ISSN:0975-9646

Performance Analysis of Parallel Programming Tools

Md Firoj Ali, Rafiqul Zaman Khan
Department of Computer Science
Aligarh Muslim University
Aligarh-202002, India.

Abstract—**Numerous parallel programming tools have been developed so far for supporting parallel programs. This paper presents performance analysis of wide range of parallel programming simulation tools. This paper also compares the features of different tools. PVM and MPI are most widely used standards for parallel and distributed computing. MPI has better performance in high performance massively parallel processing (MMPs) computer systems to provide highly optimized and efficient implementations than PVM. In MMP, all of the processing elements are connected together to be one very large computer. This is in contrast to the distributed computing where massive numbers of separate computers, connected through a network, are used to solve a single large problem. PVM is most suitable in heterogeneous networks to gain optimal performance. One may favor the other tools depending on the need. With the help of our performance comparison one can choose which one would be the better for a particular application.**

Keywords—**Heterogeneous computing, virtual machine, massively parallel processor.**

I. INTRODUCTION

Computational power is always a thrust area from the early days of computer systems. Of course technology has gained its considerable progress in processing power, data storage capacity, circuit integration scale etc. in last several few years but still it becomes unsatisfactory for some scientific computations for today’s applications. So the development of high speed computers becomes one of the driving forces for that. Since many applications now a days need huge computational power, there must be a solution with low cost and high desirable performance. A costly approach for this solution is supercomputer which may be out of budget for many institutions or organizations. So parallel computing came into existence and is a very successful way of increasing desirable computation speed. A collection of workstations can be the computational equivalent of a super computer. The computer networks become the ideal platforms for the parallel computing recently and this type of computations is known as network or heterogeneous computing [18].

Distributed heterogeneous computing may be defined as a particular form of parallel computing in which each computing task is processed on the most appropriate computing framework available. A large task consists several numbers of small manageable tasks which can be concurrently executed on the most suitable framework to increase the efficiency. The degree of parallelism depends upon the parallel computer architecture.

Network computing interconnects different heterogeneous systems into a single unified computing resource [2, 3, 5]. The network which is designed for the high computation should have communication potential that equivalent to the logical computational model of the application [3, 25]. The network designed for the purpose of high computing performance should involve the following criteria:

a) scalable cumulative power [3, 18],
b) decreased inter-process communications,
c) pre-requisition for multiple logical communication channels with guaranteed bandwidths,
d) support for flexible application-dependent virtual topologies [3, 25, 26, 27],
e) support of easy partition,
f) effective system management.

The knowledge in internal architecture and the components of the distributed system is required for the parallel programming tools [14].The communication and synchronization among the various users is the most important issue for the parallel computing environments. The communication tools implement one of the two communication techniques: message passing or shared memory. Message passing interface (MPI) and parallel virtual machine (PVM) are the most popular examples of the message passing tools [22]. So many parallel computing environments are available to support parallel execution of programs on computer networks. But which environment will be the best one? The straight forward answer of this question is very difficult. To determine which environment will be the best depends one so many factors like the domain of application, particular algorithm at hand, programmer’s proficiency as well as personal interests, ease of use, fluency and efficiency. The runtime-efficiency is only a single important factor which determines which environment would be the best one in general [19]. However, efficient utilization of the computational potential of computer networks is still an open problem and a research challenge for the future.

The advantages of parallel computing system are that it ensures the processing power of widely available various types of computer systems connected through networks because these networks are at leisure or partially idle most of the time. These are beautiful resource of efficient free processing power.

Different frameworks have different advantages as well as shortcomings. It is unrealistic that a framework that would be unanimously employed for any parallel distributed computing applications. The environments are developed depending upon the requirements to solve the problem at current hand. But PVM becomes the de-facto environment for this purpose.
Rest of this paper is organized as follows. Section 2 includes the classification of simulation tools. Section 3 and 4 enumerate the different simulation tools for parallel and distributed computing and their comparisons. Section 5 and section 6 describes MPI and PVM respectively and their comparison in section 7.

II. CLASSIFICATION OF SIMULATION TOOLS

The simulation tools for parallel and distributed systems have been classified.(Fig 1) These tools very recently developed and most of them still are under research.

