Performance Analysis Of MPI Over Cluster

Achuth PV, 133079007
Anirudh Rao, 133079005
Avishkar P, 143070037

1 Introduction

Clusters are widely used in the field of high performance computing to scale computing beyond a single node. In clusters, multiple nodes connected via communication networks share the computation load, thus giving scalability and speed up. But, the communication networks between these nodes have always been a major bottleneck in the performance and scalability of these high performance clusters.

There are multiple modes of communication used in these clusters. Some examples are: Ethernet, Myrinet, InfiniBand, etc. Nodes communicate via these interfaces using various message passing systems, message passing interface (MPI) being one of the standard and commonly used systems.

Ethernet is the most widely used LAN technology. It works using TCP/IP stack and is based on IEEE 802.3 standard. It supports a wide range of connectors and has good interoperability and is cheap.

InfiniBand (IB) is a communication architecture which is used for low latency and high bandwidth inter-server communication [2]. It was developed as part of Open Fabrics Alliance. It is highly reliable, scalable and has very good performance. Unlike Ethernet based systems which runs algorithm on CPU, the IB adapter can handle network protocol itself, thus freeing up CPU and OS, as well as bypassing multiple system calls. Depending upon the PCI-e technology node's motherboard supports, the performance of InfiniBand network changes.

MPI can also be used with CUDA as CUDA over MPI to utilize the GPU cards residing in multiple nodes in a cluster. The communication bottle neck in this method is that, for each data transfer between machines, data has to be moved from the device to the host memory and then use MPI to transfer the data between machines. The data is then copied to device from host at the receiving node. CUDA-Aware MPI [1, 7] is used to reduce this overhead. It uses unified virtual addressing (UVA) feature, available in CUDA cards with compute capability 2.0 and later, and CUDA 4.0 and later, which combines the host memory and memory of GPUs within a system as one large virtual address space. CUDA-Aware MPI thus supports transferring GPU buffers which exists on a particular machine's GPU card to other machine's GPU card directly, without using host buffer. This reduces many overheads and thus increases communication bandwidth between GPU cards over network. Thus it is easy to pipeline all operations requiring message transfer. Also features like GPU direct can be used by MPI libraries to speed up the communication, by bypassing multiple hardware level overheads.

In this project, We intend to evaluate various communication platforms available for parallel computing/ cluster computing. We evaluate different MPI implementations, CUDA, CUDA-Aware MPI techniques over different network setup like Ethernet, IP over InfiniBand and shared memory. For this purpose, we use a matrix multiplication program, with matrix size varied from 2000x2000 to 10000x10000, and plot the various computation and data transfer times. The plots are used to evaluate and provide inferences.There are 2 open implementations of MPI available, which support InfiniBand communication along with Ethernet Viz. OpenMPI and MVAPICH2. Both these implementations also support CUDA-Aware MPI technique. We use these two implementations for our evaluation.

2 Tools and Methods

For our work, we used the following tools: OpenMPI and MVAPICH2. Both of them implement MPI standard. OpenMPI targets certain common usages, while MVAPICH2 is an extension of MPICH to enable support to different communication modes. Both of them support lots of communication interfaces like InfiniBand, Ethernet, etc. They also support CUDA-Aware MPI.

Our test code, matrix multiplication was performed using OpenMPI and MVAPICH2 over a cluster of 2 machines connected using InfiniBand as well as Ethernet. This test program was run with different matrix sizes, different number of processes per node and different modes of communication.

Using OpenMPI, MVAPICH2 implementation, the following combination of the code was run:

1. OpenMPI with matrix size 2000, 4000, 6000, 8000, 10000 on a single node with 8,16 processes
2. OpenMPI with matrix size 2000, 4000, 6000, 8000, 10000 on two nodes with 8,16,32 processes and communication between hosts through Ethernet

3. OpenMPI with matrix size 2000, 4000, 6000, 8000, 10000 on two nodes with 8,16,32 processes and communication between hosts through InfiniBand
4. MVAPICH2 with matrix size 2000, 4000, 6000, 8000, 10000 on a single node with 8,16 processes
5. MVAPICH2 with matrix size 2000, 4000, 6000, 8000, 10000 on two nodes with 8,16,32 processes and communication between hosts through Ethernet
6. MVAPICH2 with matrix size 2000, 4000, 6000, 8000, 10000 on two nodes with 8,16,32 processes and communication between hosts through InfiniBand

