Determinism Modeling of PVM's Non Deterministic Actions

Sampath S, Bharat Bhushan Sagar

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 by taking advantage of non-local resources and overcoming memory constraints. Parallel computing environment is realized using cluster based parallel computing architecture based system. MPI and PVM are the major tools used for establishing parallel computing environment. The performance of parallel application mainly depends on the methodology used for assignment of tasks. MPI follows the deterministic nature for assigning the tasks whereas the PVM follows the non deterministic nature which makes the PVM performance slower than that of MPI. In our work, we analyze the deterministic nature of MPI and non deterministic nature of PVM. This is realized by implementing matrix multiplication using both MPI and PVM. The computation is performed for different sizes of matrices over the different number of nodes. The results are compared with the computation time obtained for executions with MPI and PVM. Our work also focuses on the analysis of performance variations when the non deterministic nature for assignment of tasks of PVM is made deterministic forcibly.

Keywords—Parallel Execution, Cluster Computing, MPI (Message Passing Interface), PVM (Parallel Virtual Machine).

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. MPI and PVM are the major parallel programming tools.

Message Passing Interface (MPI)

MPI (Message Passing Interface) is specification for message-passing libraries that can be used for writing portable parallel programs. In MPI programming, a fixed set of processes is created at program initialization. Each process knows its personal number. Each process knows number of all processes and they can communicate with other processes. Process cannot create new processes and the group of processes is static.

MPI provides an interface for the basic user, yet is powerful enough to allow programmers to use the high performance message passing operations available on advanced machine. When a MPI-program is executed on a master process, the function MPI_Init() initializes the execution environment. It opens a local socket and binds it toa port and verifies that communication can be established with other processes. Then, it distributes MPI internal state to each task so that they start execution. Each task will execute its own portion of code and when it finishes, it will call MPI_Finalize() which closes the communication library and terminates the service. Both MPI_Init() and MPI_Finalize() must be executed by each task. MPI has several kinds of send and receive functions to facilitate many applications requirements. There are many versions of MPI, these are detailed in the Appendix part. In our project, we are using MPICH2 for providing parallel environment using MPI [20].

- *Sampath S, B.E, M.Tech is a Research Scholar in Department of CS & E, Shri Venkateshwara University, Gajraula, Uttarpradesh, INDIA*
- *Bharat Bhushan Sagar Ph.D working as an Assistant Professor in Department of CS & E, Birla Institute of Technology, Noida, Uttarpradesh, INDIA*

Parallel virtual machine (PVM)

PVM (Parallel Virtual Machine) is a software package that allows a heterogeneous collection of workstations (host pool) to function as a single high performance parallel virtual machine.

The PVM system consists of the daemon (or pvmd), the console process and the interface library routines. One daemon process resides on each constituent machine of the virtual machine. Daemons are started when the user starts PVM by specifying a host file, or by adding hosts using the PVM console. The PVM console is the interface for users to interact with the PVM environment. The console can be started on any machine of the virtual machine. Using the console, a user can monitor the status of the PVM environment or reconfigure it. The final component, the PVM interface library, has routine for message-passing, spawning of application processes and coordination of these processes.

The daemons in PVM play a pivotal role. Besides being responsible for spawning new processes and fault-detection, a daemon is also the message router. All communication between applications processes will normally be brokered by the daemons since by default, each application process can only communicate directly with the local daemon. Each daemon maintains information concerning the location and status of all application processes in the virtual machine. When a message for a route process is received from a local application process, the local daemon determines the physical location of that remote process and forwards the message to the remote daemon. Both UDP and TCP sockets are used in PVM. There are many versions of PVM; these are detailed in the Appendix part. In our project, we are using PVM3.4.6 for providing parallel environment using MPI [20].

Deterministic communicator

In MPI, communicator is an ordered set of nodes. Each node in the communicator has a rank and ranks are strictly defined and

not exchanged. Where as in PVM there is no ordering, the order may change dynamically.

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.

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. Hai Jin, Rajkumar Buyya, Mark Baker (2001) [13] discussed the incentive for using clusters as well as the technologies available for building clusters. and also discussed a number of Linux-based tools such as MPI, PVM etc. and utilities for building clusters.

