A Micro-Benchmark Framework for evaluating Performance of MPI

Manisha Dhanke, Subba Ramanna, Rakesh K Agrwal
IBM, Bangalore
{madhanke, subbodda, rkagrwal}@in.ibm.com

Abstract

Several approaches have been used to benchmark the performance of MPI point-to-point, and collective communications calls on parallel systems. One of the major overheads incurred in the execution of parallel programs is due to communication of information among processes. The latency and overall cost of communication is dependent on a variety of design and configuration parameters that include the network topology, routing, algorithms and the software protocols. Handling messages refers to how messages are passed on from one node to the next. Hence, providing a well defined set of micro-benchmarks that can serve as a conceptual framework for evaluating the performance of MPI communications on different computing platforms is important. The software being presented here is a suite, which measures and provide the basis for the analysis and performance of different point-to-point and, collective communication calls. ,The benchmarks are designed to provide the end user with insights into the functionality and performance of the MPI library, the transport protocols, and the interconnect topology. It would also help provide insights into the expected behavior of MPI based parallel applications on the underlying system..

Introduction

MPI [1] is a standard which provides a flexible environment for developing high performance parallel applications in a portable fashion with several mechanisms for point-to-point and collective communications. In MPI, it is often possible to express the same application communication requirement in many different ways, using different combinations of MPI primitives, with different types of resource overheads, and with different logical topologies. For example, MPI provides different send and receive types, with different synchronization and resource usage semantics. Measuring communication overheads for different MPI library calls on a given cluster play an important role for performance estimation of parallel applications and system benchmarks on parallel systems. Latency and bandwidth are two fundamental metrics used to assess the performance of a network or of a system interconnect.

Quantifying overheads plays an important role in the estimation of performance of parallel application. Since parallelism and interaction overheads can be a significant part of many parallel applications, quantifying and segmenting them can help in the analysis of the performance estimation. The time to execute a parallel program depends on the time needed for the computation (potentially in parallel), the interaction operations, and synchronization and communication overheads. The overheads in a parallel program can be divided into three classes: The load-imbalance overhead, the parallelism overhead, and the interaction overhead. There are three types of interaction operations, which are sources of interaction overhead: *Synchronization*, such as barrier, locks, critical regions and events, *Aggregation*, such as reduction and scan and *Communication*, such as point-to-point and collective communication and reading/writing of shared variables. Understanding the overheads can help a programmer to decide how to restructure the parallel program for maximum efficiency. Although measuring overhead may look deceptively simple, to obtain accurate measurement results is a challenging undertaking. This is a difficult exercise due to three reasons: i) most computer systems provide coarse time resolutions, on the order of microseconds or even milliseconds; ii) the processors in a parallel computer, especially clusters, often operate asynchronously, not following the beat of a common clock causing time synchronization problems. It is, usually, very difficult to force the processors to start an operation at the same time (as an example measuring the time for send/recv when the two operations are occurring on different nodes and hence on different clocks) and iii) the measurement results can sometimes vary significantly even for the same communication operation.

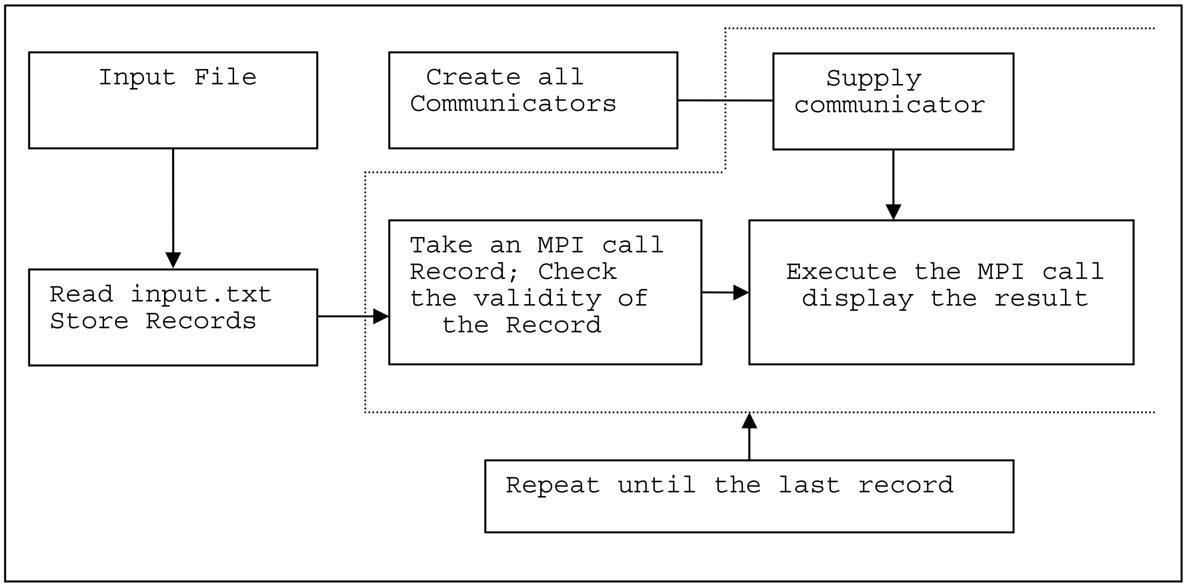
Motivation for the proposed micro benchmark suite

