PERFORMANCE EVALUATION OF LARGER MATRICES OVER CLUSTER OF FOUR NODES USING MPI

Sampath S[1], Bharat Bhushan Sagar[2]

[1]*Departmen[t] of CS&E, Research Scholar, Shri Venkateshwara University, Gajraula, Uttarpradesh, India*
[2]*Department of CS, Assistant Professor, Birla Institute of Technology, Noida, Uttarpradesh, India*

Abstract

Parallel computing operates on the principle that large problems can often be divided into smaller ones, which are then solved concurrently to save time (wall clock time) by taking advantage of non-local resources and overcoming memory constraints. The main aim is to form a cluster based parallel computing architecture for MPI based applications which demonstrates the performance gain and losses achieved through parallel processing using MPI. This can be realized by implementing the parallel applications like solving matrix multiplication problem, using MPI. The architecture for demonstrating MPI based parallel applications works on the Master-Slave computing paradigm. We aim to evaluate the time statistics of parallel execution and do comparison with the time taken to solve the same problem in serial execution. We also demonstrate communication overhead involved in parallel computation. The results with runs on different number of nodes are compared to evaluate the efficiency of MPI based parallel applications. We also show the performance dependency of parallel and serial computation, on RAM. Finally we show the relationship between the number of slave processes to be specified for computation and the number of cores available for parallel computation.

Keywords: *Parallel Execution, Cluster Computing, Symmetric Multi-Processor (SMP), MPI (Message Passing Interface), RAM (Random Access Memory).*

----------***----------

1. INTRODUCTION

Parallel processing refers to the concept of speeding up the execution of a program by dividing the program into multiple fragments that can execute simultaneously, each on its own processor. This paper deals how to handle Matrix Multiplication problem that can be split into sub-problems and each sub-problem can be solved simultaneously. With computers being networked today, it has become possible to share resources like files, printers, scanners, fax machines, email servers, etc. One such resource that can be shared but is generally not, is the CPU. Today's processors are highly advanced and very fast, capable of thousands of operations per second. If this computing power is used collaboratively to solve bigger problems, the time taken to solve the problem can reduce drastically.

1.1 Existing Frameworks

- MPI: The specification of the Message Passing Interface (MPI) standard 1.0 [5] was Completed in April of 1994. This was the result of a community effort to try and define Both the syntax and semantics of a process message-passing library that would be useful to a Wide range of users and implemented on a wide range of Massively Parallel Processor (MPP) platforms.
- MPI2: All major computer vendors supported the MPI standard and work began on MPI-2, where new functionality, dynamic process management, one-sided communication, cooperative I/O, C++ bindings, Fortran 90 additions, extended collective operations, and miscellaneous other functionality were added to the MPI-1 standard [5]. MPI-1.2 and MPI-2 were released at the same time in July of 1997. The main advantage of establishing a message-passing standard is portability.
- Openmp: It has emerged as the standard for shared-memory parallel programming. The openmp application program interface (API) provides programmers with a simple way to develop parallel application for shared memory parallel computing.
- MPICH2: An all-new implementation of MPI designed to support both MPI-1 and MPI-2. In MPICH2, the collective routines are significantly faster and has very low communication overhead than the “classic” MPI and MPICH versions [6].

1.2 Framework used in the Proposed System

This paper deals with the implementation of parallel application, matrix multiplication under MPI using MPICH2 for communication between the process and for the computation. Because they are very much suitable to implement in LINUX systems

2. RELATED WORKS

Traditionally, multiple processors were provided within a specially designed "parallel computer"; along these lines, Linux now supports SMP Pentium systems in which multiple processors share a single memory and bus interface

within a single computer. It is also possible for a group of computers (for example, a group of PCs each running Linux) to be interconnected by a network to form a parallel-processing cluster [7]. Amit Chhabra, Gurvinder Singh (2010) [1] proposed Cluster based parallel computing framework which is based on the Master-Slave computing paradigm and it emulates the parallel computing environment. Petre Anghelescu (2012) [2] showed how the implementation of a matrix multiplication on a network computers can be accomplished using the MPI (Message Passing Interface) standard and presented extensive experimental results regarding the performance issues of matrix parallel multiplication algorithms. Rajkumar Sharma, Priyesh Kanungo, Manohar Chandwani (2011) [3] evaluated performance of parallel applications using MPI on cluster of nodes having different computing powers in terms of hardware attributes/parameters. Sampath S, Sudeepa, Nanjesh B.R (2012) [4] presented the MPI framework that demonstrates the performance gain and losses achieved through parallel/distributed processing and made the performance analysis and evaluation of parallel applications using this cluster based parallel computing framework. We aim to present an architecture using MPI that demonstrates the performance gain and losses achieved through parallel/distributed processing. And also demonstrates the performance dependency of parallel applications on RAM.

