Volume 3, Issue 1, January 2013 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Performance Evaluation and Comparison of Message Passing Interface and Parallel Virtual Machine

Sampath S[1]
Department of C S & E
Shri Venkateshwara University

Bharat Bhushan Sagar[2]
Department of IT
Uttar Pradesh Technical University

Nanjesh B R[3]
Department of C S & E
Vishveshwariah Technological University

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 common single node architecture for both MPI and PVM, which demonstrates the performance gain and losses achieved through parallel processing using MPI and PVM as separate cases. We also demonstrates the performance dependency of parallel applications on RAM of the nodes (desktop PCs) used in parallel computing. This can be realized by implementing the parallel applications like solving matrix multiplication problem, using MPI and PVM separately. The single node architecture consists of a client, a master, capable of handling requests from the client, and a slave, capable of accepting problems from the master and sending the solution back. The master and the slave communicate with each other using MPICH-2 and PVM3.4.6 when computation is under MPI and PVM respectively. The master will monitor the progress and be able to compute and report the time taken to solve the problem, taking into account the time spent in assigning the problem into slave and sending the results along with the communication delays. . We aim to evaluate and compare these statistics of both the cases to decide which among MPI and PVM gives faster performance and also compare with the time taken to solve the same problem in serial execution to demonstrate communication overhead involved in parallel computation. We aim to compare and evaluate the statistics obtained for different sizes of RAM under parallel execution in a single node involving only two cores, where one acts as master and other as slave. We also show the dependency of serial execution on RAM for the same problem by executing its serial version under different sizes of RAM.

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

I. 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.

A. Existing Frameworks

1) MPI: The specification of the Message Passing Interface (MPI) standard 1.0 [6] 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 core 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.

2) 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 [6]. 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.

3) 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.

4) PVM 1.0: PVM that have been released from the first one in February 1991. PVM 1.0 cleaned up the specification and implementation to improve robustness and portability [7].

5) PVM 2.X Versions: PVM 2.1 provided with process-process messages switched to XDR to improve portability of source in heterogeneous environments and simple console interpreter added to master pvmd. Later versions belonging to PVM 2.x provided with more and more useful functionalities such as pvmd-pvmd message format switched to XDR, get

and put functions vectorized to improve performance, broadcast function deprecated ,improved password-less startup via rsh/rcmdetc [7].

6) PVM 3.X Versions: These allow scalability to hundreds of hosts, allow portability to multiprocessors / operating systems other than Unix, allows dynamic reconfiguration of the virtual machine, allows fault tolerance, includes dynamic process groups, provide the option to send data using a single call [7].

7) PVM 3.4.6: Includes both Windows and UNIX versions and improved use on Beowulf clusters. Also includes the latest patches for working with the latest versions of Linux (like fedora 14), Sun, and SGI systems New features in PVM 3.4.x include communication contexts, message handlers, persistent messages. In our project, we are using PVM 3.4.6 for providing parallel environment using PVM [8].

B. Frameworks used in Proposed System

This paper deals with the implementation of parallel application, matrix multiplication under MPI using MPICH-2and under PVM using PVM3.4.6 for communication between the cores and for the computation. Because they are very much suitable to implement in LINUX systems.

II. 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.

V.S Sunderam, G.A Geist, J Dongarra, R Manchek (1994) [5] describe the architecture of PVM system, and discuss its computing model, the programming interface it supports, auxiliary facilities for process groups and MPP support, and some of the internal implementation techniques employed. 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. Kamalrulnlzam Abu Bakar, Zaitul Mlirlizawati Zalnuddln (2006) [4] made the performance comparison of PVM and RPC. The comparison is done by evaluating their performances through two experiments namely one is a broadcast operation and the other are two benchmark applications, which employ prime number calculation and matrix multiplication. Sampath S, Sudeepa, Nanjesh B.R (2012) [3] presented the 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. Rafiqul Zaman Khan and Md Firoj Ali (2011) [2] represented the comparative study of MPI and PVM parallel programming tools in parallel distributed computing system they described some of the features for parallel distributed computing system with a particular focus on PVM (Parallel Virtual Machine) and MPI(Message Passing Interface) which are mostly used in today's parallel and distributed computing system. We demonstrate the performance gain and losses achieved through parallel processing using MPI and PVM as separate cases. We also demonstrates the performance dependency of parallel applications on RAM of the nodes (desktop PCs) used in parallel computing.