There are several popular MPI benchmarks such as IMB, SkaMPI, MPBench, Pallas [7,8,9], but these benchmarks either cover a small set of operations covering some of the MPI calls or else cover only few datatypes and patterns. The MPI benchmarks being discussed in this article cover almost most of the commonly used MPI-1 and MPI I/O operations. These benchmarks are also aimed at testing different MPI datatypes for different collective computation operations with a very generic design of the code wherein a datatype or operation would become an argument and same code is reused for each benchmark. Testing the collective computation calls with different reduction operations such as MPI_SUM, MPI_MAX brings out performance due to any optimizations done in the software or hardware for such operations along with communication to happen. Another important feature of these benchmarks is to cover different physical topologies of the system and associated task geometries in the case of collective communications and computations. The benchmarks form communicators over the nodes of the parallel system forming physical topologies like lines, plane, cuboids and helps in easy tabulation of performance for the various options which enables study of performance in case of hardware modifications and connectivity changes. On systems like BlueGene [10] where different topologies are used for different types of MPI communications, such as tree network topology for collective calls, it is important to observe how the latencies and bandwidths vary across different lines, planes, different physical topologies and task geometries. This also becomes effective in case of different large bandwidth switches changing the routing protocols for performance improvement. The creation of communicators over such physical topologies is dependent on system, hence, this suite provides simple and generic interface for writing code for creation of communicators based on physical location which allows users to simple plug code as per their requirement to support their own topologies.

Architecture and Features

The design of the benchmark suite is kept simple, modular, reusable and easy to add additional metrics. It is flexible and can be modified with ease and according to requirement. The input file is used to store the information in record format and each record has details about a particular MPI call.   A record with information about a MPI call has a number of attributes such as: data type, communicator, element size range, and its increment type and operation which is specific to MPI call MPI_Reduce and MPI_Allreduce.

The block diagram for the overall architecture is drawn in figure below. The input file is prepared as per each MPI call. This system reads the input file, stores all the records with attributes. It creates all the included communicators mentioned in the input file.


Input File
Create all Communicators
Supply communicator
Read input.txt Store Records
Take an MPI call Record; Check the validity of the Record
Execute the MPI call display the result
Repeat until the last record

From the number of records, each record is taken one by one for execution of an MPI call. If the record is valid, the needed communicator is supplied and MPI call is executed. The record format of the input file is shown in figure below

MPI call	Data Type	Element Size(Min)	Element Size(Max)	ADD/MUL	Val	Comm/Top.	Operation	Stride

Following are the features of the MPI benchmarks out of which most are not available in currently available MPI benchmarks.

1. Useful for testing the functionality, analyze the performance of MPI library as well as the system interconnects, topologies and the transport protocols
2. Timestamps almost all important MPI point-to-point communication calls, collective communications, collective communication and computation calls, I/O calls.
3. Tests different physical topologies like lines, planes, cube, etc.
4. Tests the MPI calls for different message sizes. Takes care of accurate timings by averaging and having a warm-up loop before actually starting the benchmark timing.
5. Tests different data-types such as MPI_INT, MPI_FLOAT, MPI_CHAR, MPI_DOUBLE and complex data-types.
6. Tests the collective computation calls with different operations such as MPI_SUM, MPI_MAX, complex data-type operations.
7. Tests data with various strides (byte alignment).

8. Code is written in very generic form that there is only a single executable created which uses a single input file for taking the list of benchmarks, operations, data-types, and topologies.
9. The framework uses a generic style with high reusability of the code in such a way that data-types, operations; message sizes would become just arguments in the same code written once.
10. The framework is written in a very modular fashion that it is very easy to insert a new benchmark, data-type, operation, or physical topology.
11. The coding of point-to-point communications allows user to either chose detailed information about all the pairs of ranks or simple summary of all the pairs of nodes on the system. Also, it allows to choose specific pairs of nodes which have a specific criteria.

Following is the design of the steps followed by the suite of MPI benchmarks. The steps highlight the modularity, reusability, effectiveness of the measurements, flexibility of the code as well as the benchmarks.

```
→
Read the entries in input file where each entry consists of benchmark name, datatype,
minimum and maximum number of elements, increment operation, topology and operand
for the benchmark.

→
Call functions pertaining to creation of different topologies such as lines, planes, cube
and create an array of communicators. All processes enter these functions and as per the
criteria, they would form communicators. Developers who want to add more topologies
can just add the functions for the topologies and assign in the array.

→
For each entry in the input file

        →
        Based on the topology specified, get the MPI communicator from the array of
        communicators assigned in step above

        →
        Get the operand, operand for benchmarks which include point to point
        communications would include either NOP or DETAIL, DETAIL would give a
        detailed output of the benchmark.

        →
        For the message size range specified
             →
             Execute the benchmark for the operand and communicator specified

        End For
End For
```

The output is designed in such a way that it is descriptive in form of sentences as well as forms a specific format wherefrom a graphic tool could easily grab the data and use for graphing.

