High Performance Computing (DD2356)

Assignment 3: MPI Programming

Akhil YERRAPRAGADA(akhily@kth.se)
Gibson CHIKAFA(chikafa@kth.se)

Code Available at https://github.com/gibchikafa/mpi_programming.git

Exercise One

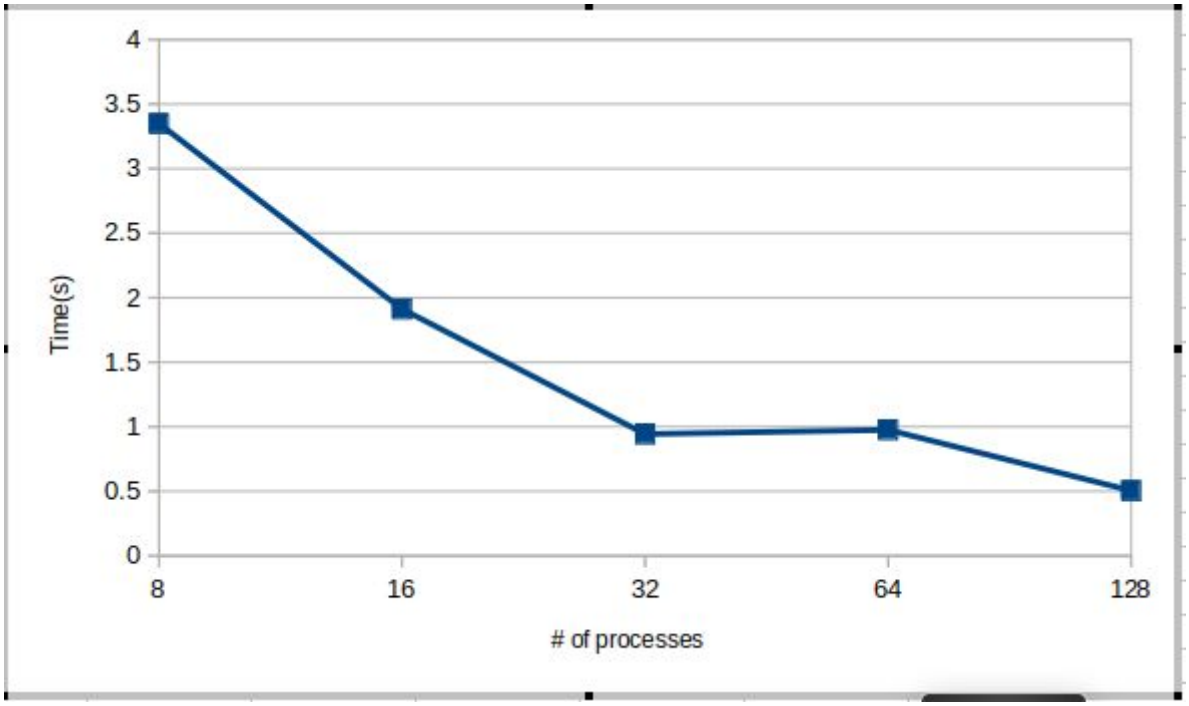
Question 2

i. cc hello_world_mpi.c -o hello_world_mpi.out.
ii. srun -n <number of processes> ./hello_world_mpi.out.
iii. By changing the number of processes parameter in the command srun -n <number of processes> ./hello_world_mpi.out.
iv. MPI_Comm_rank for retrieving rank. MPI_Comm_size for getting the total number of processes.
v. OpenMPI and MPICH