3. SYSTEM REQUIREMENTS

3.1 Hardware Requirements

- Processor: Pentium D (3 G Hz)
- Two RAM: 1GB and 2GB
- Hard Disk Free Space: 5 GB

3.2 Software Requirements

- Operating System : Linux
- Version: Fedora Process 14
- Compiler: GCC
- Network protocol: Secure Shell
- Communication protocol: MPI

4. SYSTEM DESIGN

4.1 Cluster Based Parallel Computing architecture

The main problem is taken by the master process and assigns the task into slave process. Each slave process send back the solutions of the assigned sub problem. The working principle involved in this architecture is shown in Fig.1 shows the cluster based parallel computing architecture.

4.2 MPI Configuration

Download the mpich-2 package and type the following commands in the terminal to install.

Unpack the tar file and go to the top level directory:
tarxzf mpich2-1.3.2.tar.gz
cd mpich2-1.3.2

Configure MPICH2 specifying the installation directory:
./configure --prefix=/home/<USERNAME>/mpich2-install |& tee c.txt
Build MPICH2:make 2>&1 | tee m.txt
Install the MPICH2 commands:
Make install 2>&1 | tee mi.txt

Add the bin subdirectory of the installation directory to your path in your startup script (.bashrc for bash, .cshrc for csh):
PATH=/home/<USERNAME>/mpich2-install/bin:$PATH;
export PATH.

