Overlap Communication in MPI Implementations

Thananon Patinyasakdikul[1*] George Bosilca[1] and Jack Dongarra[2]

[1]*Innovative Computer Laboratory, University of Tennessee*
[2]*1120 Volunteer Blvd University of Tennessee, Knoxville, TN 37916*

ABSTRACT

High performance computing has been a very big trend for the past decade. We are in the age that can not just rely on speedup from silicon alone anymore. Computer scientists have been implementing ways to improve the performance of application by making use of the increased of CPU core. One of those technique is "Message Passing Interface (MPI)", which has a long history back to 1980. MPI standard has been improved since then. Many organizations developed their own implementation of MPI under MPI standard to serve as a tool to achieve higher performance. In this paper, we focus on the basic part of MPI which is point to point communication. We benchmark famous MPI implementations and implement overlap communication for Open MPI. We demonstrate the process, the problem we encountered and the result of our implementation.

ARTICLE INFO

Article history:
Received 2 July 2014
Received in revised form
12 September 2014
Accepted 19 November 2014
Available online
20 December 2014

Keywords:
MPI
overlap
benchmark
asynchronous
point to point

Introduction

MPI is a programming scheme to speed up many applications that has been standardized in early 1991 but is not widely used because back then, we could still rely on the increasing clock speed of CPU. When Moore's law is slowed down in late 2000 (Ian, 2013) programmer looked back to MPI. MPI provide rich feature to the user and give the user freedom of design with derived datatype and communication pattern. The standard allows the user to run MPI program from 20 years back in modern hardware. MPI program can also run on any machine (Portable) no matter what implementation of MPI library installed on the machine.

* *Corresponding author.*
E-mail address: tpatinya@vols.utk.edu

MPI Implementations

MPI has multiple implementations but is bound by MPI standard (MPI Forums, 2000). The user does not have to know about what is inside each MPI implementation and can expect MPI to do its best job to optimize every operation. There are 3 major MPI implementations that is being used nowadays. Intel MPI, MVAPICH and OpenMPI. These three implementations is also the subject of this paper.

MPICH is one widely used implementation of MPI including the world fastest computer, Tianhe- 2 (Gropp, 1996). That can run on any UNIX-like operating system. The 'CH' in 'MPICH' comes from 'chameleon' which represents portability which is known as its main feature. The original MPICH implemented MPI-1.1 standard and keep on updating itself to MPI standard (currently with MPI-3.0 standard).

MVAPICH is an implementation that uses MPICH as the foundation. It is widely used by supercomputers in national labs, industries and universities. Now MVAPICH is maintained by Ohio States University (Ohio State University, 2002). MVAPICH also has been used on many supercomputer in TOP500 list (TOP500.org, 2014). Which is the supercomputer speed ranking that updates every year. Intel developed their own MPI library also use MPICH as foundation. Along with many Intel libraries, MPI is one of the most powerful library of Intel. As usual, it is not open-source software and the user have to purchase it from Intel at 499$. Intel advertised its own library as 'Scalable' which support up to 120,000 processes at the same time (Intel, 2014). Open MPI is a merge from University of Tennessee's FT-MPI, Los Alamos National Lab's LA-MPI and Indiana University's LAM/MPI (Gabriel, 2004). The goal of Open MPI is to create a free, open-source implementation that has high performance, high reliability and user friendly. Open MPI is also used in many supercomputer in TOP500 list.

To choose between MPI implementations, the user has to match his application characteristics with MPI implementations characteristic. Some MPI might be better in some aspect and worse in another. The research beforehand might help in the selection process.

MPI Communication

In MPI, we have numerous number of communication method that users can use to create their own application. MPI gives very high flexibility and rich customization to the users. We classified 2 types of communication in MPI as *Point to Point communication and Collective communication.*

Point to Point Communication

Point to point communication is the communication between two processes as shown in Figure 1 Examples of point to point communication method is MPI_Send() and MPI_Recv(). It is useful to send the data to a particular process or signal another process to start the procedure.