III. TOOLS FOR SIMULATION OF PARALLEL AND DISTRIBUTED SYSTEMS

This section provides an overview of some of the many available tools that simulate parallel and distributed systems. Table 1 and Table 2 both describe the different simulation tools for parallel and distributed computing systems.

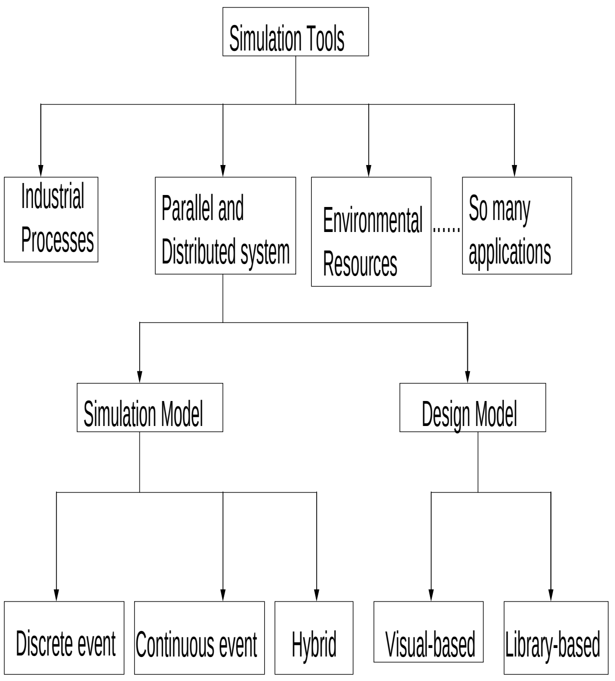

Fig.1 Shows Different Simulation Tools

TABLE I: AN OVERVIEW OF SOME THE MANY AVAILABLE TOOLS THAT SIMULATE PARALLEL AND DISTRIBUTED SYSTEMS

S. N	**Tool**	**Description**	**Language supported**	**Domain of simulation**	**Type of Simulation**	
					Library - Based	**Visual-Based**
1	Dimemas	It is a performance analysis tool for message passing programs. It provides to develop and tune parallel applications by affording accurate prediction of performance on target machine architecture. http://www.cepba.upc.es/dimemas/	C	Analyses the performance of parallel programs	Yes	Yes (using Paraver)
2	SvPablo	It is a performance analysis tool that captures and analyses data from serial and parallel programs. It rapidly identifies and resolves the performance bottlenecks. http://www.pablo.cs.uiuc.edu/Software/Pablo/pablo.html	C, ForTran9, ForTran9, HPF	Analyses the performance of parallel programs	Yes	Yes
3	Clue	It is able to simulate the performance of parallel programs using the message passing library PVM for communication. http://www.math.tuwien.ac.at/~aurora/group5/node27.html	C, JAVA etc.	Performance evaluation of message passing programs that are developed based on PVM.	Yes	No
4	Ptolemy 2	It is a Java-based library to design and simulate heterogeneous concurrent systems. http://ptolemy.eecs.berkeley.edu/ptolemyII/	JAVA	Simulates the behaviour of concurrent heterogeneous components.	Yes	Yes
5	MPISim	It predicts the performance of MPI programs based on architectural characteristics such as number of processors and message communication delays. http://pcl.cs.ucla.edu/projects/mpisim/	C	Analyses the performance of parallel programs	Yes	No