To check the performance of CUDA-Aware MPI, we setup a CUDA cluster. We installed OpenMPI and MVAPICH2 on this cluster. Codes for both the CUDA over MPI and CUDA-Aware MPI bandwidth test are written such that, they send messages of lengths from 1 Byte to 4 MB between two MPI processes. In the CUDA over MPI case, we use pinned host memory to communicate between the host and the device, thus speeding up host-device communication. We used non-blocking method for communication. Two methods are used for evaluation:

1. Two processes are run on same node.
2. Two processes are run on separate nodes.

Bandwidth for different message sizes are used for performance evaluation. We also implemented matrix multiplication using CUDA-over MPI, where we evaluated timing performances for:

1. Matrix size 2000, 4000, 6000, 8000, 10000 on a single node with 4 processes
2. Matrix size 2000, 4000, 6000, 8000, 10000 on two nodes with total 4 processes and communication between hosts through Ethernet

3 Details of Infiniband Cluster

- Number of machines: 14 - 2 masters, 12 slaves
- Filesystem: GlusterFS
- OS: CentOS release 6.5 (Final)
- CPU: Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz
  - 2 CPUs: NUMA with 8 cores/each, 1 thread/core
  - Cache: L1d - 32K, L1i - 32K, L2 : 256K, L3 : 20480K
- 64GB RAM
- OpenMPI Version : 1.8.4
- MVAPICH2 Version: 2.1
- InfiniBand card: QLogic Corp. IBA7322 QDR InfiniBand HCA
- Infiniband switch : QLogic 12200-18 36-ports
- Link speed: 40 Gb/sec (4X QDR)
- Ethernet controller: Intel Corporation I350 Gigabit Network Connection

For more details about CPU and memory, please refer Infiniband_CPU_details.txt and Infiniband_Memory_details.txt.

4 Details of CUDA Cluster

We setup a CUDA cluster of 4 machines. All the machines are Dell Optiplex 3020 with the following specifications

- 4 Machines
- Ubuntu 12.04 64 bit
- Intel(R) Core(TM) i3-4130 CPU @ 3.40GHz
  - 2 cores, 2 threads each
  - Cache - 128kB L1, 512kB L2, 3MB L3
- 4 GB RAM
- GeForce GT 640
  - CUDA cores 384
  - 2048 MB RAM
  - Memory clock rate: 891 Mhz
  - Memory bus width: 128-bit
  - GPU clock rate: 902 MHz
- CUDA Toolkit 6.5
- 500 GB harddisk
- Ethernet controller: Realtek Semiconductor Co., Ltd. RTL8111/8168/8411
- OpenMPI: 1.8.8
- MVAPICH2: 2.1

To set up the CUDA cluster, following steps were followed:

- A file based partitions is created [3]
- A NFS server is setup on the same machine, sharing the file based partition [4]
- An user account 'mpiuser' is created on the machine with the NFS partition containing its home
- The NFS partition is mounted on other machines on the cluster
- The same 'mpiuser' with same gid and uid are created on all the machines, with the same home folder location as earlier.
- After logging in as 'mpiuser', SSH key is generated and added to authorized_keys file
- All the desired machines for communication are added into known_hosts
- Both MVAPICH2 [5] and openmpi [6] versions supporting CUDA-aware MPI are downloaded, built and installed on the home folder

5 Plots for MPI

Using the methods discussed in Section **??**, performance evaluation is done for all the cases. The data obtained is then plotted using MATLAB for performance analysis.

The flow for our test code in MPI involves :

1. Broadcast B matrix to all nodes.
2. Send parts of A from master process (process 0) to other processes (within same node and also to another node).
3. Computation assigned to each process
4. Send back part od C computed at each process to master process.

Three out of four steps in the code involves communication. We have recorded timings of each of these steps to evaluate the performance for different combinations of methods of communication, matrix size, number of processes and different implementations of MPI. The plots are made with fixed number of processes, for varying size of matrices to evaluate performance of different methods of communication in parallel programming and different implementations of MPI which is our main goal.