III. SYSTEM REQUIREMENT

A. Hardware Requirements

- Processor: Pentium D (3 G Hz)
- Two RAM: 256MB and 1GB
- Hard Disk Free Space: 5 GB

B. Software Requirements

- Operating System: Linux
- Version: Fedora Core 14
- Compiler: GCC
- Communication protocol: MPI and PVM

IV. SYSTEM DESIGN

A. System Analysis

The system is to be designed such that it demonstrates the performance dependency of parallel and serial execution on RAM and also it demonstrates the following:

- How a client can submit the entire problem to a master and collects the solution back from it without bothering about how it has been solved.
- How the master detects the available slaves on the network, and how it detects the system load on that machine to determine whether it is worth sending a task to that particular client.
- How a problem can be submitted to the slaves.
- How the solutions of the given problem can be retrieved from the slave.
- How the slaves solve the given problem.

The design was made modular i.e. the software is logically partitioned into components that perform specific functions and sub-functions.

1) *Master:* It is designed such that it has functionality to manage connection and communication with the slave, It scans and identifies all the cores or slaves available on the node here it is only one slave to be identified. It then assigns the processor ranks to identify the cores. The masterassigns the problem to slave. It also has to accept the results sent back by the slave after they finish the computation of the sub-tasks assigned to them. Then the received result has to be assembled in the right order to obtain the solution for the main problem.

2) *Slave*: It is designed to have the functionality to read the problem (in case of single slave)/sub-problem sent by the master, evaluate the problem (in case of single slave)/sub-problem and send the result back to the master.

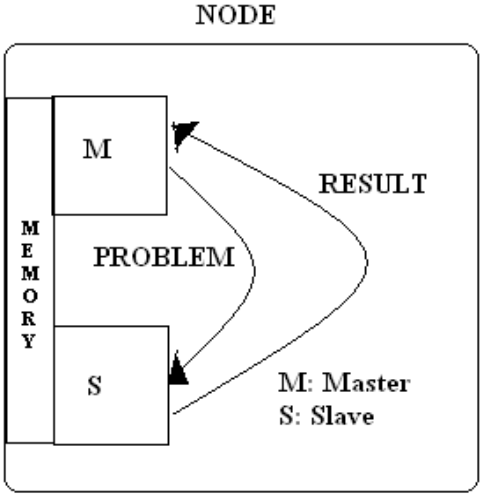

Fig 1. Architecture for presenting Parallel Computing Framework.

B. Single node architecture

The main problem is taken by the master core and assigns the task into slave core. Slave core send back the solutions of the assigned task or problem as shown in Fig. 3.

C. Configuration

1) *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, etc.):
PATH=/home/<USERNAME>/mpich2-install/bin:$PATH; export PATH

2) *PVM configuration*:

Download the software package from http://www.netlib.org/pvg3
Unpacking:
Command: $tar zxvf pvm3.4.6.tgz
Opened the .bhrc file through terminal using vi editor and set the following lines in the file [3] and closed the file.
The .bhrc is a hidden file of course and can be done as:
$home
$ls –a
$vi .bhrc
Going to the insert mode add the following lines as:
PVM_ROOT=$HOME/pvm3
PVM_DPATH= PVM_ROOT/lib/pvmd
Export PVM_ROOT PVM_DPATH
Going to pvm3 directory ($cd pvm3) type make ($make). This would make pvm(the PVm console), pvmd3(the pvm daemon), libpvm3.a(PVM C/C++ library), libfpvm3.a (PVM Fortran library) and libgpvm3.a (PVM group library). All these files would be placed in the $/pvm3/lib/LINUX and pvmgs (PVM group server) would be placed in $/pvm3/bin/LINUX. Open .rhosts file in the home directory ($ vi .rhosts) and added the name of the node (computer) in the cluster of four computers.
Set the following environment variables in .bashrc file as:
export PVM_ARCH='$PVM_ROOT/lib/pvmgetarch'
export PVM_ROOT= $HOME/pvm3/xpvm

export PATH =$PATH:$PVM_ROOT/lib
export PATH==$PATH:$PVM_ROOT/lib/$PVM_ARCH
Going to the pvm3 directory and again executed make command. If a prompt pvm> is got means pvm is successfully installed.

V. IMPLEMENTATION

Implementation is the most crucial stage in achieving a successful parallel system. The problem to be solved has to be parallelized so that computation time is reduced. The framework consists of a client, a master core, 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 MPICH-2 under MPI and PVM3.4.6 under PVM. The problem has to be divided such that the communication between the server and the client is minimum. The total computational time to solve the problem completely is effected by the communication time between the nodes.

