:

```
Begin:
1. Accept the length of the array (in multiples of 2)
2. divide the array into two halves one of even sequence and other of odd sequence
3. for each sequence use parallel partitions
        Do
        Call partition (array,start,end)
        End
4. merge two sorted arrays. Merge(array1,array2,size)
5. End
Partition()
Begin
mergesort()
end
Merge()
Begin
compare two arrays and merge
End
```

Initially we accept a single array from the user and then split into two, one of odd sequence and other of even sequence. Then each sequence is sorted in parallel by using merge sort, and then we get two array which are sorted. The two arrays are merged and sorted into a single array.

II. FLOW CHART

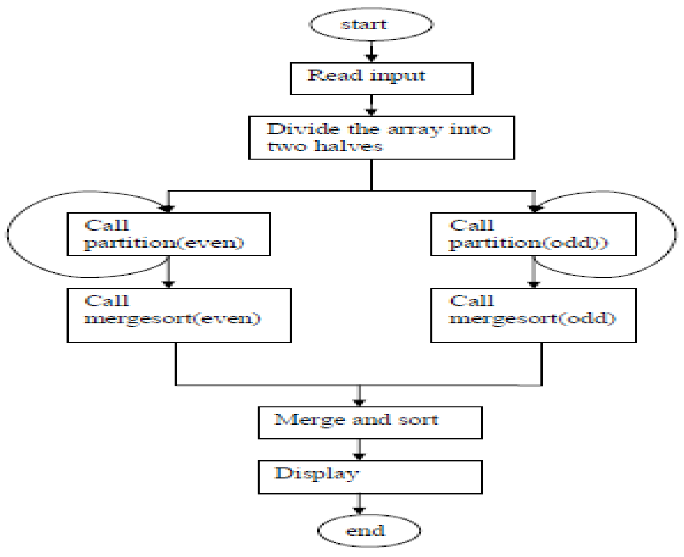


Fig1: flow chart for parallel odd-even merge sort

IV. METHODOLOGY

A. *Openmp:*

Partition(): This method is used for partitioning the odd and even array and then it call the merge sort for each partition.
Merge(): This method is used to combine the two sorted array that is even and odd and form the final single sorted array.
#pragma omp parallel section: independent sections can be parallelized by using this method.
# pragma omp section: It used to divide the section among the threads.
#pragma omp parallel for: The for loop can be parallelized by using this method.

B. *MPI:*

Read_list(): process 0 reads the list from stdin and scatters it to the other processes.
Partition(): To partition the array of add and even array and to sort using merge sort.
Merge():This method is used to combine the two sorted array that is even and odd and form the final single sorted array.
MPI_Init(): This method is used to initialize the MPI execution environment. This is mandatory method for every MPI program. It must be called only once in a program and before any other method is called.
MPI_Comm_size(): It gives the total number of processors available.
MPI_Comm_rank(): It assigns a unique identification number for each processor in use.
MPI_Finalize(): This is the last method to be called by any MPI program .
MPI_Bcast(): This method broadcast a message from the process which have rank “root” to remaining processes.
MPI_Scatter(): It can Distribute distinct messages to all other processes in a group from a single source
MPI_Send(): It is basic blocking send operation. If application buffer of sending task is free then this method returns.

C. *Concurent Java:*

Partition(): This method is used for partitioning the odd and even array and then it call the merge sort for each partition.
Merge(): This method is used to combine the two sorted array that is even and odd and form the final single sorted array.
Run(): When an object is created the interface Runnable is called automatically to create a thread, when the thread is started it calls the run method
Start(): This method start the execution of thread.
Synchronized(): This method can not allow two invocations of the synchronized methods on the same object to interleave.

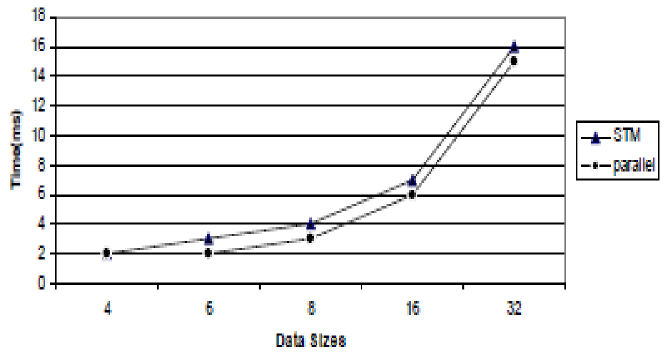
V. RESULTS

A. *OpenMP :*

Data Set	Serial Time(ms)	Parallel Time(ms)
4	2	2
6	3	2
8	4	3
16	7	6
32	16	15

Table 1:Execution time for serial and parallel in openmp

Graph :