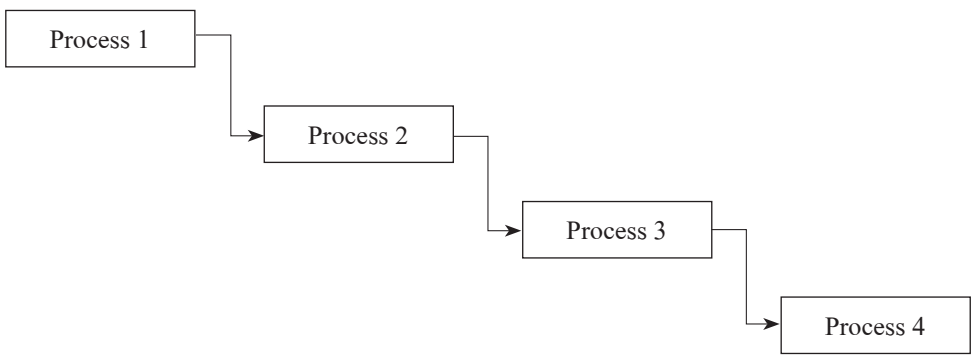


Figure 1 Point to point communication

Collective Communication

Collective Communication is communication that involves all the process in MPI Communicator. For example, broadcasting. Broadcasting is sending data from 1 process to every process. After the broadcast, everyone will have the same data. The implementation of collective communication is to make use of multiple point to point communications to send and receive messages in some certain order that depends on MPI implementation.

Consider this user application shown in Figure 2

```
1:      If(rank==0) {
2:              MPI_Isend();
3:              Work();
4:              MPI_Wait();
5:
6:      }else if(rank==1){
7:              MPI_Irecv();
8:              Work();
9:              MPI_Wait();
10:
```

Figure 2 Simple Non-Blocking Point to Point Communication code

The user posted Isend request and then go to work() which might run for some amount of time and then return to do MPI_Wait() on line 4. The posted send will not progress. The reason is because the user doesn't give time to MPI library to do so. The user was busy calculating another instance of the program resulting in communication halt. The posted send will get process and done in MPI_Wait() where the user give runtime to MPI library.

The expected runtime in non-overlapping communication illustrated in Figure 3 (a). The total runtime will be T(Isend) + T(Work) + T(Comm) + T(Wait).Next, with overlap communication illustrated in Figure 3 (b), MPI library can progress on the communication by itself while the user is doing another work.

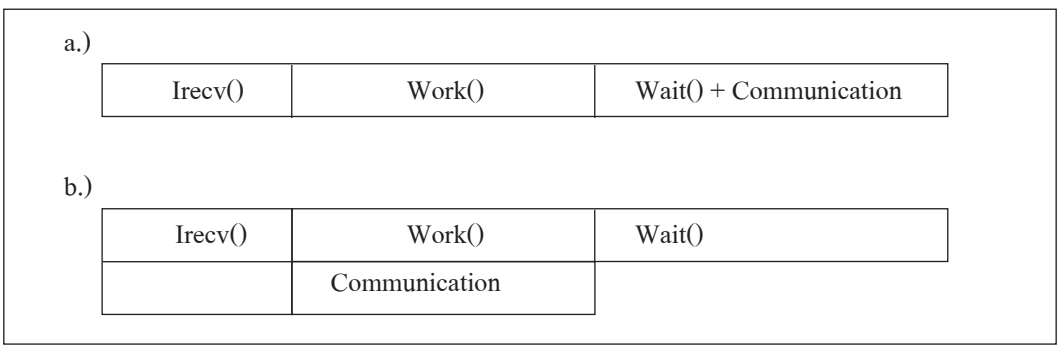


Figure 3 the runtime diagram of the overlapped application

The total runtime will be T(Isend) + max (T(Work),T(comm)) + T(Wait) which is better. This also applies to the sender on the other end of the network. In this project, we will focus on the overlapping for every MPI implementation we have. We will see how the implementation of overlap communication affects other aspect of MPI communication.

Implementation

To enable overlapping, the team at ICL creates a communication thread to help progress all the communication request on Open MPI. We begin with the simplest form of communication, which is point to point.The communication model in OpenMPI is demonstrated in Fig 4. We created the thread in a module of Open MPI called Byte Transporting Layer (BTL) which takes care of low level communication. We redirect send and receive events to this thread and have the original Open MPI wait for the result of this thread as illustrated in Figure 5In order to do this, we must make sure that every operation is thread safe.