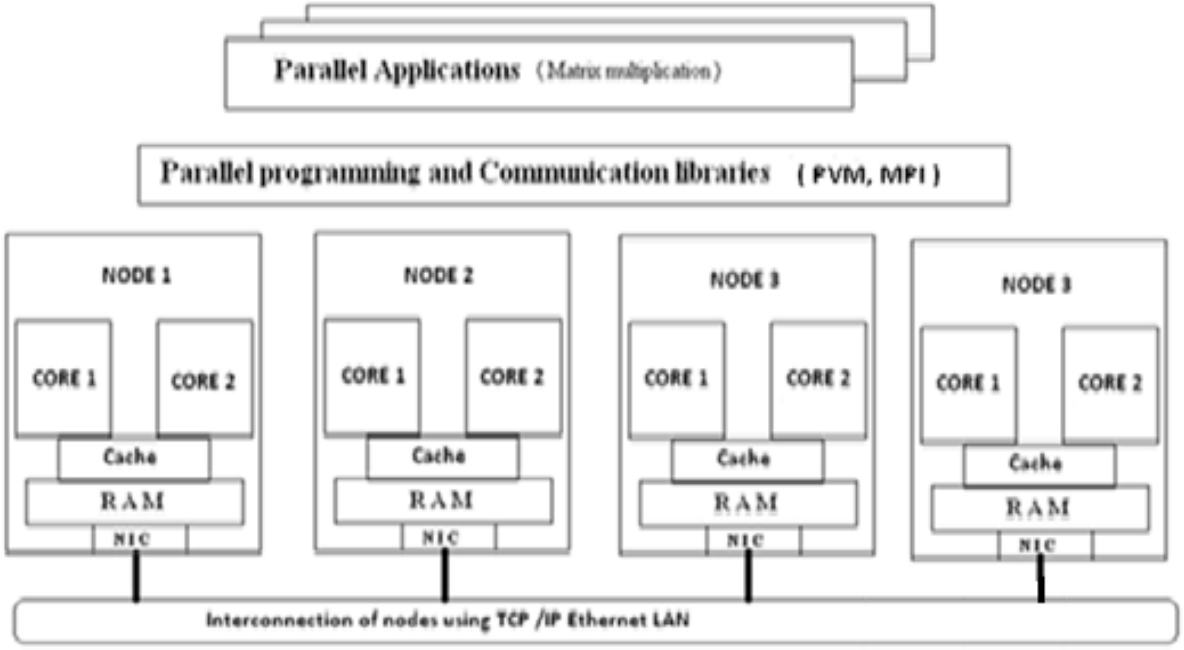


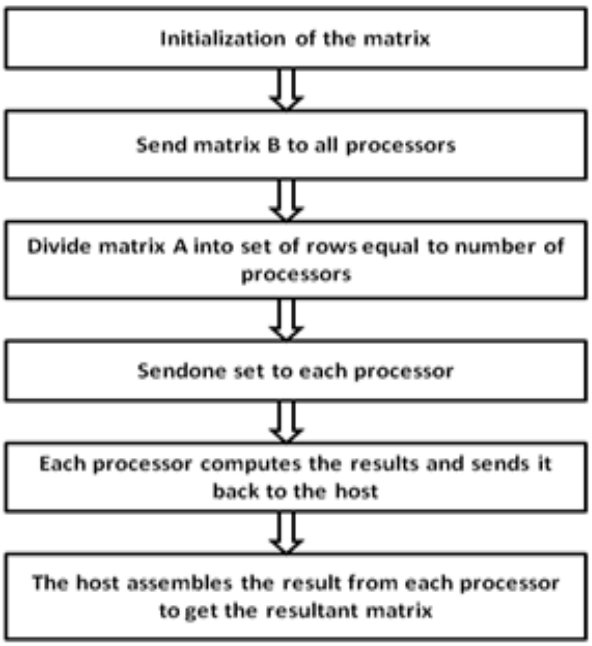
Fig 1.Cluster based parallel computing architecture using three nodes

5. IMPLEMENTATION

The problem to be solved has to be parallelized so that computation time is reduced. The architecture consists of a client, a master process, capable of handling requests from the client, and slave, capable of accepting problems from the master and sending the solution back. The master and the slave communicate with each other using MPICH2 under MPI. The problem has to be divided such that the communication between the master and the slaves is minimum. The total computational time to solve the problem completely is effected by the communication time between the nodes.

5.1 Parallel Matrix Multiplication Design

In the algorithm which we have implemented is for solving matrix multiplication problem on several nodes it may be for only one or more slaves. It divides the matrix into set of rows and sends it to the slaves rather than sending one row at a time [6].The slaves compute the entire set of rows that they have received and send it back to the master in one send operation. Hence, we need to implement parallel systems consisting of set of independent desktop PCs interconnected by fast LAN cooperatively working together as a single integrated set of independent desktop PCs interconnected by fast LAN cooperatively working together as a single integrated computing resource so as to provide higher availability, reliability and scalability. But to show the performance dependency on RAM we are considering only single node with two processes, one act as master and other as slave. So there will be no division of problem, instead entire problem is submitted to the single available slave. Rest of the work is carried out with the multiple nodes. Consider two matrix, matrix A and B. The flow of multiplication of matrix A and B takes place as shown in Fig. 2a. The operations involved in dividing first matrix into set of rows and multiplying each set with entire second matrix giving resultant matrix is shown in Fig. 2b.



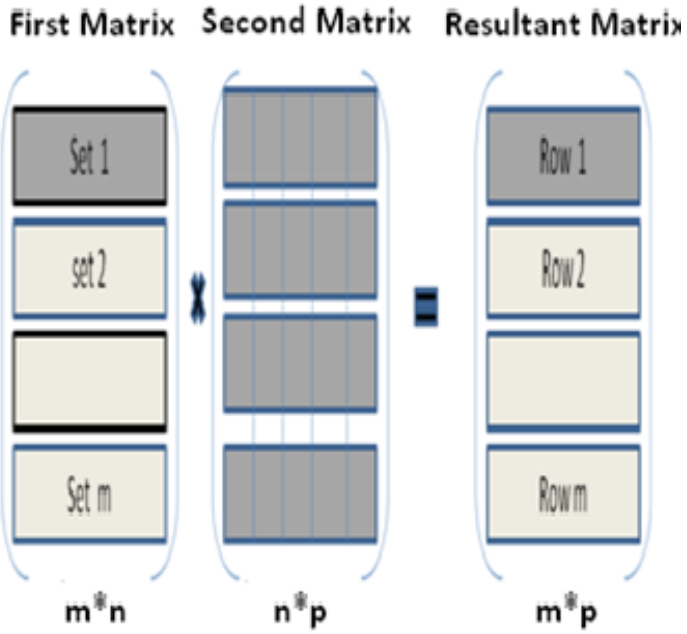


Fig 2a Flow diagram for solving matrix multiplication problem on several Nodes **Fig 2b** Parallel matrix multiplication design

6. RESULTS AND ANALYSIS

Case1: Analysis to show the relation between number of cores and the number of processes. If multiple hosts are in the hostfile, then mpirun will attempt to run processes on these hosts, in our work we use three nodes each of having two cores. MPI by default start assigning the slaves to cores present in first host or master host and then to the cores present in next host. MPI follows the order which is present in hostfile for the utilization of nodes. Table I shows that If the number of processes created is n then the number of cores utilized is equal to n.

