Analysis of Parallel Merge Sort Algorithm

Manwade K. B.

Department of Computer Science & Engineering, Tatyasaheb Kore Institute of Engineering & Technology, Warananagar, Dist: Kolhapur (MS), India 416113

ABSTRACT
The parallel computing on loosely coupled architecture has been evolved now days because of the availability of fast, inexpensive processors and advancements in communication technologies. The aim of this paper is to evaluate the performance of parallel merge sort algorithm on loosely coupled architecture and compare it with theoretical analysis [1].The parallel computational time complexity is O (p) [3] using p processes and one element in each process. It has been found that there is no major difference between theoretical performance analysis and the actual result.

Keywords
Parallel computing, Parallel Algorithms, Message Passing Interface, Merge sort, Complexity, Parallel Computing.

1. INTRODUCTION
Here, we present a parallel version of the well-known merge sort algorithm. The algorithm assumes that the sequence to be sorted is distributed and so generates a distributed sorted sequence. For simplicity, we assume that *N* is an integer multiple of *P*, that the *N* data are distributed evenly among *P* tasks. Also we have analyzed the performance of the proposed algorithm and it is compared with theoretical analysis.
The sequential merge sort requires O (N log N) [3] time to sort *N* elements, which is the best that can be achieved (modulo constant factors) unless data are known to have special properties such as a known distribution or degeneracy. This paper describes implementation of the merge sort within a parallel processing environment. In the fully parallel model, you repeatedly split the sub lists down to the point where you have single-element lists. You then merge these in parallel back up the processing tree until you obtain the fully merged list at the top of the tree.

Manuscript received October 5, 2009.
K. B. Manwade was with Department of Computer Science & Engineering TKIET Warananagar, Kolhapur 416113 INDIA. He is now with the Department of Computer

While of theoretical interest, you probably don't have the massively parallel processor that this would require.

2. THE PARALLEL ALGORITHM
This algorithm uses master slave model [4] in the form of tree for parallel sorting. Each process receives the list of elements from its precedor process then divides it into two halves, keeps one half for itself and sends the second half for its successor. To address the corresponding preccedor & successor we have used the concept of 'myrank_multiple'. For a process having odd rank [5] it is calculated as Myrank_multiple=2*Myrank+1;Temp_myrank=Myrank_multiple and for the process having even rank it is calculated as Myrank_multiple=2*Myrank+2;Temp_myrank=Myrank_multiple. It uses recursive calls both to emulate the transmission of the right halves of the arrays and the recursive calls that process the left halves. When the number of processors in the system exhaust then each processor will sort the remaining data. After that it will receive the sorted data from its successor & merge that two sub lists. Then it sends the result to its precedor. This process will continues up to root node.

```
procedure parallel_mergesort(DataArray,SizeofData)
  Begin
        MyData=LeftHalfof[DataArray]
        TempData=RightHalfof[DataArray]
        Send(TempData)
    MyData = Mergesort(MyData,i,j)
    Receive(TempData)
        DataArray=MergeResult(MyData,TempData)
  End
procedure Mergesort(MyData,i,i)
  Begin
    If(j-i>16)
        {
    MergeSort(MyData,i,(i+j)/2)
        MergeSort(MyData,(i+j)/2,j)
   }
     Else
 InsertionSort(MyData,i,j)
    End
procedure InsertionSort (MyData,i,j)
    Begin
        //Sequential_ InsertionSort
    End
```

3. THEORETICAL ANALYSIS
The sequential time complexity is O(nlogn). In case of parallel algorithm the complexity involves both communication cost and computational cost.

A. Communication

In the division phase, communication only takes place as follows,
Communication at each step,

$$\text{Tstartup} + \left(\frac{n}{2}\right)\text{Tdata} \qquad P0 \rightarrow P1$$

$$\text{Tstartup} + \left(\frac{n}{4}\right)\text{Tdata} \quad P0 \rightarrow P2, \quad P1 \rightarrow P3$$

$$\text{Tstartup} + \left(\frac{n}{4}\right)\text{Tdata} \quad \text{P0} -> \text{p2}, \quad \text{p1} -> \text{p3}$$

$$\text{Tstartup} + \left(\frac{n}{4}\right)\text{Tdata} \quad \text{P0} -> \text{p2}, \quad \text{p1} -> \text{p3}$$

$$\text{Tstartup} + \left(\frac{n}{8}\right)\text{Tdata} \quad \text{P0} -> \text{p4}, \quad \text{p2} -> \text{p6},$$