Exercise Two

Question 2.1

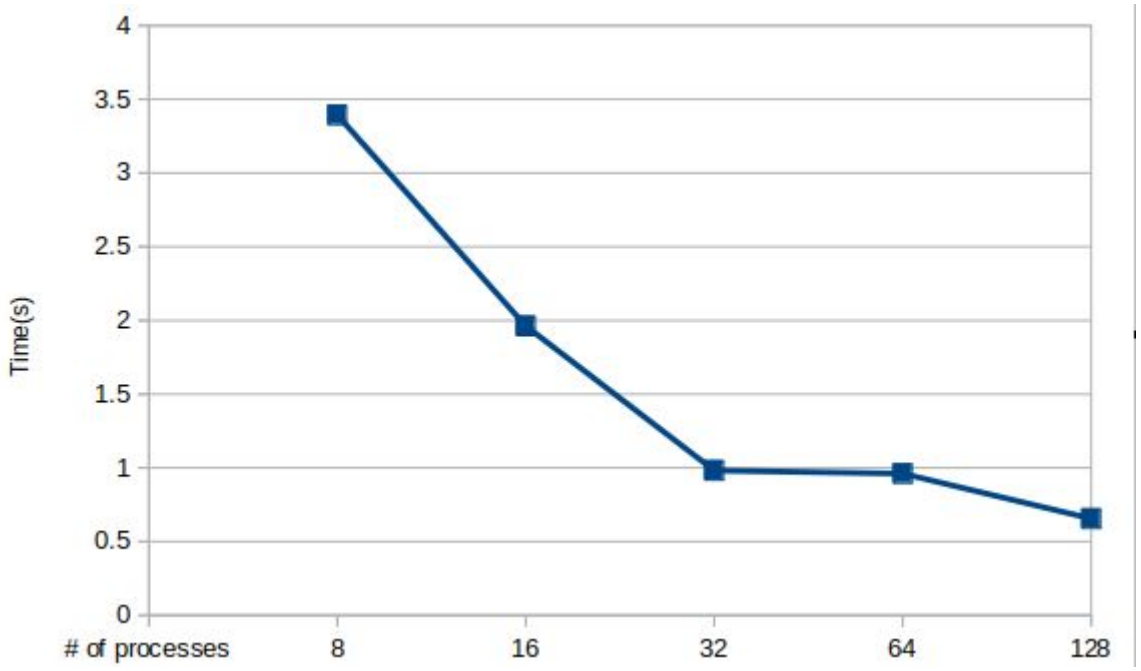
i.



ii. For MPI_Recv it means that the message has been received into the buffer specified by the buffer argument. For MPI_Send it means that the message envelope has been created and the message has been sent or that the contents of the message have been copied into s system buffer.
iii. MPI_Wtime()

Question 2.2

i.