A. 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 server 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 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 cores, 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. Consider two matrix, matrix A and B. The flow of multiplication of matrix A and B takes place as shown in Fig. 2. 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. 3.

VI. TESTING

There is a need of testing whether the resultant matrix is correct or not. Hence testing is one of the important steps that must be performed, which is explained in the following section. Separate testing is made for smaller order matrices and larger order matrices.

A. Testing of Small Order Matrices

We have tested the output of our system by verifying the output using a online matrix calculator. We have repeated this test for different smaller sizes of matrix. The tool used by us is Blue Bit online matrix multiplication calculator.

B. Testing of higher order matrices

Blue Bit online matrix multiplication calculator is limited to matrix of order 32 as the testing tool does not support the matrix multiplication of higher order. To verify our result for higher order matrix we have assigned 1 to all the elements of the matrix. The elements of the output matrix will be equal to the order of the matrix. Example is shown in Fig. 4.

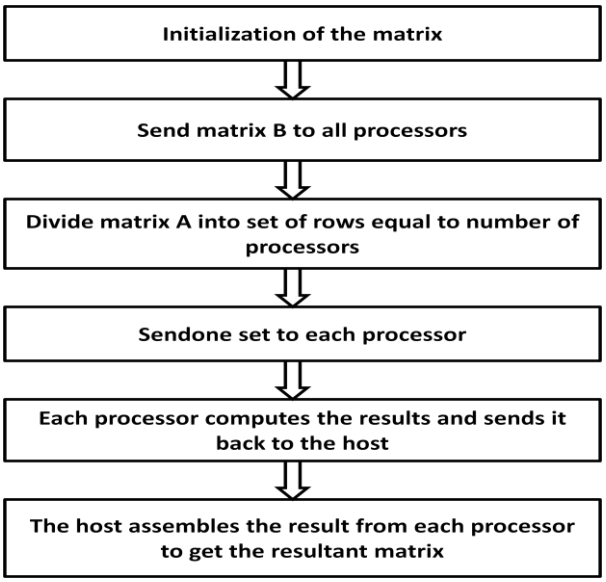

Fig.2. Flow diagram for solving matrix multiplication problem on several nodes

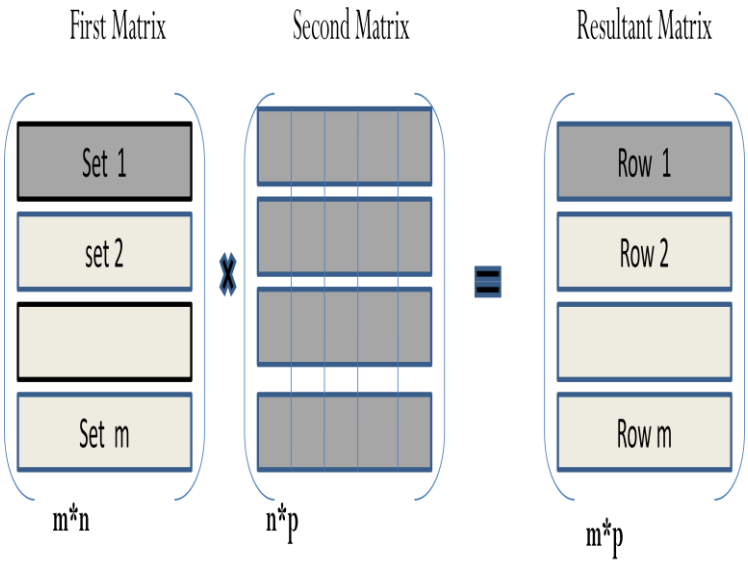

Fig. 3. Parallel matrix multiplication design

TABLE I
COMPARISON OF RESULTS OF SERIAL AND PARALLEL (MPI and PVM) EXECUTION

TYPE OF EXECUTION	MATRIX SIZE → / RAM SIZE ↓	100*100	500*500	1000*1000	1500*1500	2000*2000
SERIAL	256MB	0.0159 seconds	1.2744 seconds	9.8071 seconds	32.9938 seconds	78.1319 seconds
	1000MB	0.0157 seconds	1.2713 seconds	9.8112 seconds	32.9798 seconds	78.1526 Seconds
PARALLEL WITH MPI	256MB	0.2521 Seconds	5.4241 seconds	24.0342 seconds	78.2816 seconds	345.4321 seconds
	1000MB	0.1121 seconds	3.4212 seconds	20.1462 seconds	70.2437 Seconds	166.5418 Seconds
PARALLEL WITH PVM	256MB	0.3752 Seconds	8.6112 seconds	46.5221 seconds	769.1426 seconds	1085.2721 seconds
	1000MB	0.1951 seconds	6.8113 seconds	26.9792 seconds	76.2485 seconds	271.6431 seconds

