Evolution of MPI Standard

By: Diksha garg

March 8, 2015

MPI is a de facto standard message-passing system designed by a group of researchers to work on variety of parallel systems. It is a good programming interface which is based on the concept of point-to-point communication. MPI provides portability and high performance for shared as well as parallel systems. There are several versions of MPI in market, currently used version is MPI3.0. In this paper, we have discussed various MPI standards, their need and evolution.

1 Introduction

Traditionally, programmers came up with the softwares for serial computation where a problem was divided into a sequence of instructions. These instructions were executed sequentially one after another on a single processor. Current trends in applications, computer architecture and networking suggest a future in which parallelism is relevant not only for supercomputers but also for workstations, personal computers, and networks. Given the proliferation of multicores, programs are required to exploit the multiple processors on-die as well as across the fabric. Efficient usage of multiple processors for a single task requires communication between all the processors. The communication between processors can be done using two well-known message passing approaches - Parallel Virtual machine (PVM) and the Message Passing Interface (MPI).

MPI is attracting programmers and gaining popularity in High Performance Computing. This model of parallel computation [2] is an expressive, efficient, and well-understood paradigm for executing parallel application on large scale cluster. Concurrency is a fundamental requirement for algorithms and programs which has motivated the study of MPI in parallel computing. Numbers of attempts are made to develop interfaces and standards for parallel computation. Highlight of good programming interface is the high level of abstraction which meets the user requirements without affecting its performance.

The main advantages of MPI are portability and high performance. The high performance structure is achieved by optimizing MPI libraries and providing user with full control over development cycle. Presence of MPI libraries on variety of machines and the standard API results in portable environment. In a nutshell, MPI program can be used on both shared and distributed memory systems. This paper analyses how the MPI standard expanded as a result of

variety of research and discussions. All the released versions of MPI are studied and analyzed.

The preceding sections are organized as follows. Section 2 introduces with related work available in this field. Sections 3 gives a brief overview of MPI followed by Section 4 which presents initial MPI standards released. Section 5 discusses the currently used version i.e. MPI 3.0 . Sum up of the lessons learned and possible future work is presented in Section 6.

2 Related Work

When MPI standard became available several open source implementations were developed. The main limitation of these implementations was portability. For instance, CHIMP-MPI[5] could only be used on IBM systems and LAM[7] operated only on specific HP workstations. Later these implementations were much more portable and easy to use. MPISecI/O[5] and MPICH[6] are the mechanism which provide security in network communications. MPI and openMP are commonly used parallel programming standards. Rabenseifer et al. [9] proposed a hybrid approach using both standards. S.Saini et. al.[10] describe the performance gain of hybrid models in terms of memory usage and CPU cycles. Absence of standard interface for communication between MPI processes and OpenMP threads was the main drawback of hybrid approaches. This resulted in poor performance of OpenMP. MPI 3.0[10] is working on its improvement. Research has been done on how MPI could be extended to work with multiple clusters. As per the defined standard, MPI works with only one cluster. Hafeez et. al.[4] discusses an approach which provides a framework build upon MPI so that different clusters can communicate over internet. A-JUMP[13] is one such mechanism which provides an interface for grid computing on distributed systems. As a result distributed clusters can communicate with each other. A-JUMP also provides support for hybrid models discussed above. Another approach which provides similar capabilities is mpiJava [1].

3 Overview of MPI

The base of message passing methodology was proposed by C.A.R. Hoare[17] in 1970 ,presenting the concept of sequential process communication in parallel systems. According to Hoare, every process performs its computation using its local address space and communicates with other processes by passing messages. This message passing is done by using library function calls like send, receive or broadcast to all the processes. Message passing concept works well for parallel systems having distributed memory fabric. Several vendors and labs came up with message passing implementations which focused on small user groups and suffered from issues like portability and large redundancy. This raised the need for a common standard. Later in early 90s, group of researchers collaborated to formulate the Standard for Message Passing and an initial draft was proposed

for MPI1 by Dongarra, Hempel, Hey, Walker[18,19]. On November 1993, this draft was formally introduced at Supercomputing '93 conference. 1st version of MPI, version 1.0 was made public in June 1994 after a series of meeting and email discussion among the members of MPI forum. The key points that MPI standard covered in its initial version were safe encapsulation of messages, high performance, scalable design and support for C and Fortran.