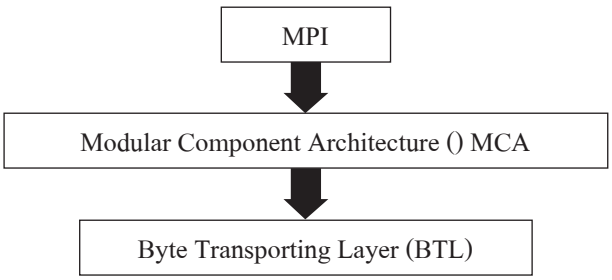


Figure 4 MPI Communication model

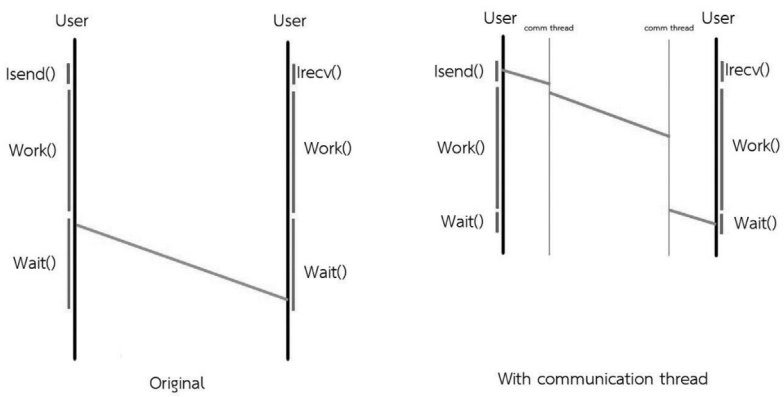


Figure 5 The communication thread progress in the communication while user's application continue to do work simultaneously

To guarantee the correctness of execution, we have to concern about thread safety (Race conditions) inside of the MPI library. MPI library doesn't provide the thread-safe environment because there was only one thread on communication issue. To increase the number of thread, we implemented locks and unlocks on key environment variable, create critical section where only one thread can pass at a time, etc. The thread-safe method is expensive in term of runtime. It is widely known that lock and unlock variables will cause the drop in performance as the trade-off for correctness. In this project, we will see the impact of this factor.

Currently, this implementation on Open MPI is done and it is on performance testing phrase. We verified the correctness of messages that get sent with the communication thread. The performance will be shown on the next part.

Performance

In this part, we will compare the overlapping communication on every MPI implementation to our newly implemented Open MPI to see if we achieve the higher performance than others then we will look deeper and see the impact of our newly implemented communication thread to other aspect of the communication.

Experimental Setup

Per node,

2 x Westmere-EP E5606 @2.13GHZ, 8 cores

24 G RAM

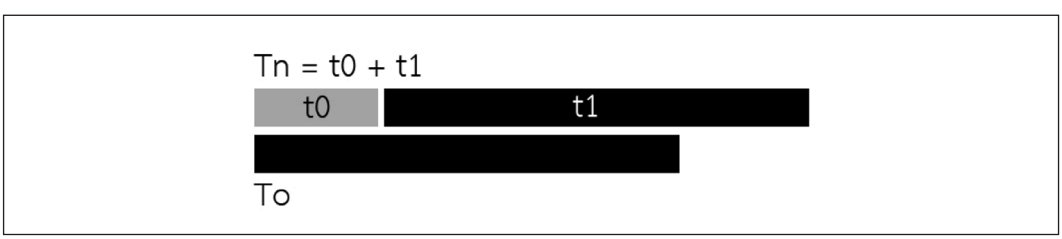
Infiniband 10G

OpenMPI 1.8

Simple Overlap testing

The algorithm to test the significance of the overlapping is the simple program described in part (3). Where we have two process trying to do non-blocking send/recv. We first measure the time it takes to transmit a message of size k to another process and save it to t0. Now, rerun the program again with nanosleep() to simulate the work between Isend/Irecv and Wait() with sleep time t1 .