TABLE II: A COMPARISON OF SOME THE MANY AVAILABLE SIMULATION TOOLS FOR PARALLEL AND DISTRIBUTED SYSTEMS

S. N	Tool	Description	Application Area	Efficiency	Limitation
1	PACE (Performance Analysis and Characterization Environment)	Performance evaluation techniques for computing systems include: measurement, analytical modeling (that is mathematical modeling), and simulation.	The performance prediction of distributed systems	Moderate	Lack of structural information of the system
2	RSIM (Rice Simulator for ILP Multiprocessors)	RSIM is the ability to simulate processors that use instruction-level parallelism (ILP). ILP processors are capable of executing multiple instructions in parallel	simulator of cache-coherent non-uniform memory access (CCNUMA) shared-memory machines	Poor	It is not suitable for evaluation of various designs of real-world programs
3	PEVPM(Performance Evaluating Virtual Parallel Machine)	The basic idea behind PEVPM is to use statistical distributions to model the performance of the message passing operations, such as *send* and *receive*	performance modeling system for message passing programs	Moderate	it is not possible to use it for shared memory programs, or mixed message passing and shared memory programs
4	POEMS(Performance Oriented End-to-end Modeling System)	POEMS proposed a methodology for the evaluation of system model using multiple evaluation tools. The model of system is composed of component models	to develop an environment for performance modeling of parallel computing systems e	Difficult to predict	hard to estimate the efficiency
5	POSE (Parallel Object-oriented Simulation Environment)	POSE is used for simulation of the performance behavior of programs that are executed on large-scale machines such as IBM Blue Gene	a parallel discrete event simulator	Poor	not possible on a single processor machine
6	PMaC (Performance Modeling and Characterization)	PMaC approach involves the determination of the machine profile and the program signature. A machine profile comprises the information on how fast the machine can perform basic operations A program signature comprises the information on the quantity and the type of basic program operations	framework for performance prediction of message passing programs	Moderate	this technique may not be suitable for modeling computer programs
7	PAL (Performance and Architecture Laboratory)	The PAL approach expresses the execution time of a program on a machine as a parameterized mathematical model	performance modeling and prediction of distributed computing systems	Moderate	may not be suitable for the model-based performance evaluation of various program designs

IV. MPI

MPI is a common message passing library approach in which a process calls the library for the purpose of exchanging messages with another processes. MPI is a standardized interface for inter-process communication. The reason behind the design of MPI was that every Massively Parallel Processor (MPP) merchant was developing their own specific message-passing API. This is why a portable parallel application could not be written. To remove this serious problem all the vendors of MPP come to a single platform and MPI is the outcome of that. So each MPP vendor accept MPI as the standard message-passing API and as a result MPI becomes faster than PVM on MPP hosts as each vendor focuses to increase the performance of MPI [10].

The early versions of MPI provided only message passing primitives, MPI comes in a form of software library (e.g., MPICH), so the programmer can use calls to library functions to perform process management or data exchange between processes. Implementations of MPI exist for popular programming languages (e.g., Fortran, C, C++, and Java) on a variety of platforms including networks of workstations. The MPI interface is meant to provide essential virtual topology, synchronization, and communication functionality between a set of processes (that have been mapped to nodes/servers/computer instances) in a language-independent way, with language-specific syntax (bindings), plus a few language-specific features. The earlier version MPI-1 has the salient features like:

1) a huge numbers of point-to-point communication primitives
2) a huge numbers of collective communication routines among group of processors
3) a communication context that supports the design of safe parallel software libraries
4) ability to adjust with communication topologies
5) ability to generate derived data types which depict messages of non-contiguous data.

But application programs in MPI-1 were not portable across the network as there was no standard way to start MPI tasks on different nodes. So different MPI implementations used different methods. In MPI-2 version the standard process creation and start-up functions are included. The following communications functions are included in MPI-2:

1) non-blocking collective communication functions
2) language binding for C++
3) MPI_SPAWN functions to start both MPI and non-MPI processes
4) one-sided communication functions as *put* and *get*

MPI-2 is a richer source of communication functions than PVM having a total of 248 functions while MPI-1 has 128 functions only [10].

