An Adaptive Framework towards Analyzing the Parallel Merge Sort

Husain Ullah Khan [1], Rajesh Tiwari [2]

[1] ME Scholar, Department Of Computer Science & Engineering, Shri Shankaracharya College of Engineering & Technology Bhilai, India,
khan.husain@gamil.com.

[2] Department Of Computer Science & Engineering, Shri Shankaracharya College of Engineering & Technology, Bhilai, India,
raj_tiwari_in@yahoo.com

Abstract: ***The parallel computing on loosely coupled architecture has been evolved now a day because of the availability of fast and, inexpensive processors and advancements in communication technologies. The aim of this paper is to evaluate the performance of parallel merge sort algorithm on parallel programming environments such as MPI .The MPI libraries has been used to established the communication and synchronization between the processes Merge sort is analyze in this paper because it is an efficient divide-and-conquer sorting algorithm. it is easier to understand than other useful divide-and-conquer strategies Due to the importance of distributed computing power of workstations or PCs connected in a local area network. Our aim is to study the performance evaluation of parallel merge*** **sort.**

Keywords: parallel computing, parallel Algorithms, Message Passing Interface, Merge sort, performance analysis.

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. The sequential merge sort requires O (n logn) [5] time to sort n elements. Due to the importance of distributed computing power of workstations or PCs connected in a local area network [7] researcher have been studying the performance of various scientific applications [6][7]][8].

2. Methodology of Parallel Merge

Methodology used for parallel algorithm uses master slave model 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 it and sends the second half for its successor. To address the corresponding preccedor & successor we have used the concept of rank calculation.
For a process having odd rank 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. 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 continue up to root node.

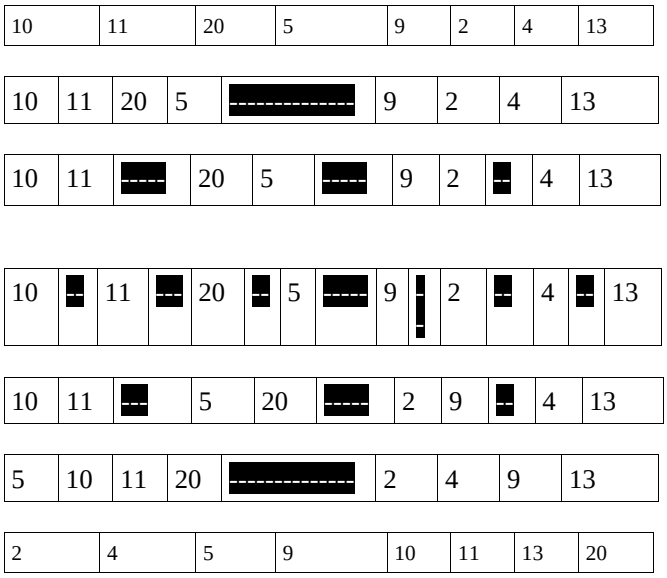

Figure: -2.1 Communications and sorting in Merge Sort

In above figure divide and conquer strategies are shown. Here list is divided into two sub lists and again sub list is divided again two sub list, this process is repeated until we found single element. Then merge process start with sorting. Finally we get sorted sequence.

3. Communication and Processing

The sequential time complexity is O(n logn). In case of parallel algorithm the complexity involves both communication cost and computational cost. The

communication of sub problems is an overhead expense that we want to minimize. Also, there's no reason to allow an internal node process to sit idle, waiting to receive two results from its children. Instead, we want the parent to send half the work to the child process and then accomplish half of the work itself. Central to computing within MPI [2] is the concept of a "communicator". The MPI communicator specifies a group of processes inside which communication occurs. MPI_COMM_WORLD is the initial communicator, containing all processes involved in the computation. Each process communicates with the others through that communicator, and has the ability to find position within the communicator and also the total number of processes in the communicator.

In the division phase, communication only takes place as follows, Communication at each step, The root process prepares the array to sort and then invokes parallel merge sort [1] while each helper process receives data from its parent process and invokes parallel merge sort and sends sorted data back to parent .

Tcomm=2(Tstartup+(n/2)Tdata+Tstartup+(n/4)Tdata+…)

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

Tcomp=1 P1;P2;P1;P3
Tcomp=3 P0;P1
Tcomp=7 P0

Therefore the total time required is,

Ttotal= Tcomp+Tcomm

4. Choosing the Parallel Environment

MPI (Message Passing Interface): MPI is an easily used parallel processing environment whether our target system is a single multiprocessor computer with shared memory or a number of networked computers: the Message Passing Interface (MPI) [4] As its name implies, processing is performed through the exchange of messages among the processes that are cooperating in the computation. Central to computing within MPI is the concept of a "communicator". The MPI communicator specifies a group of processes inside which communication occurs. MPI_COMM_WORLD is the initial communicator, containing all processes involved in the computation. Each process communicates with the others through that communicator, and has the ability to find position within the communicator and also the total number of processes in the communicator. Through the communicator, processes have the ability to exchange messages with each other. The sender of the message specifies the process to receive the message. In addition, the sender attaches to the message something called a message tag, an indication of the kind of message it is. Since these tags are simply non-negative integers, a large number is available to the parallel programmer, since that is the person who decides what the tags are within the parallel problem solving system being developed. The process receiving a message specifies both from what process it is willing to receive a message and what the message tag is. In addition, however, the receiving process has the capability of using wild cards, one specifying that it will accept a message from any sender, the other specifying that it will accept a message with any message tag. When the receiving process uses wild card specifications, MPI provides a means by which the receiving process can determine the sending process and the tag used in sending the message. For the parallel sorting program, we can get by with just one kind of receive, the one that blocks execution until a message of the specified sender and tag is available. you need to initialize within the MPI environment. The presumption is that this one is called from the programs main, and so it sends pointers to the argc and argv that it received from the operating system. We choose MPI to implement message-passing merge sort on single and networked computers because

(i) MPI is implemented for a broad variety of Architectures, including implementations that are freely available.