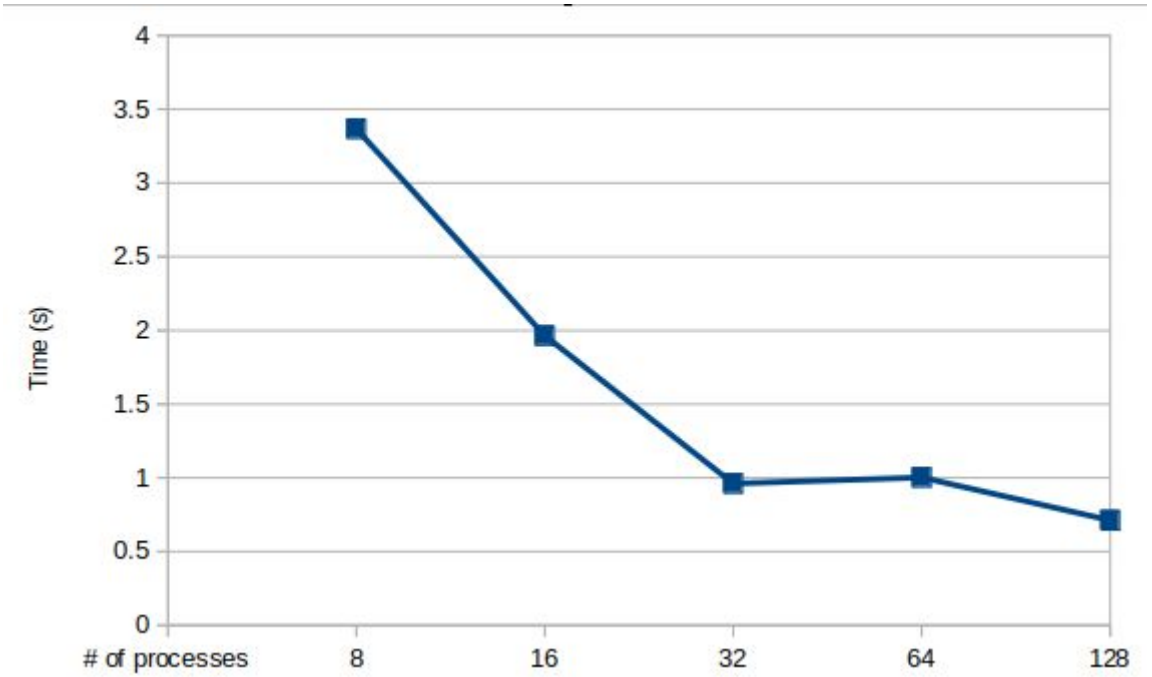


ii. The Linear Reduction Algorithm performs better

iii. For a large number of processes the Linear Reduction Algorithm will perform better because less messages are sent between processes. In linear each process sends directly to the root while in binary tree message goes through multiple processes.

Question 2.3

i.

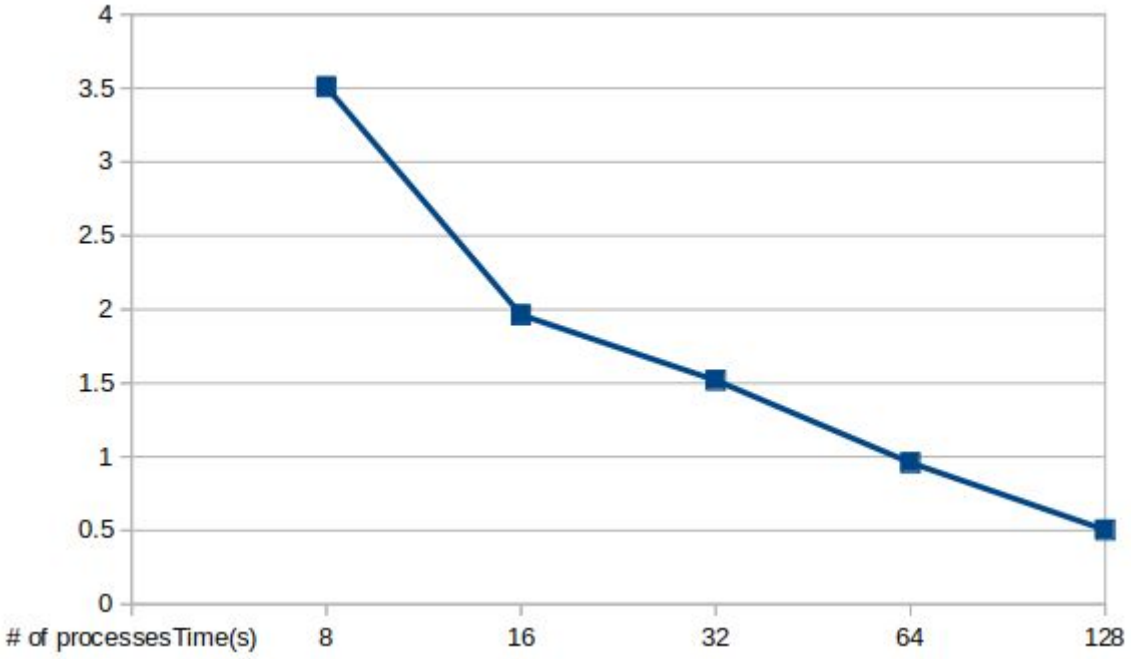


ii. MPI_Irecv(*buf, count, datatype, source, tag, comm, *request, ierror)
and MPI_Isend(*buf, count, datatype, destination, tag, comm, *request). The I in MPI_Irecv stands for “immediate” i.e they return immediately.

iii. The blocking performs better.