V. PVM

Parallel Virtual Machine (PVM) [8] and Message Passing Interface (MPI) [6] are the most familiar examples of the message passing systems. PVM is particularly effective for heterogeneous applications that exploit specific strengths of individual machines on a network. The individual computers may be shared- or local-memory multiprocessors, vector supercomputers, specialized graphics engines, or scalar workstations that may be interconnected by a variety of networks, such as Ethernet, FDDI, etc. PVM is an integrated software tools and libraries that is mainly designed towards networks of workstations. The central notion to the design of PVM is virtual machine concept. Virtual machine is defined as the collection of heterogeneous computers connected by a network which appears to a user as a single large computation system [10]. So using the combined speed and storage of many computers, the large computational problem can be solved with more cost effectively. The PVM system has been used for applications such as molecular dynamics simulations, superconductivity studies, distributed fractal computations, matrix algorithms, and in the classroom as the basis for teaching concurrent computing.The PVM system consists of two parts. The first part is a daemon which is known as pvmd3 and simply known as pvmd. Pvmd exists in all the computers making up virtual machine. Any user can install pvmd on a machine using a valid login [24]. A user willing to utilize PVM, first make an arrangement for a virtual machine by specifying a host-pool list. The daemons are started on each machine and co-operate to imitate a virtual machine. A machine to be member of virtual machines, it must run its own daemon. The PVM application can then be started from a command line prompt on any of these machines [16, 24]. Multiple users can build up overlapping virtual machines and each user can run several PVM applications on simultaneous basis. The PVM applications co-ordinate with the daemons via sockets and / or pipes [9]. Thus applications can be signed up into PVM and then can be monitored by it although they were not started by it [9]. Applications are free to join or leave a virtual machine at any number of times allowing them to belong several virtual machines.

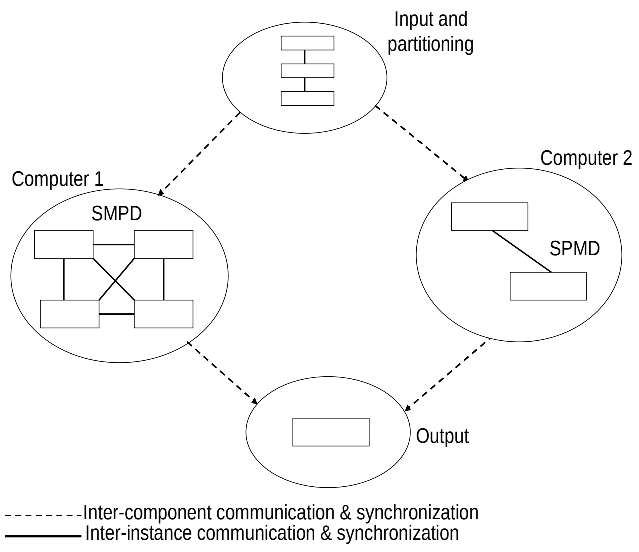
Input and partitioning
Computer 1
SMPD
Computer 2
SPMD
Output
Inter-component communication & synchronization
Inter-instance communication & synchronization

Fig. 2 PVM Computational Model

PVM supports dynamic process management while in other systems the processes are statistically defined [24]. Dynamic process groups are layered above the core PVM routines. A process can live to multiple groups, and groups can change dynamically at any time during a computation. Routines are provided for tasks to join and leave a named group. Group members are uniquely numbered from zero to the number of group members minus one. If gaps appear in this numbering due to tasks leaving the group, PVM attempts to fill these gaps with subsequently joining tasks. Tasks can also query for information about other group members. Functions that logically deal with groups of tasks such as broadcast and barrier use the users explicitly defined group names as arguments [24].

The second part of the system is a library of PVM interface routines [16, 24].This library holds the functionally complete user callable routine for message passing, spawning process, co-coordinating tasks and modifying virtual machines. Application programs for the execution must be linked to these libraries like other programming languages. Moreover, PVM programs written for different architectures can communicate to each other, thus allowing for building of heterogeneous network computing systems.

The PVM software contains a collection of protocol algorithms to implement reliable and sequenced data transfer, distributed consensus and mutual exclusion. These algorithms make the system robust by introducing error detection mechanisms and failure notification to applications. PVM uses both UDP and TCP sockets. UDP sockets are set up between a pair of daemons and between a daemon and a local task. The daemon-daemon socket is used for carrying data and inter-daemon control. When a user starts a daemon, the daemon sets up a single TCP socket with each daemon in virtual machine. These TCP carry standard-out and standard-error messages back to the user [21]. PVM requires a three-step procedure for a task to send (or receive) a message. For sending a message, a send buffer has first to be initiated, then the data is packed into the buffer, and finally the data in the buffer is sent.

Later versions e.g. (PVM 3.3) provide the option to send data using a single call [20].