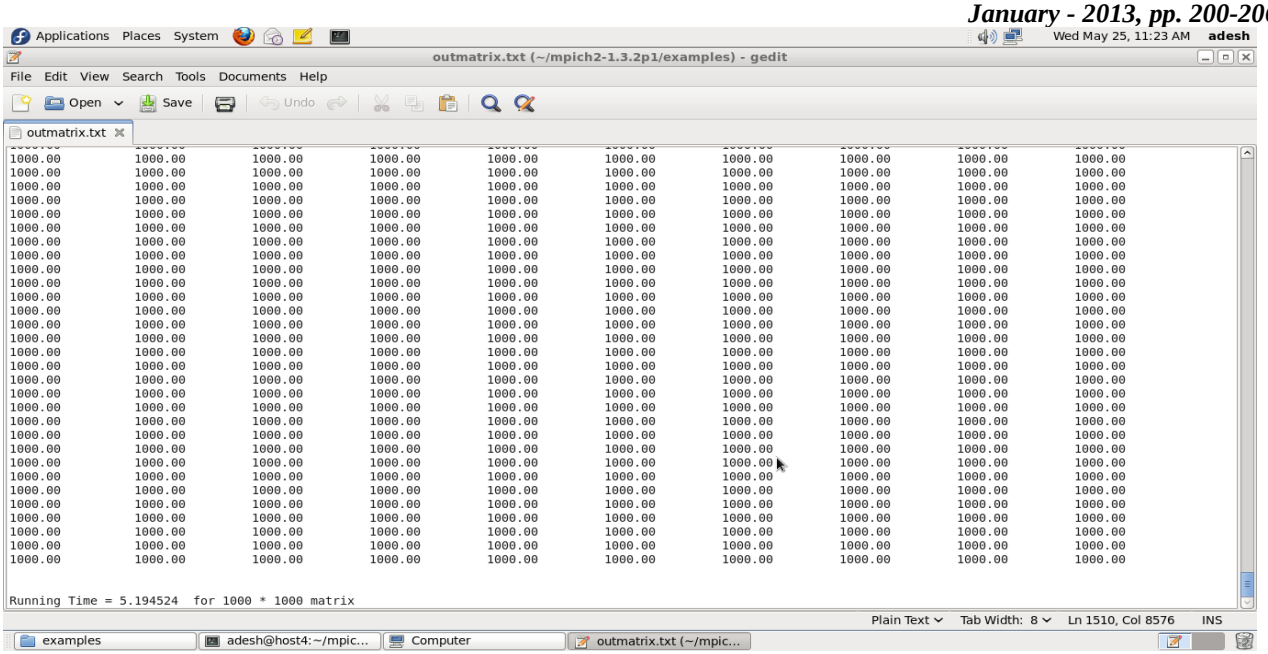

Fig 4. Output of 1000 * 1000 matrix in parallel execution

VII. RESULTS AND ANALYSIS

We have analyzed the performance of parallel method against traditional serial method. The results are tabulated and compared. We calculated the time for solving the matrix multiplication problem using both serial and two cases of parallel algorithm MPI and PVM. From the Table I, we can conclude that performance of MPI is faster than that of PVM. The table also shows that performance of serial execution almost remains same even after the increase in RAM size. There are negligible computation time variations for increase in RAM size. This is because the Serial execution is performed by the cores itself with negligible RAM usage and also due to the no communication involved between cores. Hence it is independent of RAM. We can also conclude that performance of parallel execution in both the cases MPI and PVM drastically increases when there is increase in RAM size. It shows drastic decrease in computation time with the increase in RAM. Because parallel execution often uses RAM for the communication between cores and also it involves lot of send and receive operations and temporarily storing the result of problem assigned to cores.

We can analyze that higher the size of matrices the time difference is very high in the table, because higher the matrix size, more will be sends and receives resulting in the need of higher utilization of RAM. So for smaller RAM the computation time will be more and larger the RAM size computation time will be less in parallel execution finally resulting in better performance in both MPI and PVM. However by seeing the time results for higher order matrices such as 1000*1000, 2000*2000, there is a large reduction in computation time for PVM when there is increase in RAM size. But MPI shows comparatively less reduction in computation time with increase in RAM size. Hence we can conclude that PVM is more dependent on RAM size than the MPI for the matrices of larger size.