$$\text{p1} -> \text{p5}, \text{p3} -> \text{p1}$$

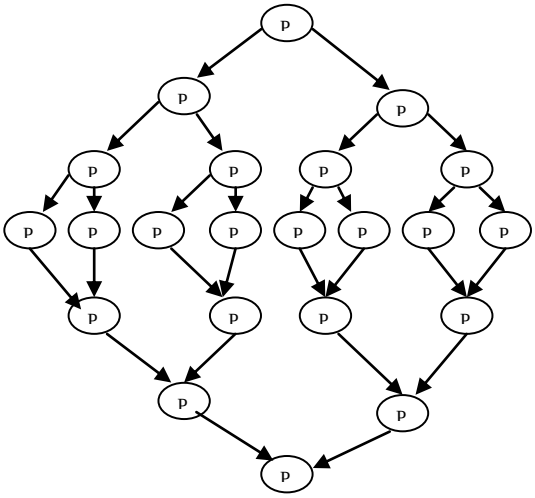

Fig. 1The merge sort tree

With log(P) steps given P processors, in the merge phase the reverse communication takes place.

$$\text{Tstartup} + \left(\frac{n}{8}\right)\text{Tdata} \quad \text{P4} -> \text{P0}, \quad \text{P6} -> \text{p2}, \quad \text{P5} -> \text{P1}, \text{P7} -> \text{P3}$$

$$\text{Tstartup} + \left(\frac{n}{4}\right)\text{Tdata} \quad \text{P2} -> \text{P0}, \quad \text{P3} -> \text{P1}$$

$$\text{Tstartup} + \left(\frac{n}{2}\right)\text{Tdata} \quad \text{P1} -> \text{P0}$$

Again log P steps .this leads to the communication time being

$$\text{Tcomm} = 2\left(\text{Tstartup} + \left(\frac{n}{2}\right)\text{Tdata} + \text{Tstartup} + \left(\frac{n}{4}\right)\text{Tdata} + \text{Tstartup} + \left(\frac{n}{4}\right)\text{Tdata} + \cdots \ldots .\right)$$

$$= 2(\log P)\,\text{Tstartup} + 2n\,\text{Tdata}$$

B. Computation

Computation only occurs in merging the sub lists merging can be done by stepping through each list, moving the smallest found into the final list first. It takes 2n-1 steps in the worst case to merge two sorted lists each of n numbers into one sorted list in this manner. Therefore, the computation consists of

$$\text{Tcomp} = 1 \quad \text{P0; P2; P1; P3}$$

$$\text{Tcomp} = 3 \quad \text{P0; P1}$$

$$\text{Tcomp} = 7 \quad \text{P0}$$

Hence:

$$d = \text{Tcomp} = \sum \log P \; i = 1 \; (2i - 1)$$

Therefore the total time required is,

$$\text{Ttotal} = \text{Tcomp} + \text{Tcomm}$$

$$= \sum \log P \; i = 1 \; (2i - 1) + 2(\log P)\text{tstart} + 2\text{ntdata}$$

Let us analyze the time by varying the number of processors by keeping n=100,

TABLE I
COMPUTATION TIME REQUIRED FOR MERGE SORTING FOR N=100 BY VARYING NUMBER OF PROCESSORS

No. of Processors	Ttotal
10	203
20	203.60
30	203.95
40	204.20
50	204.39
60	204.56
70	204.69
80	204.80
90	204.91
100	208

Now by keeping the number of processors constant to P=10 & then varying the number of elements, the time required is,

TABLE II
COMPUTATION TIME REQUIRED FOR MERGE SORTING FOR P=10 BY VARYING NUMBER OF ELEMENTS

N (No. of elements)	Ttotal
100	203
200	403
300	603
400	803
500	1003
600	1203
700	1403
800	1603
900	1803
1000	2003

The graphical representations of the result are as follows,

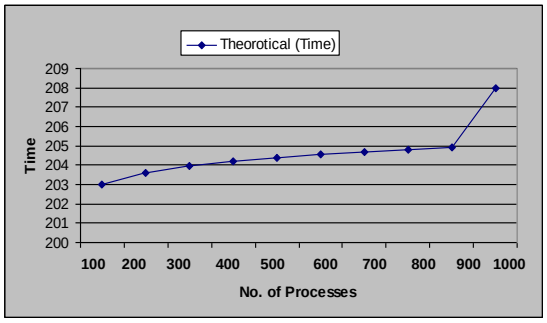

Fig. 2 Theoretical Computation time versus number of processes

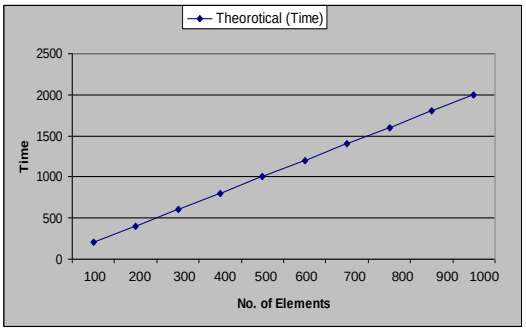

Fig.3 Theoretical Computation time versus number of elements

4. PRACTICAL ANALYSIS

We have carried out the experiments in LAN. The computing environment we have used is MPI [4], [5]. To analyze the performance of the algorithm we have used two strategies,