The number of processes can be greater than the number of available cores, then mpirun will loop to the first host in the hostfile and continue until all processes have been assigned but this just means they are oversubscribed. In such cases due to the oversubscription, the computation time will be more than the computation time obtained for the m number of processes, where m is the total number of available cores. Table 1 and Table 1 shows that computation time obtained for 9 and 10 processes, is more than the computation time for 8 slaves, where total available number of cores is 8.

Case 2: Multiple nodes analysis with higher order matrices (computation > communication). Table 1 shows that for matrix of higher order, the performance of the system increases phenomenally with increase in number of nodes. As the size of the matrix increases the computation time also increases. The computation is so large that the communication is negligible compared to it. Since the computation will be more for larger matrices, it needs more number of nodes depending upon the size of the matrix to give better performance. Hence for larger matrices, we get optimal computation time over the four nodes, as our work is limited to four nodes.

Table 1 Performance of MPI for larger order matrices (at 1GB RAM, all time in seconds)

Number of processes (1 master + slaves)	**Number of cores utilized**	**Number of nodes utilized**	**1000*1000**	**1500*1500**	**2000*2000**	**2500*2500**	**3000*3000**
2	2	**1**	10.269120	35.748371	82.646102	164.648721	286.507145
3	3	2	6.454944	20.521835	46.625184	90.571434	156.261247
4	4	2	5.73351	17.461328	38.394196	73.479432	134.337615
5	5	3	**5.281018**	**14.912146**	31.855243	58.541683	102.674381
6	6	3	5.798123	15.171543	**31.251527**	**56.816513**	**97.621531**
7	7	4	5.863169	15.349281	34.196192	70.358763	106.257814
8	8	4	5.912532	15.718843	36.192183	75.481966	113.657381
9	8	4	6.012345	15.912356	39.876512	76.012654	114.106743
10	8	4	6.874532	16.567432	4.2316741	77.654675	115.277643

6. CONCLUSION

We presented a model that demonstrates the performance gain and losses achieved through parallel processing. Matrix multiplication problem is solved in parallel under MPI. We also demonstrated the communication overhead involved in parallel computation. The results with runs on different number of nodes are compared to evaluate the efficiency of MPI based parallel applications. The total time taken to compute the result decreases drastically when the number of nodes increases.

FUTURE WORKS

Even though the method that has been used here can be deployed to solve larger order problems, it is cumbersome to give the data input for matrices of larger order. Hence this work can be extended to give input from files for larger order matrices. It can also be extended to solve other similar problems related to matrices, like finding the determinant and other backtracking problems. The analysis is also useful for making a proper recommendation to select the best algorithm related to a particular parallel application. We limited the number of nodes tofour and it can be extended.

REFERENCES

[1] Amit Chhabra, Gurvinder Singh "A Cluster Based Parallel Computing Framework (CBPCF) for Performance Evaluation of ParallelApplications", International Journals of Computer Theory and Engineering, Vol. 2, No. 2 April, 2010.

[2] Petre Anghelescu, "Parallel Algorithms for Matrix Multiplication", 2012 2nd International Conference on Future Computers in Education, Vols.23-24, 2012.

[3] Rajkumar Sharma, Priyesh Kanungo, Manohar Chandwani, "Performance Evaluation of Parallel Applications using Message Passing Interface in Ntework of Workstations of Different Computing Powers", Indian Journal of Computer Science and Engineering(IJCSE), Vol. 2,No. 2, April-May 2011.

[4] Sampath S, Sudeepa K.B, Nanjesh B R "Performance Analysis and Evaluation of Parallel Applications using a Cluster Based Parallel Computing Framework", International Journal of Computer Science and Information Technology Research Excellence (*IJCSITRE*), Vol.2,Issue 1,Jan-Feb 2012.

[5] Message Passing Interface, MPI Standard: http://www.mpi-form.org.

[6] MPICH2: A New Start for MPI Implementations, Recent Advances in Parallel Virtual Machine and Message Passing Interface, Lecture Notes in Computer Science, Volume 2474, 2002, p 7.

[7] A. Nazir, H. Liu, and S.-A. Sørensen, "On-demand resource allocation policies for computational steering support in grids, " in International Conference on High Performance Computing, Network and Communication Systems, Orlando, USA, 2007.