Question 2.4

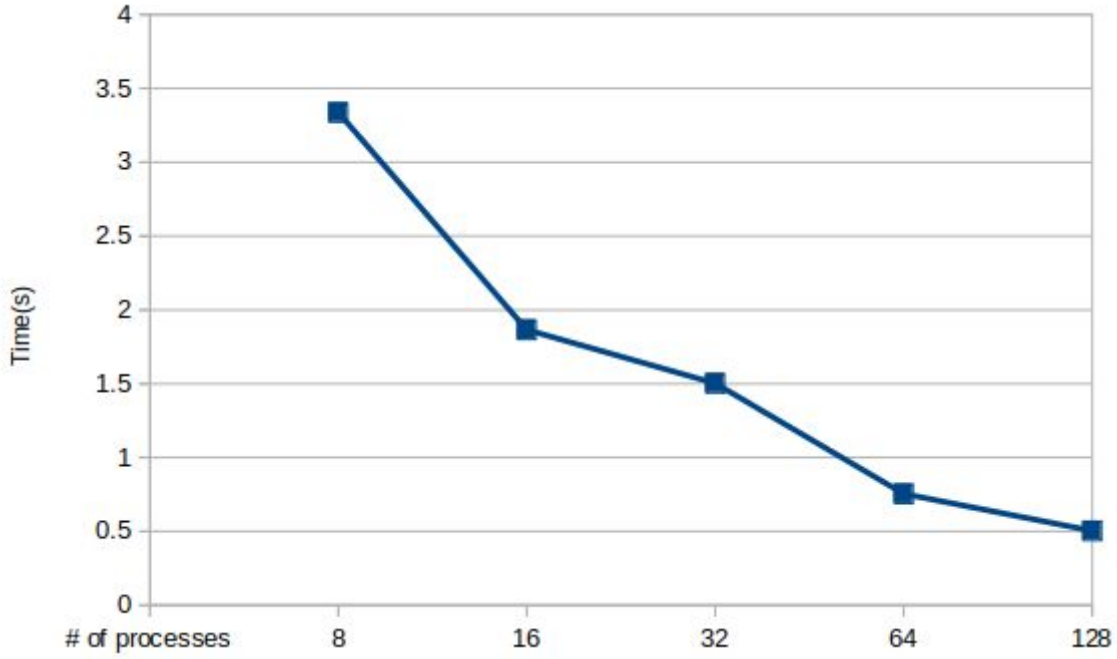
i.



ii. MPI_Bcast function e.g MPI_Bcast(&pi, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD)