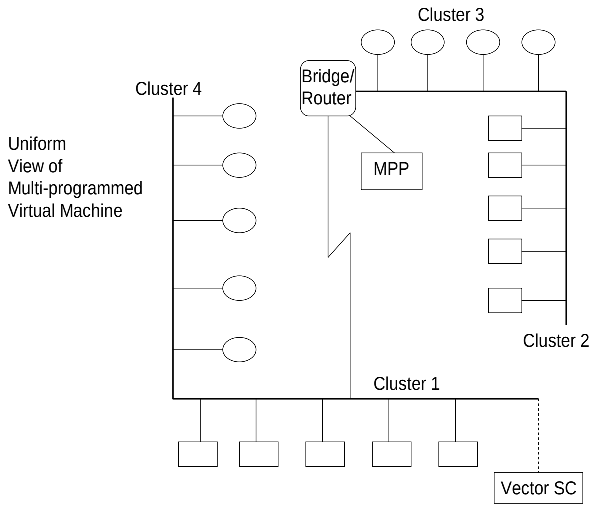

Fig. 3 PVM Architectural View

As the PVM becomes a de-facto software across the globe, it becomes mandatory to keep the PVM API backward compatible so that all existing PVM applications would continue to run as it is with newer versions.

VI. COMPARISONS BETWEEN PVM AND MPI

Now we discuss the major differences between PVM and MPI in the following sections.

A. Portability

Portability refers to the ability of the same source code which is to be copied, compiled and executed on different platforms without modifications. MPI is portable but PVM is highly portable [10, 12]. MPI is portable in the sense that MPI applications as a whole run on any single architecture and it does not able to co-operate among the different processes, running on different architectures. PVM group has done excellent work to facilitate implementations across a wide range of architectures encompassing most UNIX systems and Windows [10, 12, 23].

B. Heterogeneity

Heterogeneity refers to portability to virtual parallel machines which are obviously of different architectures. Both specifications support heterogeneity. But MPI does not mandate of that [10, 11] though it depends on the type implementations. This has a great advantage in the sense that nobody from any other vendor is allowed to use the machines of a vendor which otherwise may slowdown the systems of later one. PVM has specific functions for the support to heterogeneity. LAM [4], CHimP [1], and MPICH [13] are implementations of MPI that can run on heterogeneous networks of workstations [12].

C. Interoperability

Interoperability refers to the ability of different implementations of the same specification to exchange messages [12]. Since MPI does not mandate heterogeneity, there is no question of interoperability. PVM application programs can run across any set of different architectures and the processes can co-operate to exchange the information without any problem. PVM programs are more flexible in this sense. A separate effort (not part of the MPI Forum) has developed an “interoperability standard” called IMPI that provides sufficient standardization for some implementations details so that implementations conforming to this standard can exchange messages. IMPI is now available [7] and several vendor implementations exist [12].

Due to the lack of interoperability MPI always need to check the destination of every message whether the message is for the same host or for the other hosts. If it is for other host, the message must be converted into a format that is compatible for the other MPI version [10]. Furthermore MPI implementation uses native communication functions provided by architecture directly while PVM implementation use native communication functions during the local communication or to another host with identical architecture. But PVM uses standard communication functions for heterogeneous communications.

MPI and PVM also differ in language operability. In case of PVM, a program written in FORTRAN can send a message which can be received by a program written in C and vice-versa. But in MPI, a program written in FORTRAN does not feel to communicate with a program written in C in spite of executing on the same architecture. These two languages have two different interfaces and hence MPI does not compel two languages to interoperate [10].

D. Virtual Machine

Virtual machine is defined as the dynamic collection of heterogeneous distributed computers such as different workstations, personal computers and massively parallel computers etc, connected by a network which appears to a user as a single large computing machine [9, 10]. Virtual machine concept has brought a revolutionary change in parallel distributed computing. PVM is totally fabricated around the virtual machine concept. Virtual machine concept is the most fundamental feature of PVM [10]. This feature is the foundation for providing the facilities like portability, heterogeneity, interoperability and encapsulation of functions. On the other hand, MPI imposes attention towards message-passing and resource management and the virtual machine concept does not exist in it.