Figure 1 plots give timings for entire code. Figure 2 plots give time taken to broadcast B. Figure 3 plots give time taken to transfer parts of A. Last and final set of plots, Figure 4 give the time taken to transfer back computed matrix C.

For bandwidth test of CUDA-Aware MPI and CUDA over MPI, data of sizes from 1 Byte to 4 MB are communicated between the GPUs. For each process, 100 iterations are taken and the mean time is used to calculate the bandwidth. Figure 5 plots compare their performances on same and different nodes.

For matrix multiplication using CUDA over MPI, we have split matrices A and B to 2 equal parts A_0, A_1 and B_0, B_1. They are used to calculate 4 parts of matrix C : C_{00}, C_{01}, C_{10}, C_{11}, by sending pair (A_0,B_0), (A_0,B_1),(A_1,B_0), (A_1,B_1) to each of the 4 processes. Figure 6 plots compare performances on same and different nodes for total time for execution, send-receive time for matrix A and B, and send-receive time for matrix C.

5.1 Observations and Inferences

From Figure 1, 2, 3, and 4, it can be inferred that:

1. MPI run over InfiniBand communication takes far lesser time than Ethernet based communication. This is obvious as InfiniBand has higher data rates compared to Ethernet. This performance boost can be significantly seen in OpenMPI but not so much in MVAPICH2.

2. Communication to other processes within the same machine is faster compared to communication over Ethernet. Generally, interprocess communication is very fast compared to Ethernet communication. There is a significant difference in the time taken for these two methods of communication that can be observed in the plots.

3. The time taken for intra node inter process communications is comparable to the time taken for InfiniBand communication. InfiniBand provides very high data rates of upto 40Gbps. For smaller size of matrix i.e lesser number of bytes to be transferred, these speeds are comparable to the inter process communication. But for larger size of matrix, intra node inter process communication may perform significantly better

4. With increase in the number of processes and fixed matrix size, the computation time per node decreases; but the time taken for communication remains almost the same. This is because, the major communication is the time taken to broadcast B which remains same as the matrix size is not changed. There may be slight variations in time due to the change in the fraction of A, and C transfered between devices.

5. For a fixed number of processes, as matrix size increases, both computation time and communication time increases almost linearly. This is because, as the size of matrix increases, the number of computations per node/process increases. Also the number of bytes of data to be transmitted from/to a process also increases.

6. Comparing the two implementations of MPI, we see that OpenMPI has a better performance compared to MVAPICH2. Also we see that in case of MVAPICH2, there is no much difference in performance when we switch from Ethernet to InfiniBand for communication. The reason still unclear, may also be a configuration issue in the cluster.

In Figure 5, it can be seen that, CUDA-Aware MPI out performs CUDA over MPI, for both test cases. There is a performance difference of 10 times for messages of length 4 MB for the same node test case. This is because of the overcoming of overhead provided by CUDA over MPI, where the host to device and device to host communication becomes an overhead as well as a bottle neck. This overhead is not there for the CUDA-Aware MPI case, where communication happens directly between the two memory locations of the same GPU. For the second test case, the 1GB network of the cluster becomes a bottleneck, resulting the saturation of speed. Also, for CUDA over MPI, the multiple read and write onto the host buffer and further communication congestion results in poorer performance compared to CUDA-Aware MPI.

From Figure 6 plots, it can be inferred that

1. For send-receive of A and B, it takes more time over the 1GbE network for the different node case compared to same node case.

2. The matrix multiplication (using CUDA over MPI) performance for 4 processes on same node is poorer compared to 4 processes distributed over two nodes for larger matrix sizes, even though performance is similar for smaller matrices.

3. For the transfer of C matrix, more time is taken with in same node for higher matrix sizes. This is because, 1.2 GB memory is required per process for 10000x10000 matrix multiplication and for 4 process on same node, allocated space goes beyond 4 GB RAM. Thus, some part of C matrix gets stored in SWAP memory, resulting in slow down of data transfer.