MPI is a specification of what library should be used by the developer working on message passing applications and not the message passing library itself. Its primary agenda is how the data is shared between address spaces of two cooperative processes maintaining efficiency and flexibility. The MPI standard evolved through number of iterations with the most recent version being MPI-3. Users of MPI standard should be familiar with the fact that MPI library implementations changes with version and feature of supported MPI standard.

MPI consists of collection of routines that supports point-to-point communication among processes. In order to achieve reliable communication, implementations for blocking and non-blocking send and receive are available. A blocking send is not a synchronous communication i.e. it does not return until the data is safely transferred preventing the edit and corruption of data buffers. On the other hand, a non-blocking send returns immediately while data buffers must not be used as they are volatile at this point and can get corrupted. This can be achieved by using an implementation which either blocks or perform a non-blocking check on status messages for preventing illegal buffer usage. Receiving process is suspended in blocking receive till the incoming message is placed in a particular application buffer where as in non-blocking receive, process is not blocked and returned before being placed in buffer while a call is made to communicate this action so that application can use the data in message.

There are some drawbacks that an MPI developer might encounter, Firstly a lot of effort is required for efficient partitioning of data. As the data and its related features may vary according to the application, it is often difficult to write and maintain long development cycles. Secondly, data is distributed among various data sources for performance improvement but it may result in duplication and large memory requirements. Thirdly, like other programming models, MPI also lacks fault tolerance that in turn makes it difficult to run large MPI jobs on parallel machines. The attempts are made to overcome these shortcomings in latest versions of MPI, while there is still room for further development and enhancements.

4 Initial versions of MPI

1. **MPI 1.0**

   MPI came into existence so there could be a standardization of message passing implementations/libraries. MPI 1.0 was one of the first widely used implementation. The framework was based on all the studies and research done till now for parallel systems. The key features of MPI 1.0 were collective communication maintaining the simplicity and basic function-

ality of message passing, safe encapsulation of messages enabling library building, high performance, scalable design and support for C and Fortran. After the series of discussions and meeting among the members of MPI forum MPI 1.0 was released in May, 1994. The basic prototype implementation was "mpich" and soon many other implementations [4] were written over this to support parallel implementations. MPI 1.0 became highly successful as it was abstract, common and efficient architecture with no implementation overhead. Also, the network structure has no explicit requirement. MPI processes can communicate using abstract medium with the restriction that only those processes within the common communicator context can send and receive messages from each other using point to point communication. The receiver needs to block until data is received. This ensures reliable communications with the guarantee that messages are sent in order. This standard provides orthogonality which means that communication can include any MPI datatypes. Error handlers were also implemented to ensure safe development environment.

Programmers felt that there were some functionalities missing in the initial version. It was not possible to query or reconstruct the structure defined for a given datatype. This limited the developers with few pre-defined datatypes. Also, these datatypes were not directly accessible which made the creation of certain types of library difficult. Other non-scalable features of MPI 1.0 includes irregular collectives, topology interface in which details for full process topology was required for all processes.

2. **MPI 2.0**

   Many key issues were not attended in MPI 1.0 because of lack of agreement between members of MPI forum and pressure of time frame in which the standard had to be made public. The issue like parallel input/output, one sided communication and dynamic process management were highlight in MPI 2.0, which were not present in previous release. The design principle of MPI 1.0 states that there should be no change in MPI object i.e. number of processes running in MPI_COMM_WORLD should not change, making it impossible to alter the running application by adding or removing processes. This was made possible in MPI 2.0 with dynamic process management where inter-communicators can establish a communication link between processes or applications that are already running. The new concept of communication window and epoch modeled one-sided communication without any support of specific hardware. This model drew a clear line between synchronization and communication. MPI 2.0 introduced data-type mechanism needed for MPI-IO(ROMIO) and library support. Communication was made faster by utilization of local file systems and data-types to describe the file structure.