F. Topology

A topology is an extra, optional attribute that one can give to an intra-communicator; topologies cannot be added to inter-communicators. A topology can provide a convenient naming mechanism for the processes of a group (within a communicator), and additionally, may assist the runtime system in mapping the processes onto hardware. Although MPI does not support virtual machine concept, it supports a higher level abstraction on top of the computing resources in terms of massage passing topology. The MPI group of tasks can be arranged in a specific logical interconnection topology. Tasks then communicate to each other within that topology. The underlying physical network topology supports with a considerable speed for message passing [10].

G. Dynamic Process Group

Sometimes we want to perform global operations like broadcasting a message only to a subset of all the processes. MPI allows us to define a subset of these processes in run time using MPI library calls. Processes in this group is numbered from 0 to n-1 where n is the number of a processor in a group [9]. Each process ID in a group is known as rank. User processes in MPI can create new processes at runtime.

E. Contexts

A context is a system defined tag. Every message has its own tag which can be used to distinguish messages from one another. So messages of different groups never mixed when we create multiple groups with overlapping processes. Both PVM and MPI supports context in different ways. In PVM, any task can send a request to any other tasks without considering the willingness of the receiving tasks. But in MPI, a task can send a message to specific tasks which are interested to receive that message due to the presence of separate message contexts for each task [9]. PVM 3.4 has a concept of base context. In PVM all spawned tasks inherits their parent context. But if parent of any task dies, the child tasks inherit base context [10].

H. Communicators

The notions of group of processes and context are combined in a single object called a communicator [10]. Most communications are specified in terms of rank of the process in the group identified with the given communicator. Communicator variables are associated with the newly created groups in order to refer that group later as per the requirement. All communications takes place within a communicator. This has a great advantage in the sense that it provides high level of protection against irrelevant messages [9]. Communicator is the most essential part of MPI. PVM 3.4 would have been included communicators.

I. Process control

Process control refers to the ability to begin and end processes, for finding out which processes are running, and where they are possibly running [10, 16]. PVM functions provide the ability to

1) join or leave the virtual machines,
2) kill a process
3) send a signal to a process
4) check whether a process is running
5) notify a arbitrary process if another leaves from PM system.

PVM has some basic functions which are required to know how many processes can be started on the available computing resource. In the other hand MPI-1 has no defined functions that can start a parallel application. But MPI-2 is incorporated with some functions that can start a group of tasks and to send a kill signal to group of tasks [15].

J. Resource Management

PVM that is inherently dynamic in nature, can add or remove computing resources at will either from system console or even from within the user's applications. PVM Permits applications to interact with and manipulates the computing environment to provide a powerful paradigm for load balancing, task migration and fault tolerance. Virtual machine provides a framework that determines which tasks are responding and supports naming services so that independently spawned tasks can identify each other and cooperate [10, 16].

Another aspect of virtual machine dynamics relates to efficiency. Computational needs of user applications can change at the time of their program execution. So, a message-passing infrastructure should have a flexible control over the amount of computational power being exploited. For example, consider a typical application, which begins and ends with basically serial computations, but includes several phases of heavy parallel computation. A large Massively Parallel Processor (MPP) system need not be washed out as part of the virtual machine for the serial portions, and can be added just for those portions when it is of most value. Likewise, consider a long-running application in which the user occasionally wishes to attach a graphical front-end to view the computation's progress. Without virtual machine dynamics, the graphical workstation would have to be allocated during the entire computation [10]. MPI does not relate to the dynamics and it is designed to be static in nature to improve performance.

Virtual machine in PVM is responsible for encapsulating and organizing resources for parallel programs. The parallel programmer does not need to manually select every host where tasks are to be executed and then log into each machine in turn to actually spawn the tasks and monitor their execution, the virtual machine provides a simple abstraction to cover the distinct machines. Further, this resource abstraction is carefully layered to allow varying degrees of control. The arbitrary collection of machines then can be treated by the users as uniform computational nodes, in spite of their architectural disparity. Alternatively, the users are free to request for the execution of particular tasks on intended machines with particular data formats, architectures, or even on an explicitly named machine by traversing the increasing levels of detail [16].

Any abstraction for computing resources is not supported by the MPI standards and allows each MPI implementation or user to customize their personal choice of management schemes. This approach of personal choice is good but creates overheads [10].