,

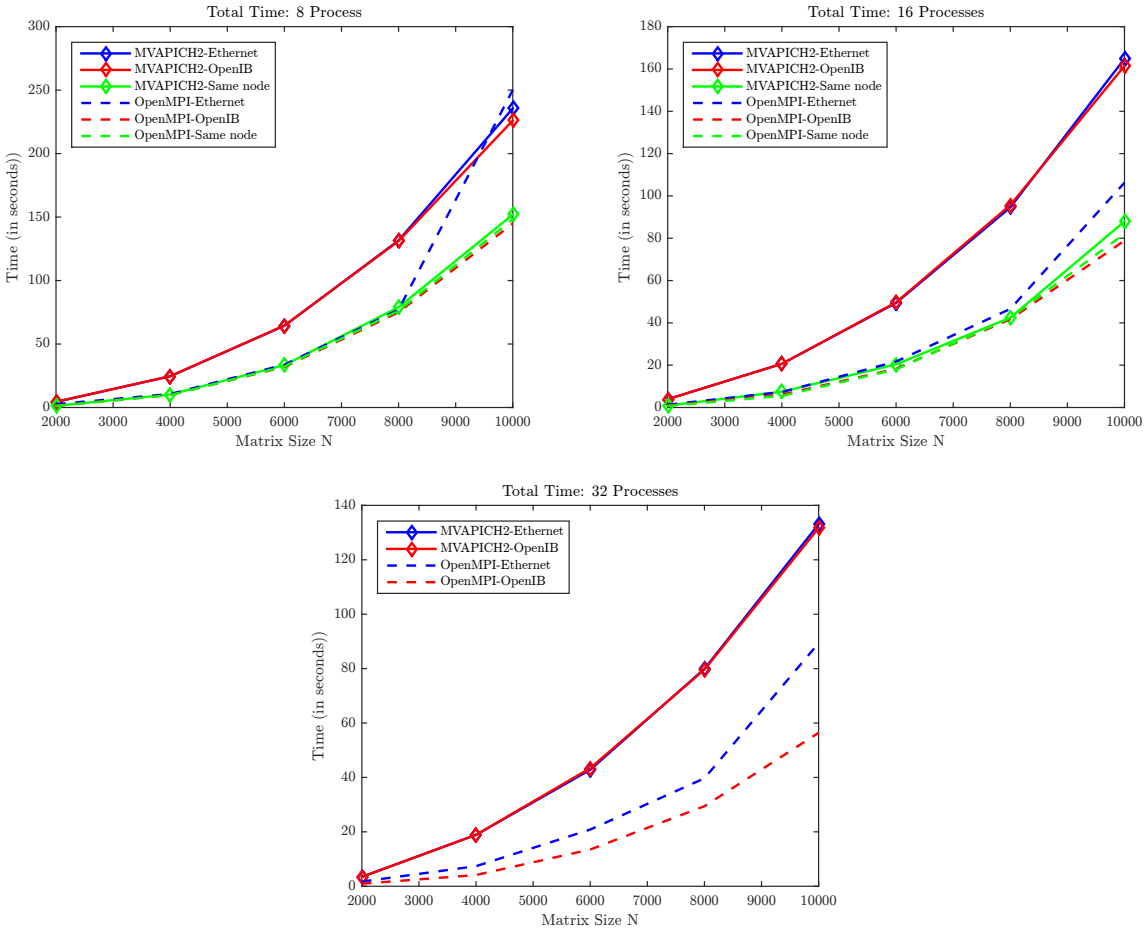

Figure 1: Comparison of Total Time taken for 8, 16 and 32 processes