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. Dorta, Leon, Rodiguez (2003) [11] presented the comparison between MPI and OpenMP. They clearly described and compared two parallel implementations of branch-and-bound skeletons using the C++ programming language.
Rajkumar Sharma, Priyesh Kanungo, Manohar Chandwani (2011) [12] evaluated performance of parallel applications using MPI on cluster of nodes having different computing powers in terms of hardware attributes/parameters. 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 and MPI which are mostly used in today's parallel and distributed computing system. Diego Mostaccio, Christianne Dalforno, Remo Suppi and Emilio Luque (2006) [9] carried out the distributed simulation for high performance models into a very useful and low-cost tool. The two distributed simulator have been developed based on PVM and MPI communication libraries. This work resumed the advantages and drawbacks of each implementation and some conclusions about the distributed simulation for this type of models are extracted. Cirtek P, Racek S (2007) [10] made the performance comparison of distributed simulation using PVM and MPI in which presented the possibilities of the simulation programs speedup using parallel processing and compared the results from an example experiments. In this work their application to an individual oriented model is analyzed. Our work also focuses on the analysis of performance variations when the non deterministic nature for assignment of tasks of PVM is made deterministic forcibly.

3 SYSTEM REQUIREMENT

A. Hardware Requirements

- Processor: Pentium D (3 G Hz)
- Two RAM: 256MB and 1GB
- Hard Disk Free Space: 5 GB
- Network: TCP/IP LAN using switches or hubs

B. Software Requirements

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

4 SYSTEM DESIGN

To make our analysis, we need to design cluster based parallel computing architecture, matrix multiplication common for both MPI and PVM

Cluster based parallel computing architecture
Cluster based parallel computing architecture involving three nodes over which MPI and PVM based parallel applications can run. Desktop PC's are termed here as nodes which are connected together using Ethernet TCP/IP LAN to work as single high performance computing system. Each node contains two cores.

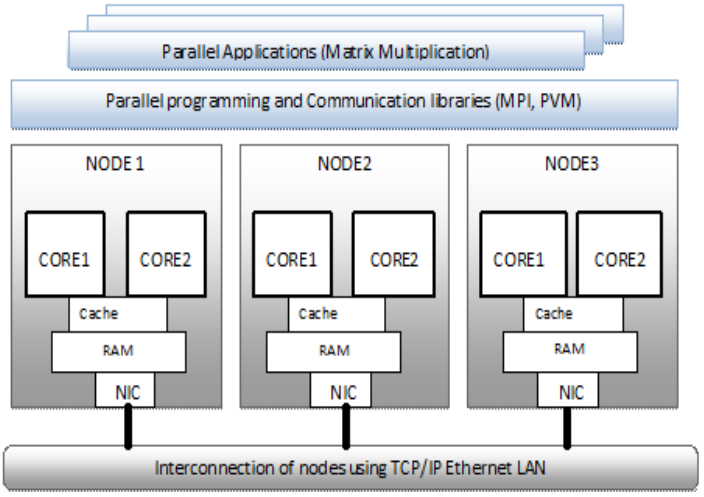

Fig1: Cluster based parallel computing architecture

Simple Matrix multiplication design

Using the capacity of underlying nodes, the processes perform the parallel computation. One of the processes acts as master and remaining processes acts as slaves. For each process unique task ids or number will be generated for identifying processes in the communication world. Message tags and offset are used for the proper arrangement of solutions by a master.

Matrix multiplication design

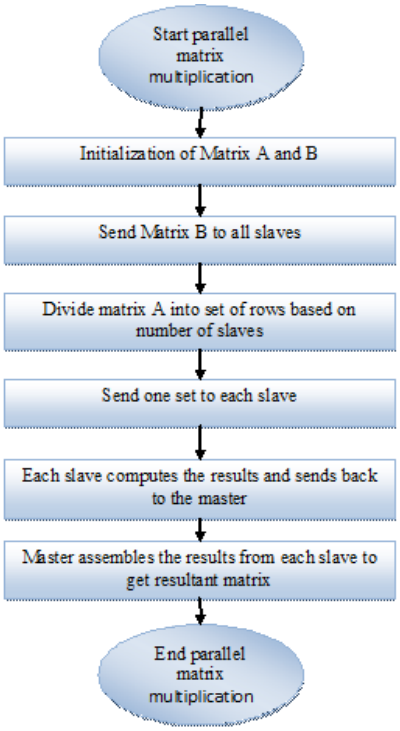

Fig2: Parallel matrix multiplication

5. IMPLEMENTATION

While spawning the new processes in PVM, it is forcibly made to act on particular nodes only