K. Fault Tolerance

Fault tolerance is the most important and most critical issue in large scientific computing applications. Without fault tolerance some long running applications can not be completed ever. When a process is registered to virtual machine or it leaves the virtual machine or the status of the virtual machine changes, it must be notified to the virtual machine. A task can post a notify for any of the tasks from which it expect to receive a message. In this context, the receiving task will get a notify message if any task fails or expires. So getting the notice of related tasks, system can reside in a safe state. A huge loss may happen if a critical task fails just before the completion of an application at last.

In a similar fashion, if a node like an I/O server, critical to an application, fails, the application tasks can post notifies for that node. The tasks will then be informed that the server is replaced with a new one in the virtual machine. When a host exits from a virtual machine by notification to the application tasks, the application tasks adjust with the remaining available resources so that the tasks do not hang.

PVM provides more support for fault tolerance and recovery by exposing to the programmer some of the properties of sockets. MPI does less, in the interest of greater portability. Fault tolerance in MPI is an important research topic. MPI-1 does not include any fault tolerance scheme while MPI-2 includes a little of that.

L. Global Name Spaces

A database (name spaces) is created for storing the names of the processes, messages or services with the object of identifying each by its name. Processes can register or unregistered to or from the name space dynamically. Advantage is that processes can be identified independently from the underlying process management and communication environment. The dynamic nature of PVM builds name service extremely useful and convenient. MPI-1 has no functionality that does require name services [10].

VII. SUMMERY

The comparison between PVM and MPI is formulated in the table below.

TABLE III: COMPARISON BETWEEN PVM AND MPI

S.N.	Parameters	MPI-1	MPI-2	PVM
1	Portability Support	Yes	Yes	Yes
2	Heterogeneity Support	Not	Not	Yes
3	Interoperability Support	Not	Not	Yes
4	Virtual Machine Support	Not	Not	Yes
5	Topology Support	Yes	Yes	Not
6	Dynamic Process Group Support	Good	Good	Yes
7	Process Control Support	Not Defined	Yes	Fully
8	Resource Management Support	Not	Not	Yes
9	Name Spaces	Not	Yes	Yes

VIII. CONCLUSION

There are so many parallel computing tools existing so far. But problem arises that which one would be the best tool that would be unanimously accepted by every researchers. Unfortunately, there is no such universal tool, but with the help of our above performance comparison one can choose which one would be the better for a particular application.

If an application is going to be executed on a single MPP, MPI would be the most suitable option. In this case, the system performance is highly increased. MPI is very rich by the communication functions so it becomes very useful for the application that exploits special communication modes which is absent in PVM. The absence of interoperability and fault tolerance in MPI enhances the communication performance.

If an application is going to be executed in heterogeneous platform, PVM is the most suitable option. Since the PVM is built around the virtual machine concept, the applications can be executed over a collection of platforms of different hosts. PVM contains the functions like dynamic process management, resource management and fault tolerance and interoperability which are key attributes for heterogeneous computing. The ability to write long running PVM applications that can continue even when hosts or tasks fail, or loads change dynamically due to outside influence, is quite important to heterogeneous distributed computing.

Programmers should assess the functional requirements and running environment of their applications and choose the API that is most suitable accordingly.

IX. FUTURE WORK

MPI does not notify any fault description. When error occurs, MPI simply exits from the program. So notification of faults is an impetus area of future research.

Both MPI and PVM have their distinctive features. If it is possible to get a programming tool which includes the good features of the both APIs, the programmers will enjoy more freedom for developing portable programs. Further the system will utilize only MPI in intra-cluster communication and PVM in inter-cluster communication.

References

[1] Alasdair R., Bruce A., Mills J. G., and Smith A. G. CHIMP/MPI user guide. Technical Report EPCC-KTPCHIMP- V2-USER 1.2, Edinburgh Parallel Computing Centre, June 1994.

[2] Beguelin A., Dongarra J., Giest G. A., Mancheck R and Sunderam V, "Heterogeneous Network Computing", In Sixth Siam Conference on Parallel Processing, Siam, 1993.

[3] Bhattacharya S., David H.C and Pavan A., "A Network Architecture for Distributed High Performance Heterogeneous Computing ". IEEE, 1994, 110-115.