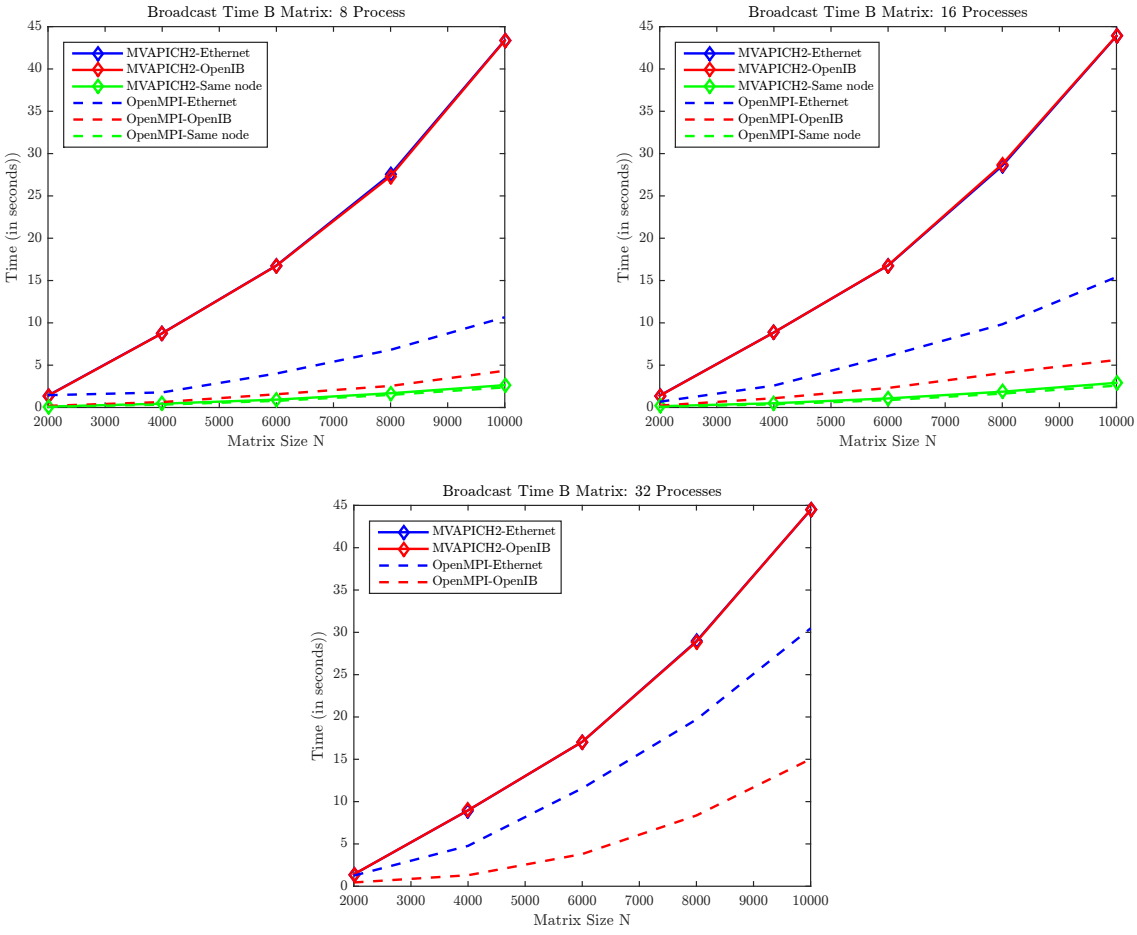

Figure 2: Comparison of Broadcast Time taken for matrix B for 8, 16 and 32 processes

