Volume 3, Issue 9, September 2013 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Analysis of Parallel Programming tools in Parallel Environment

Harvinder Singh[*]
Research Scholar, PTU Jalandhar, Punjab, Student Member, IEEE, India

Dr. Gurdev Singh
Professor, Gurukul Vidyapeeth, Ramnagar, Banur, Patiala, India

Abstract— This paper survey of parallel programming tools used for evaluation and optimization computation power in parallel environment . PVM and MPI are most widely used standards for parallel computing. PVM and MPI are both message passing libraries used to write portable parallel programs. PVM is most suitable in heterogeneous networks to gain optimal performance. MPI provide highly optimized and efficient implementation in parallel environment. The lack of standard in programming languages makes parallel programs difficult to port across parallel computers. These programming tools provide high performance optimized by poor performance in parallel environment.

Keywords— Parallel programming, Parallel environment, Computation power, Process optimization, PVM, MPI

I. Introduction

The demand for faster and more efficient computation power has grown in the recent times. In currently many applications need large computational power with low cost and high desirable performance. Parallel computing with very successful and desirable speed become ideal platform. A large problem split with numbers of small manageable tasks executed concurrently to increase the efficiency. Parallel processing is a technique of executing multiple tasks simultaneously on multiple processors. Currently different types of high performance machines based on the integration on many processors are available. Parallel programming tools typically provide support for developing programs composed by several processors that exchange information during execution. The performance of systems developed with a programming tools and the available support for communication and synchronization among the various components of a parallel program are some important issues. The programming tools should match the programming model underlying the parallel algorithm to be implemented. Performance tools can help a programmer to understand and improve parallel programs by monitoring the execution of a parallel program and producing performance data that can be analyzed to locate and understand areas of poor performance.

Many parallel programming tools are available for the implementing of parallel programs depends on the problems type. The choice of parallel programming tools to be used depends on the characteristics of each problem to be solved The lack of standard in programming tools makes parallel programs difficult to port across parallel computers. Parallel programming libraries PVM and MPI are most popular message passing tools in parallel environment.

In all parallel processing data must be exchanged between cooperating tasks. Message passing libraries have made it possible to map parallel algorithm onto parallel computing platform in a portable way. PVM and MPI are most widely used standards for parallel computing. MPI is now a well accepted standard that is widely used. PVM is very stable and also widely used. PVM is most suitable for heterogeneous environment to gain optimal performance. MPI standards has been widely implemented and is used nearly everywhere, attesting to the context to which these goals were achieved. PVM was aimed at providing a portable, heterogeneous environment for using cluster of machines using socket communication over TCP/IP as a parallel computer. So it is rather important to compare these systems in order to understand under which situation one system of programming might be favored over another, when one is more preferable than another.

II. MPI

MPI is a common message passing library approach in which a process calls the library of exchanging messages with another processes. MPI is a proposal for the standardization of a message passing interface for distributed memory parallel machines. The aim of this standard is to enable program portability among different parallel machines. it does not handle issues such as parallel program structuring, and debugging. It does not provide any definition for fault tolerance support and assumes that the computing environment is reliable.

The original MPI standard published in 1994, is renamed MPI-1. The early version of MPI provided only message passing primitives. MPI comes in a form software library ***mpich*** was developed at the Argonne National Laboratory based on the MPI 1.1 standard. ***LAM*** is another package based on the MPI 2.0 standard and developed at the Laboratory for Scientific Computing of the University of Notre Dame. This implementation provides some monitoring functions, which are very helpful for managing processes running on the physical machines.MPI-1.were not portable across the network as there was no standard way to start MPI tasks on different nodes. MPI-2 has added many features dynamic process, fault tolerance, efficient use resources and load balancing. MPI 3.0 standard added a new group-collective communicator creation routine that is collective over only those processes that will be members of the new communicator. Group-collective communicator creation relaxes the synchronization involved in team formation, enabling the development of composite applications that dynamically generate teams to tackle individual computational tasks. Three communication modes are available: ready (a process can only send a message if there is a corresponding reception operation initiated); standard (a message can be sent even if there is no reception operation for it), and synchronous (similar to the standard mode, but with a sending operation considered as completed only after its destination processor initiates a reception operation). The selection of the mode to be used depends on the type of communication needs of the application and can have a great influence in the efficiency of the parallel program.

MPI library functions include point-to-point send/receive operations, logical process topology, exchanging data between process pairs (send/receive operations), synchronizing nodes (barrier operation) as well as obtaining network-related information such as the number of processes in the computing session, current processor identity that a process is mapped to, neighboring processes accessible in a logical topology, and so on.

MPI library routines MPI_Send and MPI_Recv are used in order to send/receive a message to/from a process. It has two parts namely, the contents of the message (message buffer-the letter itself) and the destination of the message (message envelope-what is on the letter/envelope). The MPI_Send routine has six parameters. The first three parameters specify the message address, message count and message datatype. The last three parameters specify the destination process id of the message, tag and communicator.