(ii) MPI is well documented;

(iii) MPI is more popular with parallel platform than other parallel platforms, such as PVM [3] and OpenMP. And our preference for an implementation language is ANSI C because C is fast and available on virtually any platform and C is easy to implement.

Parallel merge sort is executed by various processes at various levels of the process tree, with the root being at level 0, its children at level 1, and so on . In that, the process's level and the MPI process rank are used to calculate a corresponding helper process's rank. Then, merge sort communicates for further sorting half of the array with that helper process. Serial merge sort is invoked when no more MPI helper processes are available.

Procedure for parallel merge sort:

Step.1 Calculate Rank of the process.
Step.2 Assign rank to parent process and child processes.
Step.3 Check for the left & right child according to the rank of the process.
Step.4 sort left sub arrays.
Step.5 check for the return status from child processes.
Step.6 send sorted data to parent process.
Step.7 repeats the step 4 for right sub array.
Step.8 Merge the two results back into new array

To execute above steps successfully there are various functions that can be used, some of them are.

1. MPI_Init(int *argc, char ***argv) //initialize MPI
2. MPI_Comm_rank (MPI_Comm comm, int *rank)
   //This process's position within the

communicator
3. MPI_Comm_size (MPI_Comm comm, int *size)
//Total number of processes in the communicator
4. MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
// Send a message to process with rank dest using tag
5. MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag,MPI_Comm comm , MPI_Status *status)
//Receive a message with the specified tag from the process with the rank source
6. MPI_Finalize() //Finalize MPI

5. Performance Evaluation and Result

Output from Communications Test
During the divide phase processes which are communicating to each other is shown in tabular form.

Table : - 5.1 Communication process Table

Building a height 3 tree	0 sending data to 4
0 sending data to 2	0 sending data to 1
1 transmitting to 0	0 getting data from 1
2 sending data to 3	3 transmitting to 2
2 getting data from 3	2 transmitting to 0
0 getting data from 2	4 sending data to 6
4 sending data to 5	5 transmitting to 4
4 getting data from 5	6 sending data to 7
7 transmitting to 6	6 getting data from 7
6 transmitting to 4	4 getting data from 6
4transmitting to 0	0 getting data from 4

Single Computer Results in MPI
4 processes mandate root height of 2

Size:	10000000
Parallel:	3.877
Sequential:	11.607
Speed-up:	2.994

8 processes mandate root height of 3

Size:	10000000
Parallel:	3.643
Sequential:	11.57
Speed-up:	3.18

On the other hand, you can force network communications by running the application in an MPI session involving multiple computers, and letting MPI think that each computer has only one processor. In this environment, the mismatch between processor speed and network communications speed becomes obvious.

Networked Computer Results in MPI

4 processes mandate root height of 2

Size:	10000000
Parallel:	8.421
Sequential:	11.611
Speed-up:	1.379

8 processes mandate root height of 3

Size:	10000000
Parallel:	8.168
Sequential:	11.879
Speed-up:	1.4548

Any network communications drastically degrades the performance. MPI know that each computer has four processors. In that case, MPI will deal out processes by fours before it moves to the next available computer. Thus, if you ask for eight parallel processes, ranks 0 through 3 will be on one computer, with fastest possible communications, and ranks 4 through 7 will be on another computer. Thus the only messaging is at the root, when rank 0 sends its right half to rank 4.

6. Conclusion

This paper introduces parallel environment and its implementation using MPI with C for parallel merge sort. The paper reports performance experiments of MPI in which it is concluded accordingly. A fast network can make a message-passing (with MPI) solution for some problem faster. Fast network that avoids a potentially large network overhead the next time the same resources are requested so it can provide better performance.

References

[1] K.B.Manwade, R.B.Patil; “Parallel merge sort on loosely coupled architecture”, National Conference, PSG Coimbatore.
[2] The OpenMP specification for parallel programming. Retrieved on March 1, 2011 from http://openmp.org.
[3] The Message Passing Interface (MPI) standard. Retrieved on March1, 2011from http://www.mcs.anl.gov/research/projects/mpi/.
[4] Radenski Shared Memory, Message Passing, and Hybrid Merge Sorts for Standalone and Clustered SMPs Proc .PDPTA’11, the 2011 international conference of parallel and distributed processing technique and applications, CSREA press (H.Arabnia,Ed.), 2011 pp.367-373.
[5] Wilkinson & Michael Charlotte, “Parallel programming techniques and applications using networked workstations and parallel computers”, Pearson publication.

[6] 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.
[7] 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.
[8] 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.

Husain Ullah Khan: M.E.(Pursuing) in Computer Technology & Application from Shri Shankaracharya College of Engineering & Technology, CSVTU, Bhilai ,India. Research areas are Parallel Computing & its Enhancement.

Rajesh Tiwari: Received the M.E. degree in Computer Technology & Applications from Shri Shankaracharya College of Engineering & Technology, Bhilai, India. Currently he is pursuing Ph.D from CSVTU, Bhilai. He is having nine years long experience in the field of teaching. Research areas are Parallel Computing and its Enhancement, His research work has been published in many national and international journals.