Implementation Details

The implementation is done using MPI programming in C programming language. This test suite is tested on the IBM Blue Gene system and on some other linux cluster.

MPI Calls		Data Types	Communicators	Operation
Single Point-to-point Blocking calls	Send, Recv Sendrecv, Sendrecv replace, Bsend, Rsend, Ssend	MPI_INT, MPI_FLOAT, MPI_CHAR, MPI_SHORT, MPI_DOUBLE MPI_LONG	XY-PLANE,YZ-PLANE ZX-PLANE, X-LINE Y-LINE, Z-LINE HOLLOW-CUBE MPI_COMM_WORLD	None
Single Point-to-point Non Blocking calls	Isend, Irecv, Recv, Ibsend, Irsend, Issend	MPI_INT, MPI_FLOAT, MPI_CHAR, MPI_SHORT, MPI_DOUBLE MPI_LONG	XY-PLANE,YZ-PLANE ZX-PLANE, X-LINE Y-LINE, Z-LINE HOLLOW-CUBE MPI_COMM_WORLD	None
Collective Calls	Bcast, Gather, Gatherv, Allgather, Allgatherv, Scatter, Scatterv, Alltoall, Alltoallv	MPI_INT, MPI_FLOAT, MPI_CHAR, MPI_SHORT, MPI_DOUBLE MPI_LONG	XY-PLANE,YZ-PLANE ZX-PLANE, X-LINE Y-LINE, Z-LINE HOLLOW-CUBE MPI_COMM_WORLD	None
Global Reduction Calls	Reduce, Allreduce, ReduceScatter, Scan	MPI_INT, MPI_FLOAT, MPI_CHAR, MPI_SHORT, MPI_DOUBLE MPI_LONG	XY-PLANE,YZ-PLANE ZX-PLANE, X-LINE Y-LINE, Z-LINE HOLLOW-CUBE MPI_COMM_WORLD	MPI_SUM, MPI_MAX, MPI_MIN, MPI_LAND, MPI_BAND

The table above in the support of the citation of the measure of the suite is given. In record format element size for any MPI call varies from minimum to maximum, and the way it increases size depends on the next parameter which can be ADD or MUL. ADD will keep on adding till the maximum size is reached. It adds the number given in value parameter. Similarly MUL will keep on multiplying with the value. The last entry in the record format is stride, which considers the byte alignment in the sending buffer for an MPI call. If this value is 1, the buffer starts reading from the second byte leaving behind the first byte empty.

If any new operand or benchmark needed to be added, it is very easy to add with minimal coding. And any existing benchmark's code need not be touched while adding a new operand or topology.

Future Work

The present work can be extended to include more complex data types in more complicated hardware environment. The hardware environment can be forming any

Linux is a trademark of Linus Torvalds in the United States, other countries, or both. IBM is a trademark of IBM Corporation in the United States, other countries, or both. BLUEGENE is a trademark of IBM Coporation in the United Sates, other coutries, or both. Other company, product, or service names may be trademarks or service marks of others.

complex structure i.e. tree or any graph structure in two or three dimensions. The communicators can be created out of those; the optimization of that communicator can be analyzed. We are also planning to include one sided communication test suites and more features from MPI 2 implementation

This work can add more shapes as communicator such as sphere or conic shape, to determine if there might be any improvement in communication performances. Although it has been tested on IBM Blue Gene system and other linux cluster, it can be made tested on all types of clusters.

References

1. Message Passing Interface Standard and Forum. http://www-unix.mcs.anl.gov/mpi
2. Kai Hwang, Zhiwei Xu, *Scalable Parallel Computing: Technology, Architecture, Programming*, McGraw Hill Series in Computer Sciences, New York. (1997).
3. Paecho, S Peter, University of Sanfrancisco. *Parallel Programing with MPI*, Morgan Koffman Publishers, Inc., Sanfrancisco, California (1996).
4. William Gropp, Ewing Lusk, Anthony Skjellum, Argonne National Laboratory. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, The MIT Press, Cambridge, Massachusetts, London, England. (1994).
5. HPCC Software. www.cdacindia.com
6. Argonne National Laboratory Implementation of MPI, mpich. http://www-unix.mcs.anl.gov/mpi/mpich
7. IMB : Intel MPI Benchmarks formerly known as Pallas.
   http://www.pallas.com/e/products/index.htm
8. SkaMPI : http://liinwww.ira.uka.de/~skampi/
9. MPBench : http://icl.cs.utk.edu/projects/llcbench/
10. BlueGene : http://www.research.ibm.com/bluegene/