Generally we can say parallel execution is faster than the serial execution but the results of serial execution with 1000MB RAM and parallel execution using MPI and PVM with 1000MB RAM shown in the Table I depicts that serial execution is faster than parallel execution in a single node having two cores, for different sizes of matrices. This is due to the communication overhead involved in the parallel execution but this can be overcome by increasing the number of nodes but at present it is out of scope of our work and can be done as future work. Overheads that are considered are the connection time required to connect to slave, time taken to send the problem along with inputs to slave time taken to retrieve the solutions from the client, time taken to assimilate the results obtained.

VIII. CONCLUSIONS

We presented a model that demonstrates the performance gain and losses achieved through parallel/distributed processing. Matrix multiplication problem is solved serially and also in parallel under MPI and PVM both giving result that MPI is faster than the PVM. We demonstrated the evaluation of the performance dependency of parallel applications on RAM using MPI and PVM. We also showed the dependency of serial execution on RAM for the same problem by executing its serial version under different sizes of RAM.

IX. FUTURE WORKS

We compared the results with runs on single node only. It can be extended to the more number of nodes to evaluate the performance dependency on RAM with the increase in number of nodes using same parallel matrix multiplication algorithm also to know the variations in performance of both MPI and PVM. 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. If the nodes are extended, node failure can be a problem that has to be tackled.

ACKNOWLEDGEMENT

We express our humble pranams to his holiness SRI SRI SRI Dr. BALAGANGADHARANATHA MAHA SWAMIJI and seeking his blessings.First and foremost we would like to thank Dr. C.K. Subbaraya, Principal, Adichunchangiri Institute of Technology, Chikmagalur, for his moral support towards completing our work. And also we would like to thank Dr. Suraj M.G, Dr. Mallikarjuna Bennur, for their valuable suggestions given for us throughout our work.

REFERENCES

[1] AmitChhabra, Gurvinder Singh "A Cluster Based Parallel Computing Framework (CBPCF) for Performance Evaluation of Parallel Applications", International Journal of Computer Theory and Engineering, Vol. 2, No. 2 April, 2010.
[2] Rafiqul Zaman Khan, MdFiroj Ali, "A Comparative Study on Parallel Programming Tools in Parallel Distributed Computing System: MPI and PVM", Proceedings of the 5th National Conference; INDIACom-2011.
[3] 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.
[4] Kamalrulnlzam Abu Bakar, ZaitulMlirlizawati Zalnuddln, "Parallel Virtual Machine versus Remote Procedure Calls: A Performance Comparison", JilidIS.Bit. I, JumalTeknologiMaklumat, Jun 2006.
[5] V.S Sunderam, G.A Geist, J Dongarra, R Manchek, "PVM Concurrent Computing System Evolution, Experiences and Trends", Parallel Computing - Special issue: message passing interfaces archive Volume 20 Issue 4 Pages 531-545, April 1994.
[6] Message Passing Interface, MPI Standard: http://www.mpi-form.org.
[7] History of PVM versions: http://www.netlib.org/pvm3/book/node156.html.
[8] PVM3.4.6: http://www.csm.ornl.gov/pvm/pvm3.4.6.

AUTHORS PROFILE

Sampath S is a Research Scholar in Computer Science & Engineering at Shri Venkateshwara University, Gajraula, U.P, and India. His fields of interest are High Performance Computing & Cloud Computing .He obtained his B.E degree from Kuvempu University, Karnataka, India and M.Tech degree from Visvesvaraya Technological University. Karnataka, India. Currently he is working as Associate Professor and HOD in Department of Information Science and Engineering, at Adichunchanagiri Institute of Technology, Chikmagalur, Karnataka, India.

Dr. Bharat Bhushan Sagar received his Master of Computer Applications degree from the Uttar Pradesh Technical University. He obtained his Ph.D in S.E. His research areas are Parallel Computing, Cloud Computing, Software Engineering . He also received PGDM-IRPM. Currently he is working as Associate Professor in the Department of Information Technology Ajay Kumar Garg Engineering College, Gaziabad, U.P, India.

Nanjesh B.R is doing M.Tech in Computer Science and Engineering branch, Adichunchanagiri Institute of Technology, Chikmagalur, Karnataka, India. He obtained his B.E from A.I.T. college Chikmagalur under Vishveshvaraia Technological University. His fields of interest are image processing and parallel computing. He has undertaken the comparison of MPI and PVM as M.Tech project