Without overlapping, the overall runtime T_n should be $t_0 + t_1$. If the MPI implementation has overlap in communication, the overall runtime T_o should be lesser that $t_0 + t_1$.We then calculate the overlapping percentage O by using the formula illustrated in Figure 6



$$O = ((T_o - t_0) - t1) / t_1 * 100\%$$

Figure 6 Calculating the overlapping percentage

We vary the size of t1 to see the difference between the size of the work then we compare the overlapping percentage between implementations of MPI.

Figure 7 compares overlapping percentage of point to point communications of Open MPI before and after our implementation. We can see that we achieve high percentage of overlapping. This proves that the communication is really happening even when the user application is not in MPI call. Since now we know that our implementation is doing what we want, we now see if there is any other implementation of MPI that allows overlapping. So we compare vanilla Open MPI with other MPI implementations.

Figure 8 shows that MVAPICH has overlap communication but not as big as our implementation of Open MPI. For other implementation, it is safe to say that there is no overlapping communication implemented and we are seeing noise in the figure.

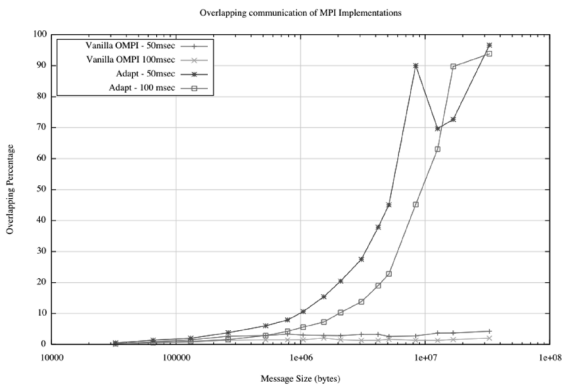


Figure 7 Overlapping percentage of MPI Implementations

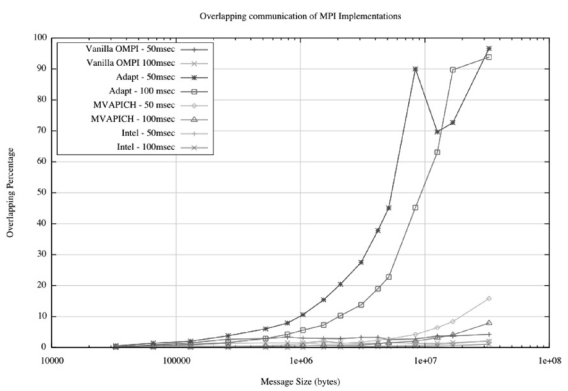


Figure 8 Overlapping percentage of MPI Implementations

In conclusion from this test, we know that only MVAPICH and our Open MPI – ADAPT are able to do overlapping communication. However, there is a very big gap in overlapping percentage between MVAPICH and us.

Now that we know we have what we want, we now look and see what is the trade-off for this implementation.

Latency test

If we are doing multiple thing at the same time, it is suspected that we will have increased latency due to the calculations and precautions that we have to make and prepare before the calculation. To measure the latency, we used the well-known benchmark called NetPIPE. The result is shown in Figure 9

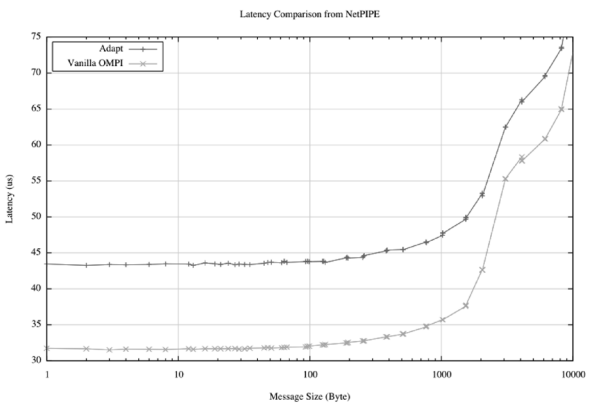


Figure 9 Latency from NetPIPE benchmark

We only compared the Adapt to the vanilla Open MPI just because it is fair. To get the overlapping, we modified Open MPI. Now we compare with our original and see that the latency increased as we expected but the disturbing result is we have 50% more latency than vanilla. It should not be that high because introducing mutex locks should cause us lesser than this.