Point-to-point operations come in synchronous, asynchronous, buffered, and *ready* forms, to allow both relatively stronger and weaker semantics for the synchronization aspects of a rendezvous-send. Many outstanding operations are possible in asynchronous mode, in most implementations. MPI provides both blocking and non-blocking send and receive operations and non-blocking versions. In blocking send/receive operation does not return until the message buffer can be safely written /read while a non-blocking send/receive mode can return immediately. MPI also has multiple communication modes.

TABLE I
Different Send/Receive Operations In MPI

MPI Premitives	*Blocking*	*Nonblocking*
Standard send	MPI_Send	MPI_Isend
Synchronous send	MPI_Ssend	MPI_Issend
Buffered send	MPI_Bsend	MPI_Ibsend
Ready send	MPI_Rsend	MPI_Irsend
Receive	MPI_Recv	MPI_Irecv
Completion check	MPI_Wait	MPI_Test

MPI provides number of ***collective communication*** routines. ***Broadcast*** (sending the same data from one member of the group to all other), ***Gather*** (sending data from all members to one specific member), ***Scatter*** (sending different data from one member to all others), ***Allgather*** (sending data from all members to all members), ***Alltoall*** (sending different data from each member to all others.

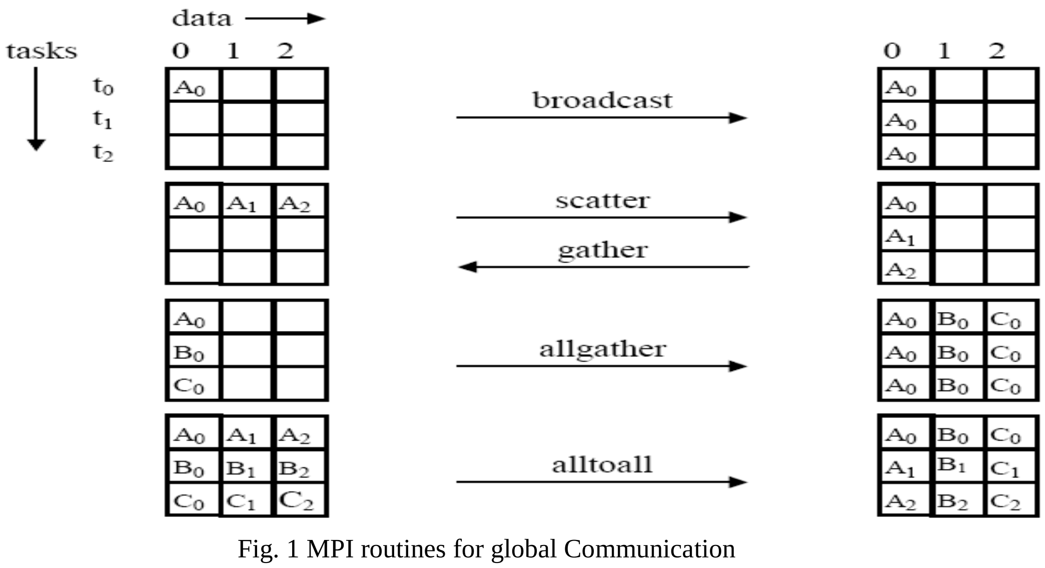

Fig. 1 MPI routines for global Communication

The MPI standard defines the syntax and semantics of a core of library routines useful to a wide range of users writing portable message passing programs in Fortran and C.Many implementations of the MPI standard are available. MPI exist for popular programming languages (Fortran, C, C++ and Java) on a variety of platforms including network of workstations. It also run on PCs (running Linux, FreeBSD, or Windows), and Sun, and DEC workstations.

III. PVM

PVM is particularly effective for heterogeneous applications, shared or local memory multiprocessors, vector supercomputer, specialized graphics engines or scalar workstations that may interconnected by networks. PVM is an integrated software tools and libraries that is mainly designed towards networks of workstations. PVM is a software system that enables a collection of heterogeneous computers to be used as a coherent and flexible concurrent computational resource, or a "parallel virtual machine". 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. So using the combined speed and storage of many computers, the large computational problem can be solved with more cost effectively. It was developed by the University of Tennessee, Oak Ridge National Laboratory and Emory University. The first version was written at ORNL in 1989, and after being rewritten by University of Tennessee, version 2 was released in March 1991. Version 3 was released in March 1993, and supported fault tolerance and better portability.

PVM is widely used message passing library created to support the development of parallel programs on heterogeneous environment. PVM program consists of a set of tasks that communicate within a parallel virtual machine by exchanging messages. A configuration file a created by user defines the physical machines that comprise the virtual machine. A managing process executes on each of these machines and controls the sending and receiving of messages among them. There are subroutines for process and initialization and termination for message sending and receiving for group creation for coordinating among tasks of a group for task synchronization and for querying and dynamically changing the configuration of the parallel virtual machine. 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.

PVM has been used in several implementations of exact algorithms to optimization problems. A parallel linear programming-based heuristic to solve set partitioning problems producing good solutions in a short amount of time. Several parallel metaheuristics for combinatorial optimization problems were also implemented using PVM.