   In practical applications, dynamic process management lacked fault tolerance in case of failure of inter-communicators. The memory and semantic model was too complicated, rigid and inefficient that programmers

refused to use it. In addition, the level of thread support was not extensively discussed. This raised the need for revised standards. MPI 2.2 was introduced in 2007 addressing the problems of scalability and missing functionality preserving compatibility with older versions. Hoefler et. al. [20] came up with new topology interface and graph for application communication in distributed environment, addressing missing functionality by MPI Reduce and providing more flexibility with new data-types. Fortran bindings were corrected and modernized with the intention of removing C++ bindings in later versions.

5 Latest version

MPI 3.0

With few years of analysis and experimentation with MPI 2.0, numbers of difficulties were encountered and a need for better version aroused. New functionalities introduced in MPI 3.0 are sparse collectives, non-blocking collectives, new one-sided communication and support of performance tool. In this version, C++ interfaces and deprecated functions were completely removed. Diving in to new features, non-blocking collectives was introduced to overcome performance overlap and increase user convenience. It allows the collective tasks to conduct operations with no blocking, resulting in considerable improvement in performance. MPI_Request object is used to check and ensure progress while keeping blocking and non-blocking collectives separate. This results in no tags and sound semantics on the basis of ordering. Hoefler and Traff [21] came up with sparse collectives which address the problem of scalability with irregular collectives. Neighborhood methods introduced topology functionality and their non-blocking versions. Improvement in performance is attained by one-sided communication model extension on hybrid/shared memory systems. One-sided operations were introduced for better handling of memory models. The operations were made atomic with passive synchronization [22] . Along with this, a new performance tool named MPIT Tool interface came up which enabled MPI implementation to uncover specific counters, internal variables and other states to the user. Multithreaded environment and Fortran 90 binding were significant addition.

There are certain areas left that are not yet addressed in any of the MPI versions. First, how those operations are handled which need more operating system support than currently provided, For example- remote execution, interrupt-driven receives or active messages. Second, there are no tools provided for program construction. Third, there is no support for debugging the MPI applications. MPI forum might include these in future versions.

MPI 1	MPI 2	MPI 3
The standard does not specify: • Explicit shared-memory operations • Operations needing operating system support than the standard; for example, interrupt-driven receives, remote execution, or active • Program construction tools • Debugging facilities • Explicit support for threads • Support for task management • I/O functions	In addition to the features of MPI 1,The standard includes: • The info object • Process creation and management • One-sided communication • External interfaces • Parallel file I/O	Features of MPI 2 were extended with some updates including: • Extension of collective operations to include non-blocking versions • Extensions to the one-sided operations • New Fortran 2008 binding • C++ bindings, deprecated routines and MPI objects removed. • Datatypes • Tool support
The standard does not specify: • Explicit shared-memory operations • Operations needing operating system support than the standard; for example, interrupt-driven receives, remote execution, or active • Program construction tools • Debugging facilities • Explicit support for threads • Support for task management • I/O functions	The standard does not specify: • Operations needing operating system support • Program construction tools • Debugging facilities.	The standard does not specify: • Operations needing operating system support • Program construction tools • Debugging facilities.

Table 1

6 Conclusion and Future Work

Table1 summerizes the significant progress observed in various versions of MPI. From this study we can conclude that additional control as well as additional communication patterns, reduction functions, etc. have been incorporated into

the standard library specification. In the future, we expect MPI to evolve to address emerging challenges born from the pursuit of the Exascale era. More specifically, achieving Exascale performance requires at least tens of thousands of processors. The system architecture for such massive supercomputers requires re-architecting systems to ensure reasonable latency and bandwidth. As such, software must evolve to adapt to such emerging architectures. Consequently, future versions of MPI may actually provide additional levels of abstraction to ease the burden on software to keep track of these tens or hundreds of thousands of CPUs. In other words, MPI may evolve towards the panacea of distributed shared memory.

References

[1] Message Passing Interface Forum, MPI2(1998): A message passing interface standard, Internet J. High Performance Comput. Appl.