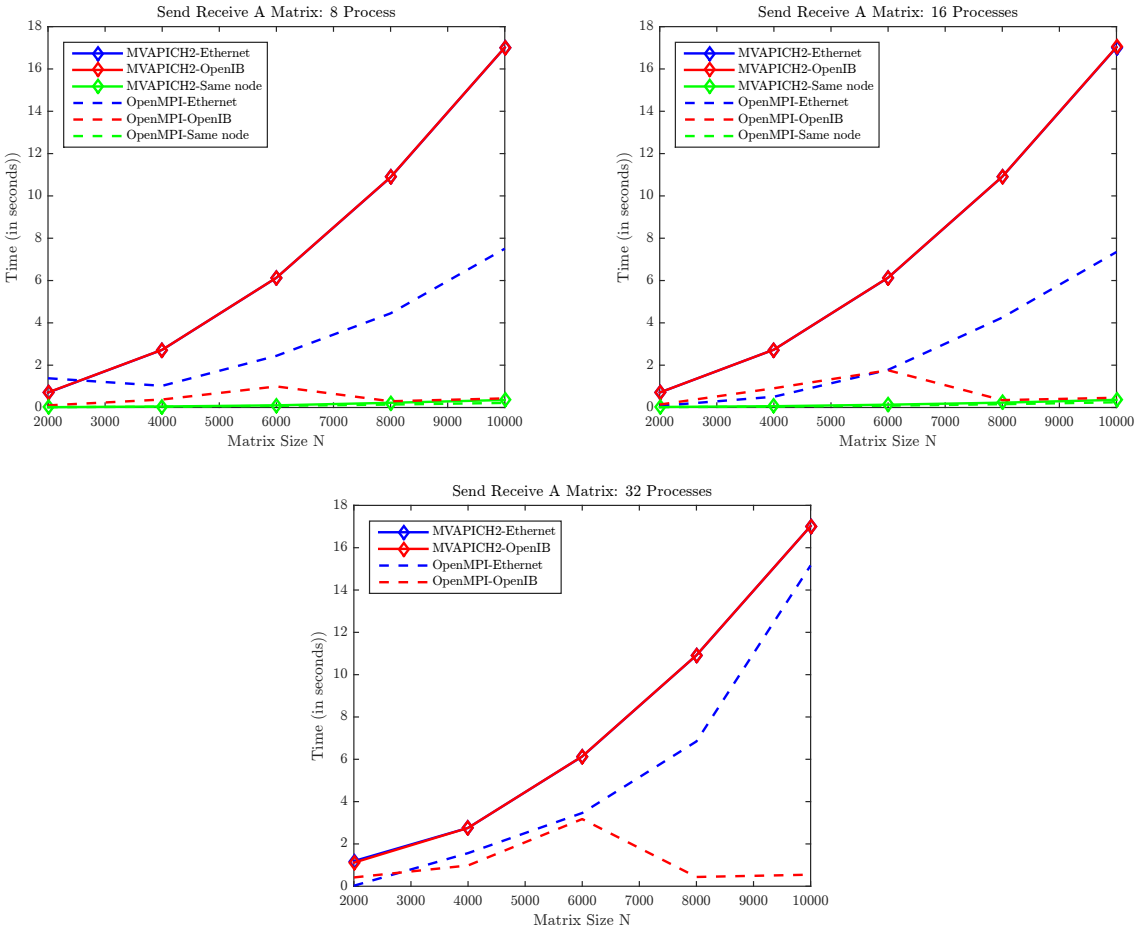

Figure 3: Comparison of Send-Receive Time taken for matrix A for 8, 16 and 32 processes