MPP Test

MPP test is a well-known MPI benchmark published in 1999. It test various aspect of MPI performance testing. For more information. After a peek into MPP test source code, I found out that MPP test is doing similar thing as my simple benchmark. First, they run the communication with

some work between Isend and Wait and measure the time. The difference is they are doing some calculation on the work part but in our simple benchmark, we only sleep the application. The work they do is simple enough, they cycling through array indexes.

Up to this part, the expectation of benchmark result is clear. It should go in the same fashion as my simple benchmark since they're doing basically the same thing. The result is shown below in Figure 10.

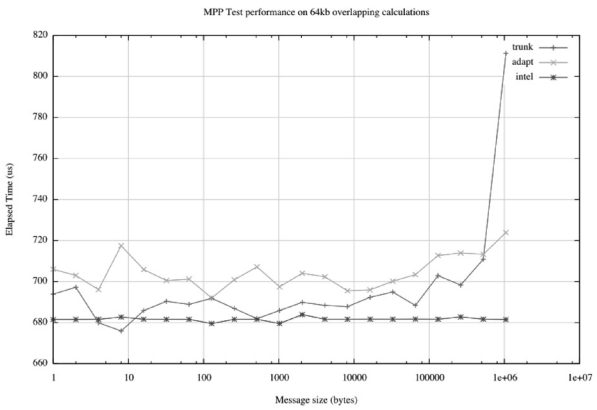


Figure10 Overlapping performance from MPPtest benchmark

Interestingly, it doesn't. We can see that we take more time than vanilla OpenMPI to send the messages. The result is contradicting to our simple benchmark. But how?

After some discussion with the team, team suggested that I should check if the communication thread is bind to the same core on CPU or not. Here is the explanation, if user application and the communication thread are bind to the same core then only one can active at a time and the other has to wait. In our simple benchmark it doesn't matter because we put the application to sleep so it give the quantum to communication thread and we can get the result but in MPP benchmark, the application doesn't go to sleep. Instead it's doing some calculating and doesn't give the communication thread the quantum and now the communication thread has to wait. So we can't achieve the overlap benefit. If that's the case then the runtime should be the same but with additional thread safety measures, we are spending more than the original and that resulted in increased runtime.We then immediately modified the code to bind communication thread to another available CPU core and retest the performance with MPP test. This time we get the performance we hope for but still not outstanding.

Conclusionand future work

Achieving overlapping in MPI is not a trivial thing to do. We have to sacrifice the latency in order to enable the communication thread. Everything that involves data moving and the mutex locks are very expensive. However, we proved that it is possible. This will accelerate every user application that is work intensive, ideally for the application that requires equally calculation and communication. The increased overhead might not suit the application that is not communication intensive.

After we decrease the overhead to satisfy our goal, we will implement the overlapping in collective communication. Which will guarantee that we will have overlapping in every form of communication in OpenMPI. The user will benefit more, especially the parallel linear algebra calculation which presents in almost every scientific calculation/simulation software. Those applications require a lot of collective communication such as broadcasting or reduce.

References

Ian, P. (2013). **The end ofMoore's law is on the horizon** [On-line]. Available:http://www.pcworld.com/article/2032913/the-end-of-moores-law-is-on-the-horizon-says-amd.html

MPI Forum. (2012). **MPI: A Message-Passing Interface Standard** [On-line]. Available: http://www.mcs.anl.gov/research/projects/mpi/mpi-standard/mpi-report-1.1/mpi-report.html

W. Gropp and E. Lusk and N. Doss and A. Skjellum. Parallel Computing 1996, **A high-performance, portable implementation of the {MPI} message passing interface standard.** Pittsburgh, Pennsylvania.

Ohio State University. (2002). **MVAPICH: MPI over Infini Band, 10GigE/iWARP and RoCE** [On-line]. Available: http://mvapich.cse.ohio-state.edu/

Top500.org. (2014). **Top 500 Supercomputer Sites** [On-line]. Available: http://www.top500.org/

Intel. (2012). **Intel MPI Library** [On-line]. Available: https://software.intel.com/en-us/intel-mpi-library

Edgar Gabriel, et al. Euro PVM/MPI 2004, **Open MPI: Goals, Concept, and design of Next generation MPI Implementation.** Budapest, Hungary.