pvm_spawn - Starts new PVM processes

pvm_spawn(char *task, char **argv, int flag, char *where, int ntask, int *tids).

1)*task is the executable slave file name.
2) argv Pointer to an array of arguments to the executable
3) flag: Spawning operation and it can be
PvmTaskDefault 0
PVM can choose any machine to start task
PvmTaskHost
1 *where* specifies a particular host
PvmTaskArch
2 *where* specifies a type of architecture PvmTaskDebug 4 Start up processes under debugger PvmTaskTrace 8 Processes will generate PVM trace data.
4) where string specifies where to start the PVM process. That is host name.
5)ntask specifies the number of copies of the slaves to be started
6)tids unique identification for the tasks spawnwd.
Below code shows the general non deterministic approach of assigning tasks.

```
for (i=0; i<NPROC; i++)
 {
    pvm_spawn("nslave", NULL, PvmTaskDefault, "", 1, &wtids[i]);
}
```

Where NPROC is the number of slaves specified by the user, Here PvmTaskDefault is specified which means pvm can start at any host. Instead of this we can make deterministic so that particular processes should run over the specified host only as in MPI.
Since MPI gives complete deterministic utilization of all the 6 cores(3 nodes) with 6 processes, we assume that number of slaves as 5 with one master

Considered three hosts: host1, host2, host3.
The code can be modified as below.

```
for (i=0; i<NPROC; i++)
 {
if(i==0)
          pvm_spawn("nslave", NULL, PvmTaskHost, "host1", 1, &wtids[i]);
else if(i==1||i==3)
          pvm_spawn("nslave", NULL, PvmTaskHost, "host2", 1, &wtids[i]);
else
          pvm_spawn("nslave", NULL, PvmTaskHost, "host3", 1, &wtids[i]);

}
```

Master and first slave are allowed to act over host1, second and third slaves are allowed to act over host2, fourth and fifth slave are allowed to act over host3. That is PVM is made forcible deterministic. It can be shown as below:

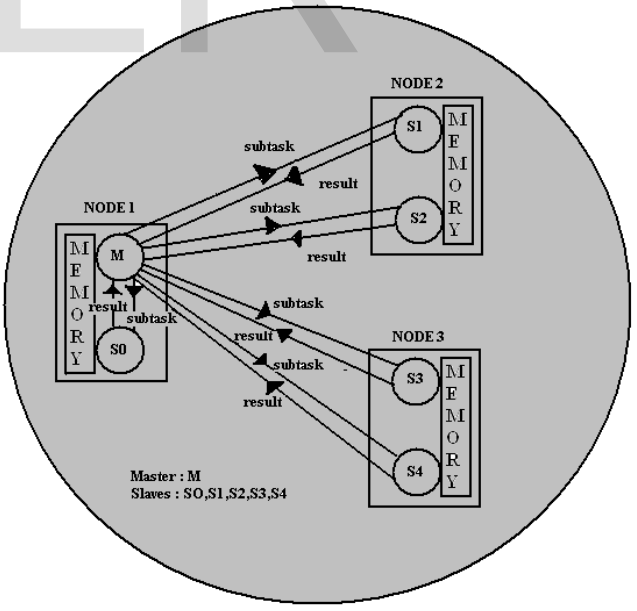

Fig 3. Determinism Modeling of PVM

6 Results and Discussion

Table 1: Performance of parallel execution for smaller order matrices

(At 1GB RAM, all time in seconds)