[2] Gottlieb, Allan; Almasi, George S. (1989). Highly parallel computing. Redwood City, Calif.: Benjamin/Cummings. ISBN 0-8053-0177-1.

[3] Rolf Hempel, David W. Walker (1999).The emergence of the MPI message passing standard for parallel computing.

[4] William Gropp, Ewing Lusk ,Nathan Doss ,Anthony Skjellum (1996). A high-performance, portable implementation of the MPI message passing interface standard.

[5] R.A.A. Bruce, J.G. Mills and A.G. Smith. CHIMP/MPI user guide, Tech. Rept. EPCC-KTP-CHIMPVZUSER 1.2, Edinburgh Parallel Computing Centre, 1994.

[6] G. Bums, R. Daoud and J. Vaigl, LAM: An open cluster environment for MPI, in: J.W. Ross, ed., Proc. Supercompuring Symp. '94. University of Toronto (1994) 379-386.

[7] R. Grabner, F. Mietke, and W. Rehm, "Implementing an mpich-2 Channel Device over Vapi on Infiniband,"Proc. 18th Int'l Parallel and Distributed Processing Symp.,p. 184, Apr. 2004.

[8] William group (2002) MPICH2: A New Start for MPI Implementations.

[9] R. Rabenseifner, G. Hager, G. Jost, Tutorial on hybrid MPI and OpenMP parallel programing, in: Supercomputing Conference 2009 (SC09), Portland, OR,2009.

[10] S. Saini, D. Talcott, D. Jespersen, J. Djomehri, H. Jin, R. Biswas, Scientific application-based performance comparison of SGI Altix 4700, IBM Power5+, and SGI ICE 8200 supercomputers, in: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, 2008.

[11] R.L. Graham, G. Bosilca, MPI forum: preview of the MPI 3 standard. SC09 Birds-of-Feather Session, 2009. Available from: <http://www.open-mpi.org/papers/sc-2009/MPI_Forum_SC09_BOF-2up.pdf>.

[12] M. Hafeez, S. Asghar, U. A. Malik, A. Rehman, and N. Riaz, Message Passing Framework for Globally Interconnected Clusters, in International Conference on Computing in High Energy and Nuclear Physics (CHEP 2010), Taipei, Taiwan, 2010.

[13] Asghar S, Hafeez M, Malik U A, Rehman A and Riaz N A-JUMP Architecture for Java Universal Message Passing Proc. FIT 2010 (Islamabad, Pakistan 2010) doi:[10.1145/1943628.1943662]

[14] Yiannis Cotronis [2003] Building Grid MPI Applications from Modular Components.

[15] Baker M, Carpenter B, Fox G, Ko S, and Lim S 1999 mpi-Java: an Object-Oriented Java Interface to MPI Lecture Notes in Computer Science 1586 748–62.

[16] Mehnaz Hafeez. "Survey of MPI Implementations", Communications in Computer and Information Science, 2011.

[17] C.A.R. Hoare, Communicating sequential process, Commun. ACM 21 (8) (1978) 666-667.

[18] Hempel, Hey, McBryan, Walker: Special Issue – Message Passing Interfaces. Parallel Computing 29(4), 1994

[19] Hempel, Walker: The emergence of the MPI message passing standard for parallel computing. Computer Standards & Interfaces, 21: 51-62, 1999

[20] T. Hoefler, R. Rabenseifner, H. Ritzdorf, B. R. de Supinski, R. Thakur, J. L. Träff: The scalable process topology interface of MPI 2.2. Concurrency and Computation: Practice and Experience 23(4): 293-310, 2011

[21] T. Hoefler, J. L. Träff: Sparse collective operations for MPI. IPDPS 2009

[22] Hoefler, Dinan, Buntinas, Balaji, Barrett, Brightwell, Gropp, Kale, Thakur: Leveraging MPI's one-sided communication for shared-memory programming. EuroMPI 2012, LNCS 7490, 133-141, 2012]

[23] MPI documentation.

[24] History and Development of the MPI Standard by Jesper Larsson Traff