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. The second part of the system is a library of PVM interface routines. PVM consists of a run-time environment and library for message-passing, task and resource management, and fault notification. While PVM will not automatically make a commercial software package run faster, it *does* provide a powerful set of functions for manually parallelizing an existing source program, or for writing new parallel/distributed programs.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 with 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 application programmer writes a parallel program by embedding these routines into C, C++, FORTRAN code. PVM has been used on the systems PCs running Win 95, NT 3.5.1, NT 4.0, Linux, Solaris, SCO, FreeBSD and parallel computers, such as CRAY YMP, Cray2, IBM SP-2, Intel Paragon.

IV. Analysis Of Tools

A. Portability

Both PVM and MPI are portable. 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. 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.

B. Heterogeneity

Heterogeneity refers to slowdown portability to virtual parallel machines which are obviously of different architectures. Both specifications support heterogeneity. But MPI does not mandate of that 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 the systems of later one. PVM has specific functions for the support to heterogeneity. LAM and MPICH are implementations of MPI that can run on heterogeneous networks of workstations.

C. Interoperability

Interoperability refers to the ability of different implementations of the same specification to exchange messages. 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. 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. 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.

D. MPMD

Both MPI and PVM different processes of a parallel program to execute different executable binary files required in heterogeneous implementation. Both support MPMD programs as well as SPMD programs, although again some implementation may not done so (MPICH, LAM support)

E. 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. 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. 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. Process Control

Process control refers to the ability to start and stop tasks , to find out which tasks are running and possibly where they are running . PVM contains all these capabilities , it can spawn/ kill tasks dynamically . In contrast MPI-1 has no method to start new task. MPI-2 contains functions to start a group of tasks and to send a kill signal to a group of tasks.

G. Dynamic Process Management

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. Each process ID in a group is known as rank. User processes in MPI can create new processes at runtime.

H. 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.

I. 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. 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.

V. CONCLUSIONS

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. The development of portable, efficient, and easy to use software tools, together with the availability of a wide range of parallel machines with large processing capacities, increasing reliability, and decreasing costs, has brought parallel processing into reality as an effective strategy for the implementation of computationally efficient techniques for the solution of large, difficult optimization problems.

In this article we have focused on both PVM and MPI have their distinctive features. Implementation issues may lead to programs that perform very poorly, independently of the quality of the underlying parallel algorithm. MPI is richer set of communication functions and PVM is effective for heterogeneous applications individual machines on a network. PVM is better fault tolerance than MPI. Performance evaluation tools have fundamental role in solving this kind of problems help to identify the origin of such bottlenecks. In global framework Programmers will enjoy more freedom for developing portable programs.

Programmer should evaluate the functional requirements and running environment of their application and choose the system that has the features they need.

REFERENCES

[1] Andrews, G.R. and Schneider, F.B., Concepts and Notations for Concurrent Programming, ACM Computing Surveys, Vol. 15, No.1 Mar. 1983,pp. 3-43.

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

[5] Cabillic, G. and Puaut, I., Stardust: An Environment for Parallel Programming on Network of Heterogeneous Workstations, Journal of Parallel and Distributed Computing, Vol. 40, No.1, 1997, pp. 55-60.

[6] Casavant, T.L., Tvrdik, P. And Plasil, F . (Editors), Parallel Computers: Theory and Practice, IEEE Computer Society Press 1996.

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

[8] 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.

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

[10] 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.

[11] 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.

[12] Geist, A., A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderman. PVM: Parallel Virtual Machine - A user's guide and tutorial for networked parallel computing, MIT Press, 1994. (http://www.netlib.org/pvm3/book/pvm-book.html)

[13] Gropp, W., E. Lusk, and A. Skjellum. Using MPI: Portable Parallel Programming with the Message Passing-Interface, MIT Press, 1994.

[14] Gropp, W., S. Huss-Lederman, A. Lumsdaine, E. Lusk, B. Nitzberg, W. Saphir, and M. Snir. MPI: The Complete Reference, Volume 2 - The MPI Extensions, MIT Press, 1998.

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

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

[17] Linderoth, J., E.K. Lee, and M.W.P. Savelsbergh. "A parallel linear programming based heuristic for large scale set partitioning problems", Industrial & Systems Engineering Technical Report, 1999. (http://www.isye.gatech.edu/faculty/Eva_K_Lee)

[18] MPI Forum. "A Message-Passing Interface Standard", The International Journal of Supercomputing Applications and High Performance Computing 8 (1994), 138-161.

[19] 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.

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

[21] S. Nemnugin and O. Stesik . Parallel programming for multiprocessor computational system. BHV-Petersburg, 2002.-400p, ISBN 5-94157-188-7 (In Russia).

[22] The MPI Forum. MPI : A message passing interface standard, 2012.

[23] W.D. Gropp. Learning from high success of MPI. In B. Monien, V.K.Prasana, and S.Vajapeyam, editors, High Performance Computing- HiPC 2001, number 2228 in Lecture notes in Computer Science, pages 81-92, Springer, Dec.2001.8th