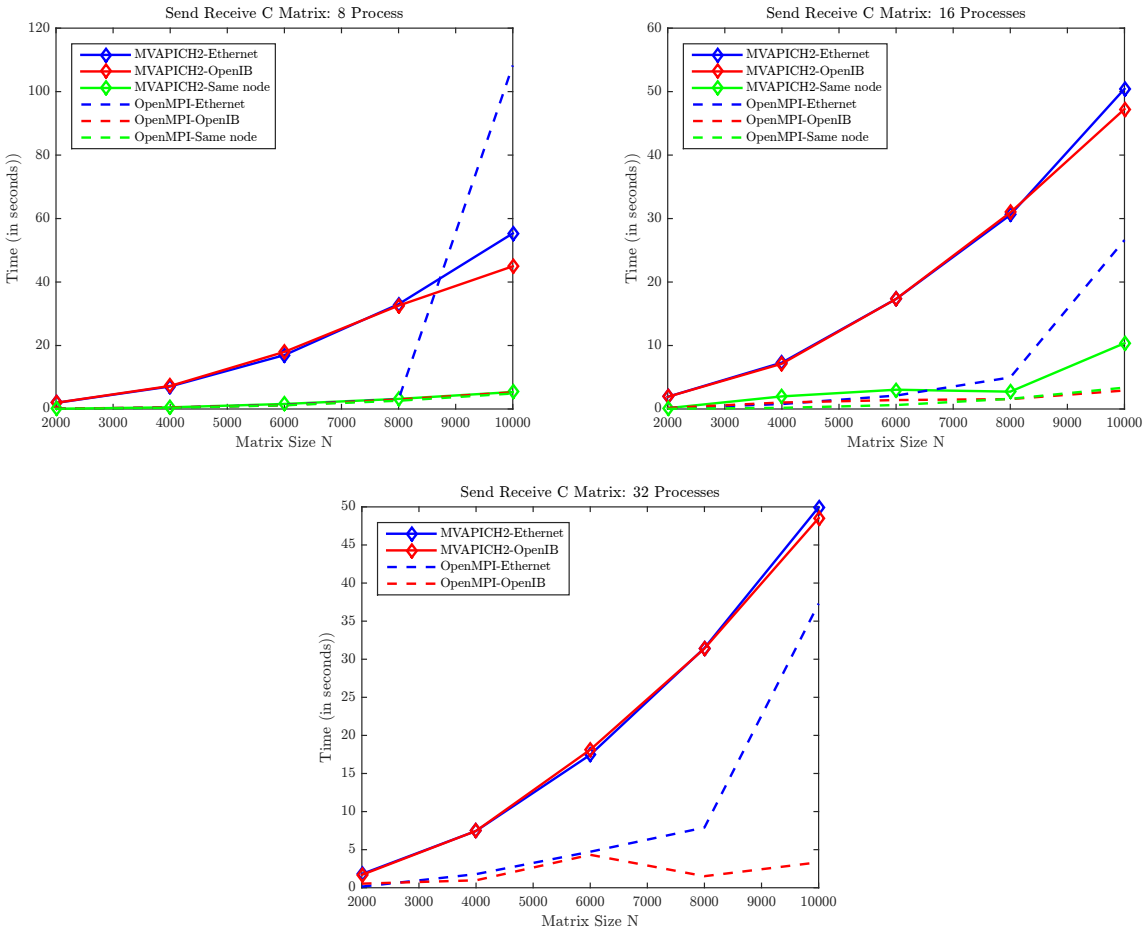

Figure 4: Comparison of Send-Receive Time taken for matrix C for 8, 16 and 32 processes

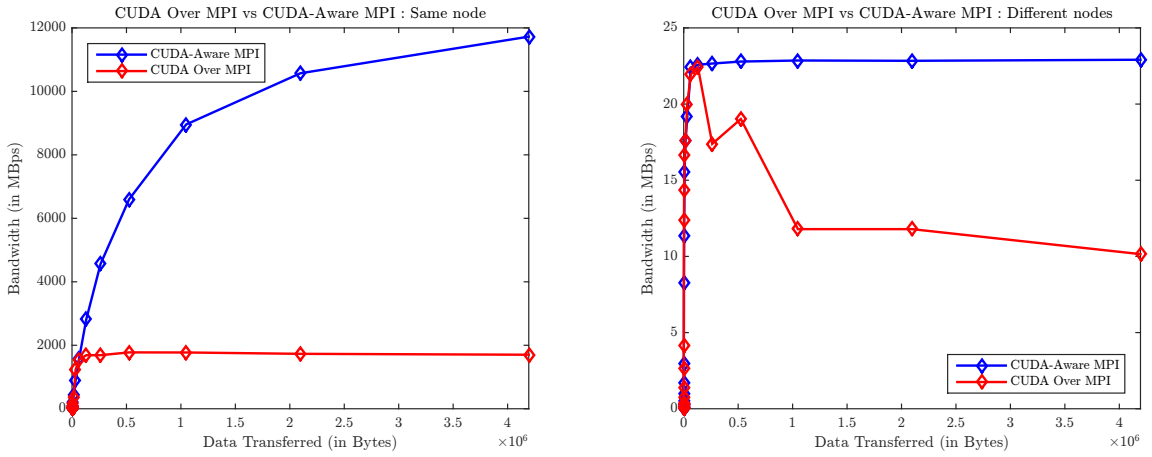

Figure 5: Comparison of CUDA-Aware and CUDA over MPI for same and different nodes