Graph 1:execution time(ms) for different data set

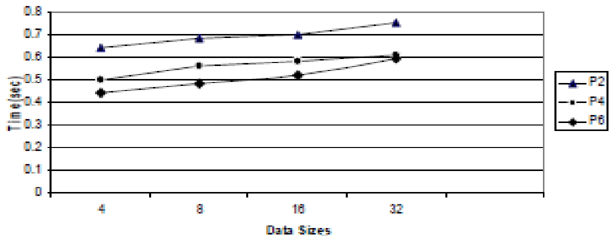
The STM in the graph 1 shows the serial execution time and parallel shows the parallel execution time.
x-axis show the different data sizes and y-axis show the time in milli-sec.

B. MPI :

Data Set	PTM (sec) 2 Processors	PTM(sec) 4 Processors	PTM(sec) 8 Processors
4	0.64	0.5	0.44
8	0.68	0.56	0.48
16	0.7	0.58	0.52
32	0.75	0.61	0.59

Table 2: parallel execution time(sec) for different processors in mpi

Graph :



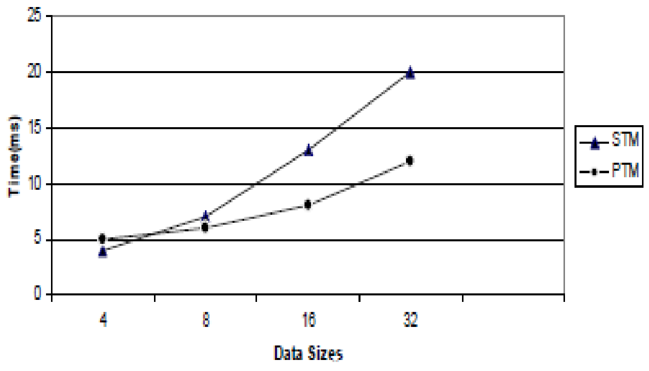
Graph 2: Time taken to execute different number of data sizes
The p2 in the graph 2 shows the two number of processors in the same way p4 and p6 shows four and six number of processors respectively. The x-axis shows the data size and y-axis shows the execution time in seconds

C. Concurrent JAVA:

Data Set	Serial Time(ms)	Parallel Time(ms)
4	4	5
8	7	6
16	13	8
32	20	12

Table 3: Execution time for serial and parallel in concurrent java

Graph :



Graph 3: Time taken to execute different number of data sizes

The STM in the graph 3 represents the serial execution time and PTM represents the parallel execution time. x-axis shows the different data sizes and the y-axis shows the time in milli-seconds.

VI. CONCLUSINONS

In this study we compare execution time for serial and parallel in different modules like MPI ,Concurrent Java, Openmp. When we analyze the parallel code, The execution time for parallel code decreases as the data set increases when compared with the serial code. Thus we can increase the performance of a system by using parallel programming model.

REFERENCES

[1]. K.E. Batcher (1968)- Sorting networks and their applications, Proceedings of the AFIPS Spring Joint Computer Conference 32, 307–314
[2]. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill, 1990. ISBN 0-262-03293-7. Chapter 27: Sorting Networks, pp.704–724S
[3]. Roel Wieringa (December 1998) - A Survey of Structured and Object-Oriented Software Specification Methods and Techniques, ACM Computing Surveys, 30(4):459-527,.
[4] O.Angel, A.E. Holroyd, D. Romik, B. Virag (2007)- Random Sorting Networks, Adv. in Math., 215(2):839–868,
[5]. D.E. Knuth. The Art of Computer Programming, Volume 3: Sorting and Searching, Third Edition. Addison-Wesley, 1997. ISBN 0-201-89685-0. Section 5.3.4: Networks for Sorting, pp. 219–247.
[6]. Ajtai, M.; Komlós, J.; Szemerédi, E. (1983), "An O(n log n) sorting network", Proceedings of the 15th Annual ACM Symposium on Theory of Computing, pp. 1–9, doi:10.1145/800061.808726 .