Matrix size → / Execution type ↓	250 × 250	350 × 350	450 × 450	550 × 550	650 × 650
Parallel on three Nodes with MPI	0.201018	0.420166	0.775246	1.254682	1.820354
Parallel on three Nodes with Non Deterministic PVM	0.428131	0.683427	0.985176	2.234166	3.164312
Parallel on three Nodes with Forcibly Deterministic PVM	0.381243	0.597642	0.812437	1.986542	2.901764

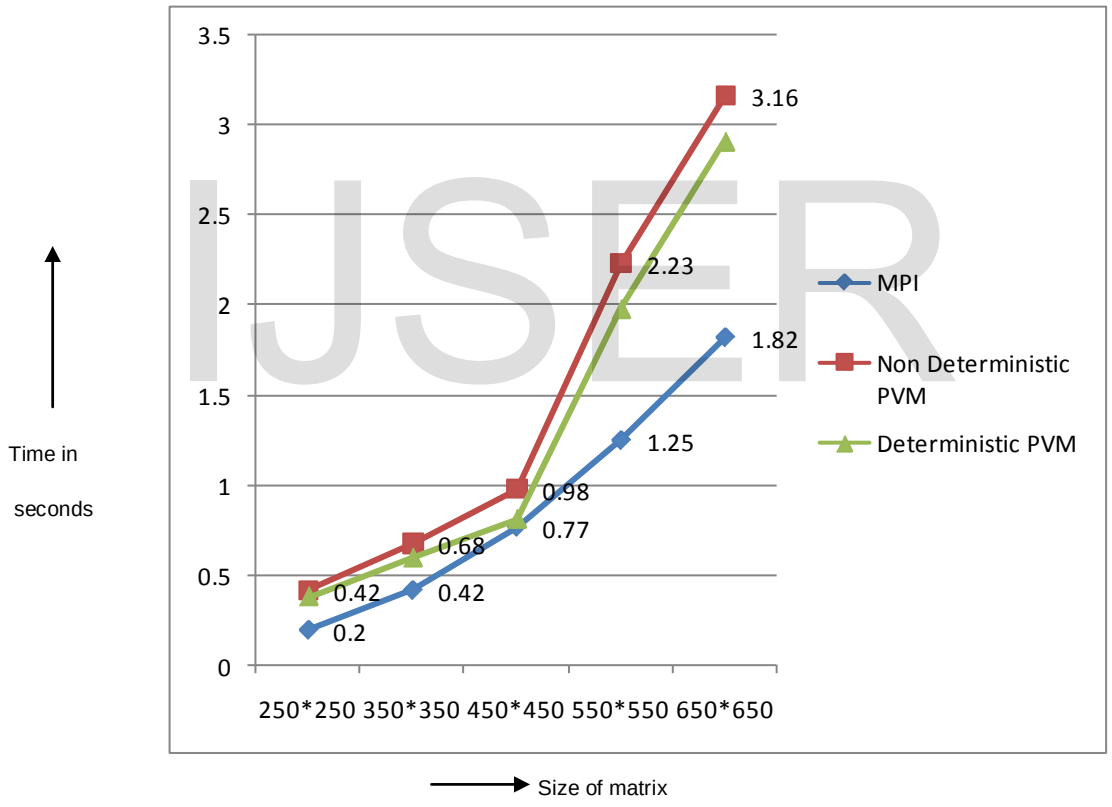

Fig 4: Comparison of MPI and PVM Non deterministic and PVM deterministic performance for smaller matrices over three nodes.

Table 2: Performance of parallel execution for larger order matrices
(At 1GB RAM, all time in seconds)

Matrix size → / Execution type ▼	1000 × 1000	1500 × 1500	2000 × 2000	2500 × 2500	3000 × 3000
Parallel on three Nodes with MPI	5.281018	14.912146	31.251527	56.816513	97.621531
Parallel on three with Non istic PVM	7.664543	19.127211	36.281561	66.958763	108.257814
on three th Forcibly istic PVM	6.128321	16.653723	34.293425	61.564372	102.453627