Keep the array size fixed i.e. N=100 and vary the number of processors.

Keep the number of processors fixed i.e. P=10 and vary the size of elements.

We have observed the following results; Table3 shows the reading for first strategy and Table4 shows the reading for second strategy.

TABLE III

COMPUTATION TIME (PRACTICAL) REQUIRED FOR MERGE SORTING VARING NUMBER OF PROCESSORS

No. of Processors	Actual Time (Sec.)
16	0.68522
17	0.692969
18	0.73497
19	0.747092
20	0.752514
21	0.825803
22	0.826923
23	0.806718
24	0.889703
25	0.876956

TABLE IV

COMPUTATION TIME (PRACTICAL) REQUIRED FOR MERGE SORTING FOR P=10 BY VARYING NUMBER OF ELEMENTS

N (No. of elements)	Actual Time (Sec.)
100	0.129613
200	0.165852
300	0.17503
400	0.237113
500	0.403223
600	0.438486
700	0.504024
800	0.628536
900	0.65149
1000	0.69522

The graphical representations of the result are as follows,

5. RESULT

After plotting the results from table3 and table4, we have got the following graphs. Figure4 shows the graph of results for strategy number one, while the Figure5 shows the graph of results for strategy number two.

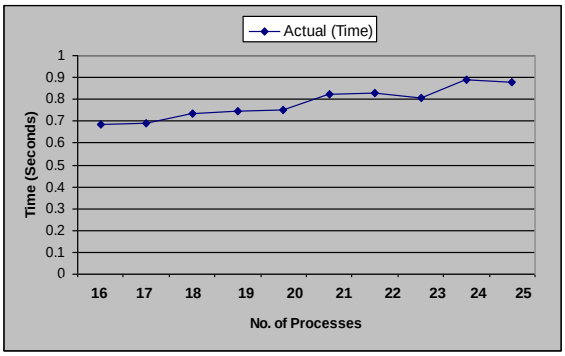

Fig. 4 Actual Computation time versus number of processes

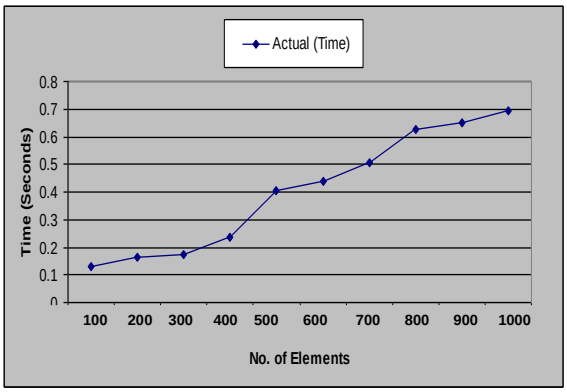

Fig.5 Actual Computation time versus number of elements

6. CONCLUSION

The algorithm has been tested on loosely coupled parallel machines and the performance of the algorithm has been observed. It has been found that the computational time of the algorithm varies logarithmically for varying number of processors scenario. Also it is found that for varying number of elements the computational time varies linearly. It is also found that the practical analysis closely matches with theoretical analysis.

7. REFERENCES

[1]. K.B.Manwade, R.B.Patil; Parallel merge sort on loosely coupled architecture; National Conference, PSG Coimbator.

[2].Ellis Horowitz, Sartaj Sahani, Sanguthevar Rajasekaran, "Computer Algorithms", Galgotia publication.

[3].Barry Wilkinson & Michael Charlotte, "Parallel programming techniques and applications using networked workstations and parallel computers", Pearson publication.

[4].http://penguin.ewu.edu/~trolfe/ParallelMerge/ParallelMerge.doc

[5]. Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. Technical Report Version 1.0, University of Tennessee, Knoxville, Tennessee, June 1995.

[6]. The MPI Forum On-line, http://www.mpi-forum.org.

[7].Ananath Grama, Anshul Gupta, "Parallel Computing", Second edition, Addison-Wesley.