[4] Burns G., Daoud R., and Vaigl J. LAM: An open cluster environment for MPI. In J. W. Ross, editor, Proceedings of Supercomputing Symposium '94, pages 379–386. University of Toronto, 1994.

[5] Casas J., Konuru R., Otto S. W., Prouty R. and Walpole J.,"Adaptive Load Migration Systems for PVM", IEEE, 1994.

[6] Clark D. et al. "Strategic Directions in Networks and Telecommunications", ACM Computing Surveys, Vol. 28, No. 4, December 1996.

[7] Committee I. S.. IMPI - interoperable message-passing interface, 1998. http://impi.nist.gov/IMPI/.

[8] Dongarra J. et al. "An introduction to the MPI standard". Technical Report CS-95-274, University of Tennessee, January 1995.

[9] Fagg G. E and Dongarra J. J., "PVMPI: An integration of the PVM and MPI systems", Calculateurs Paralleles, 2, 1996.

[10] Geist G. A., Kohl J. A., and Papadopoulos P. M., "PVM and MPI: A Comparison of Features", May 30, 1996. Available: http://www.ece.rutgers.edu/~parashar/ classes/98 99/ece566/slides/pvmvsmpi.pdf.

[11] Gropp W. and Lusk E. "PVM and MPI Are Completely Different". CiteSeer$^{X}_{beta}$, 1997.

[12] Gropp W. and Lusk E. "Goals guiding design : PVM and MPI". Conference: Cluster 2002, IEEE International Conference on Cluster Computing, Chicago, IL (US), July, 2002.

[13] Gropp W., Lusk E., Doss N., and Skjellum A.. A high performance, portable implementation of the MPI Message-Passing Interface standard. Parallel Computing, 22(6):789– 828, 1996.

[14] Hariri S., Topcuoglu H., Furmanski W., Kim D., Ra I., Bing X., Ye B., Valente J. "A Problem Solving Environment for Network Computing". IEEE, 1997.

[15] Hempel R., Walker D. W."The emergence of the MPI message passing standard for parallel computing". Computer Standards & Interfaces 21, 51–62, 1999.

[16] Hussain j. s. and ahmed g. "a comparative study and analysis of PVM and MPI for parallel and distributed systems". IEEE, 2005.

[17] Kaliher **C. "**Cooperative processes software (CPS)", AIP Conf. Proc. August volume 209, pp 364-371, 1990.

[18] Khoussainov R., Patel A., Voorde H. D. J. T. "Distributed Parallel Computing in Networks of Workstations - A Survey Study". CiteSeer$^{X}_{beta}$.

[19] Mattson T. G. "Programming Environments for Parallel Computing: A Comparison of CPS", LINDA, p4, PVM,, POSYBL and TCGMSG. IEEE, Jan, 1994.

[20] PVM 3.3 User's guide and reference manual, Oak Ridge National Lab., Oak Ridge, Tennessee.

[21] Qureshi K. and Rashid H. "A Performance Evaluation of RPC, Java RMI, MPI and PVM". Malaysian Journal of Computer Science, Vol. 18 No. 2, December 2005, pp. 38-44.

[22] Rafiqul Zaman Khan, A. Q. Ansari and Kalim Qureshi "Performance Prediction for Parallel Scientific Application", Malaysian Journal of Computer Science, Vol. 17 No. 1, June 2004, pp 65-73.

[23] Scott S. L., Fischer M., and Geist A. "PVM on windows and NT clusters". In Alexandrov V.and Dongarra J., editors, recent advances in Parallel Virtual Machine and MessagePassing Interface, volume 1497 of Lecture Notes in Computer Science, pages 231–238. Springer, 1998.

[24] Sunderam V. S. "PVM: a parallel framework for parallel distributed computing". 1990.

[25] Vetter R. and Du D., "Distributed Computing in an Environment Based on High Speed Optical Networks", IEEE Computer, Feb., 1993.

[26] Vetter R., William K. and Du D.,"Topological Design of Optically Switched WDM Networks", Proc. 16th Conf. on Local Computer Networks, 1991.

[27] William K. and Du D., "Time and Wavelength Division Multiplexed Architectures for Optical Passive Star Networks", Univ. of Minnesota Tech. Report (TR 91-44), Sep 1991.