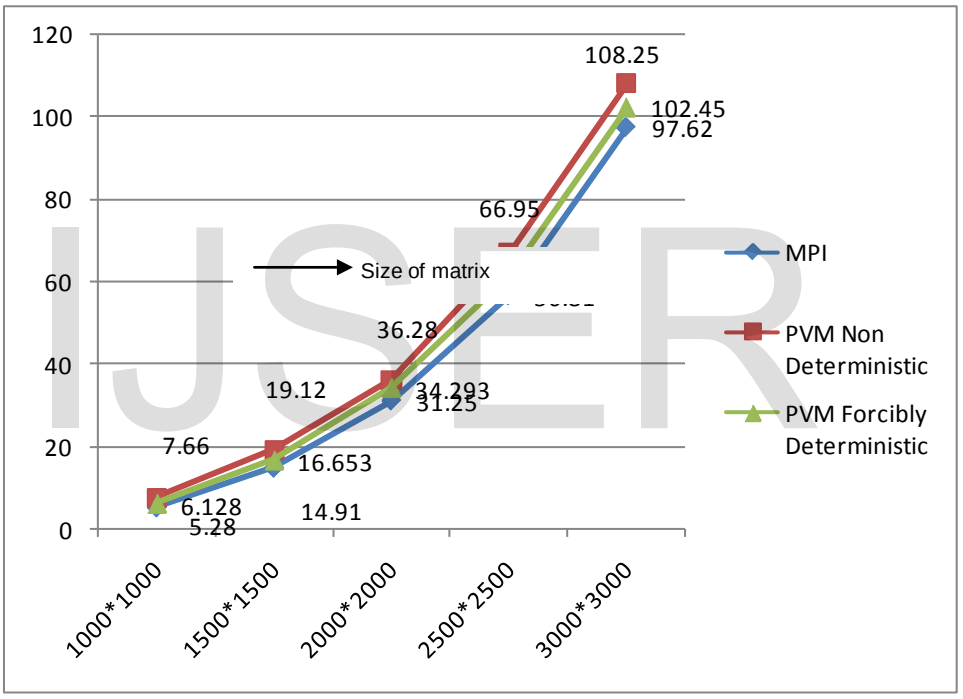

Fig 5: Comparison of MPI and PVM Non deterministic and PVM deterministicperformance for larger matrices over three nodes

7 RESULTS AND CONCLUSION

We have analyzed the performance of parallel execution with MPI, non deterministic PVM and forcibly deterministic PVM. The results are tabulated and analyzed. It shows that forcibly deterministic PVM shows higher performance than non deterministic PVM. However MPI is faster than both the types of PVM. The performance of forcibly deterministic PVM lies between non deterministic PVM and MPI.

8 FUTURE WORKS

The analysis made by fixing only three nodes where the processes are fixed to 6 processes including one master and five slaves, in future it can be extended to more number of nodes with different number of processes.

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] RafiqulZaman 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 CBPCF", International Journal of Computer Science and Information Technology Research Excellence (IJCSITRE), Vol.2,Issue 1,Jan-

Feb 2012.

[4] Kamalrulnlzam Abu Bakar, ZaitulMlirlizawatiZalnuddln, "Parallel Virtual Machine versus Remote Procedure Calls: A Performance Comparison", JilidIS.Bit. I, JumalTeknologiMaklumat, June 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] 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.

[9] Diego Mostaccio, Christianne Dalforno, Remo Suppi 1 and Emilio Luque , "Distributed Simulation of Large-Scale Individual Oriented Models", JCS&T Vol. 6 No. 2 October 2006.

[10] Cirtek P, Racek S, "Performance Comparison of Distributed Simulation using PVM and MPI", The International Conference on "Computer as a Tool". Page(s)**:** 2238 – 2241, EUROCON, 2007.

[11] Dorta, Leon, Rodiguez "A comparison between MPI and OpenMP branch-and-bound skeletons", Parallel and Distributed Processing Symposium 2003 International Proceedings, April 2003.

[12] 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.

[13] Hai Jin, Rajkumar Buyya, Mark Baker, "Cluster Computing Tools, Applications, and Australian Initiatives for Low Cost Supercomputing", MONITOR Magazine, The Institution of Engineers Australia , Volume 25, No 4, Dec.2000-Feb 2001.

[14] 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.