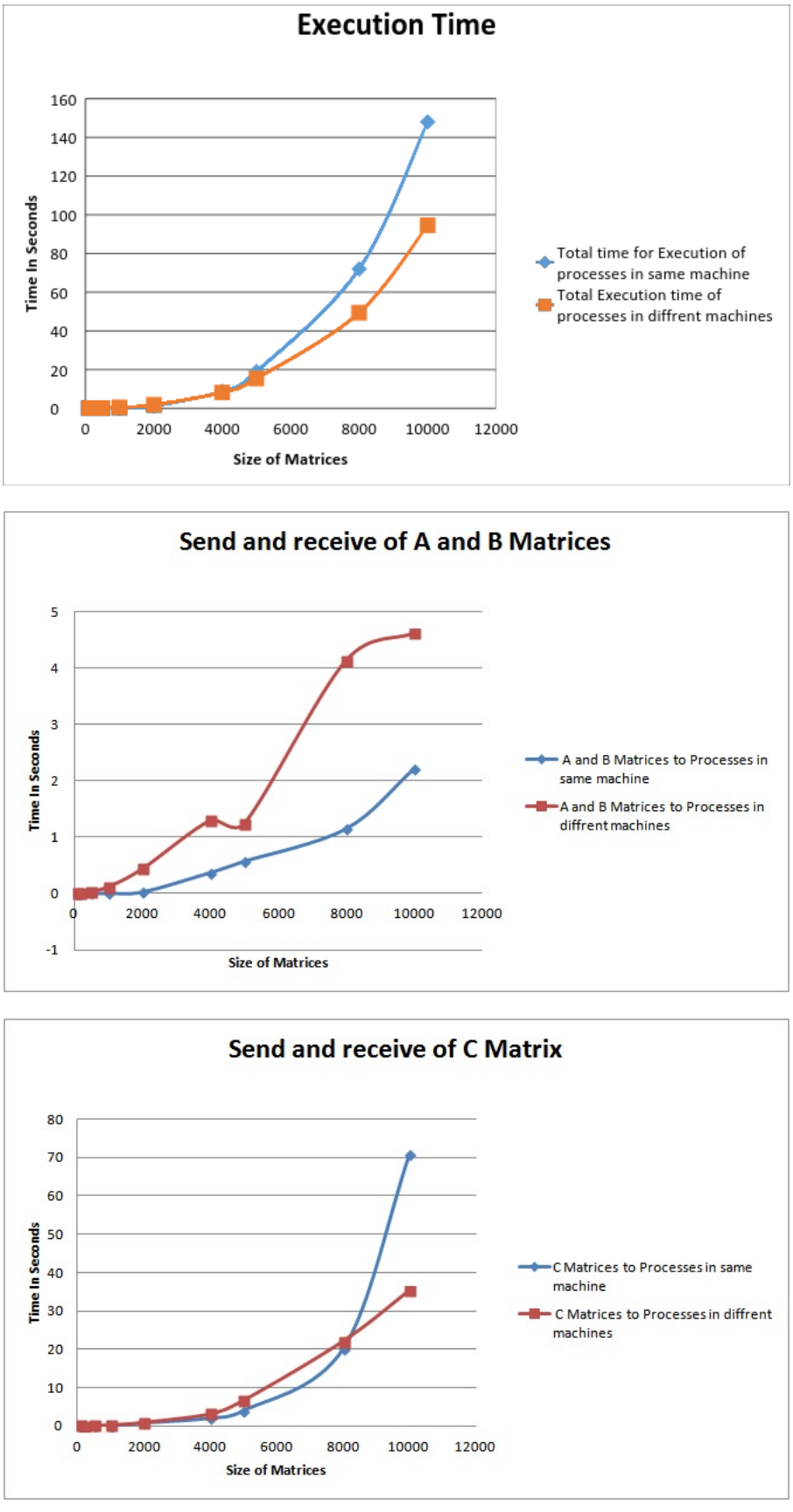

Figure 6: Comparison of Timings for CUDA Over MPI Matrix Multiplication

References

[1] J. Cheng, M. Grossman, and T. McKercher. *Professional CUDA C Programming*. Wrox : Programmer to Programmer. Wiley, 2014.

[2] Gregory F Pfister. An introduction to the infiniband architecture.

[3] File based partition, https://www.debuntu.org/how-to-create-a-filesystem-within-another-partitions-file

[4] NFS Setup, https://www.digitalocean.com/community/tutorials/how-to-set-up-an-nfs-mount-on-ubuntu-12-04

[5] MVAPICH2-2.1, http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.1-userguide.html

[6] OpenMPI, https://www.open-mpi.org/faq/?category=buildcuda

[7] CUDA-Aware MPI, https://devblogs.nvidia.com/parallelforall/introduction-cuda-aware-mpi/