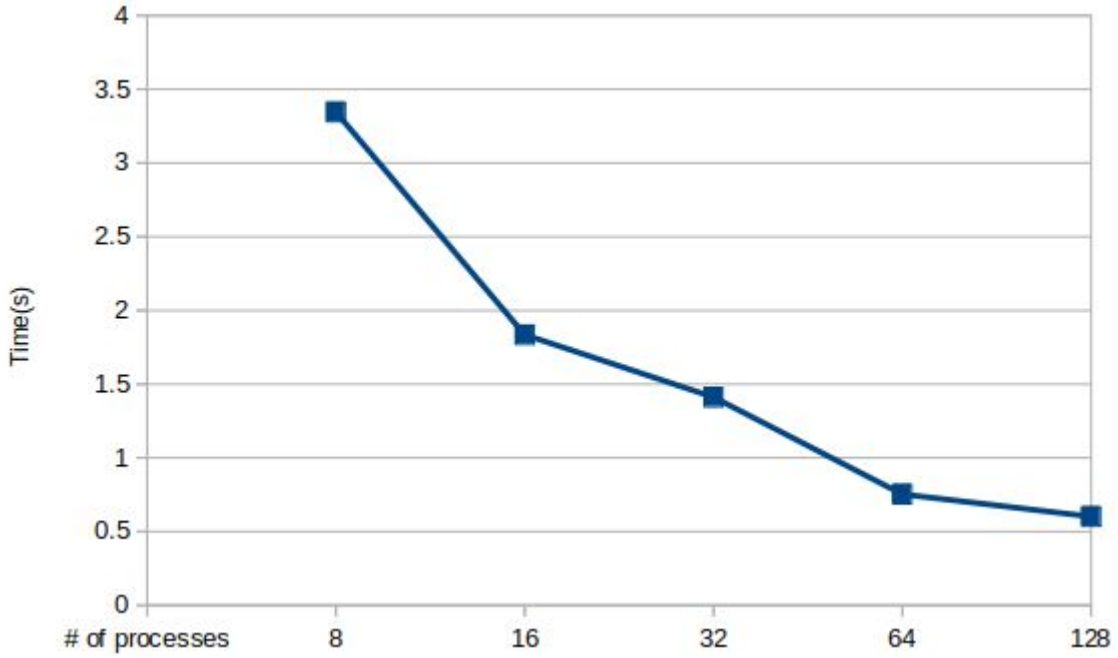
Question 2.5

i.

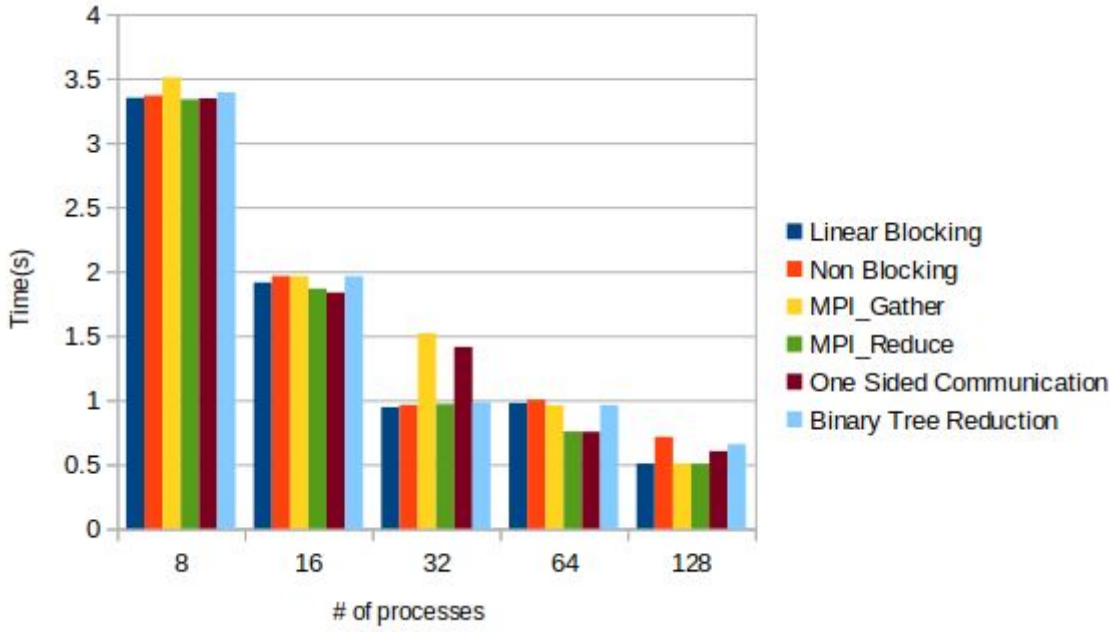


Question 2.6

i.



ii.



iii.

The MPI_Reduce performs better overall. This is because the process communication and global sums computation are implemented internally and are highly optimized. In the other approaches we are doing them manually which might not be very optimized.

The reduction operations use the following algorithms: binomial, pipelined, and pipelined binary tree algorithms [1]

Exercise three

1) **Intranode communication evaluation results for ping-pong**

8	0.000000691
16	0.000000732
32	0.000000796
64	0.000000877
128	0.000000863
256	0.000000887
512	0.000000947
1024	0.000001073
2048	0.000001330
4096	0.000001893
8192	0.000004275
16384	0.000005896
32768	0.000008330
65536	0.000015199
131072	0.000031476
262144	0.000061224
524288	0.000120764
1048576	0.000239751
2097152	0.000479314
4194304	0.000963235
8388608	0.001921356
16777216	0.003796802
33554432	0.007487705
67108864	0.015140233
134217728	0.030445054
268435456	0.061046889
536870912	0.122291410
1073741824	0.246415493

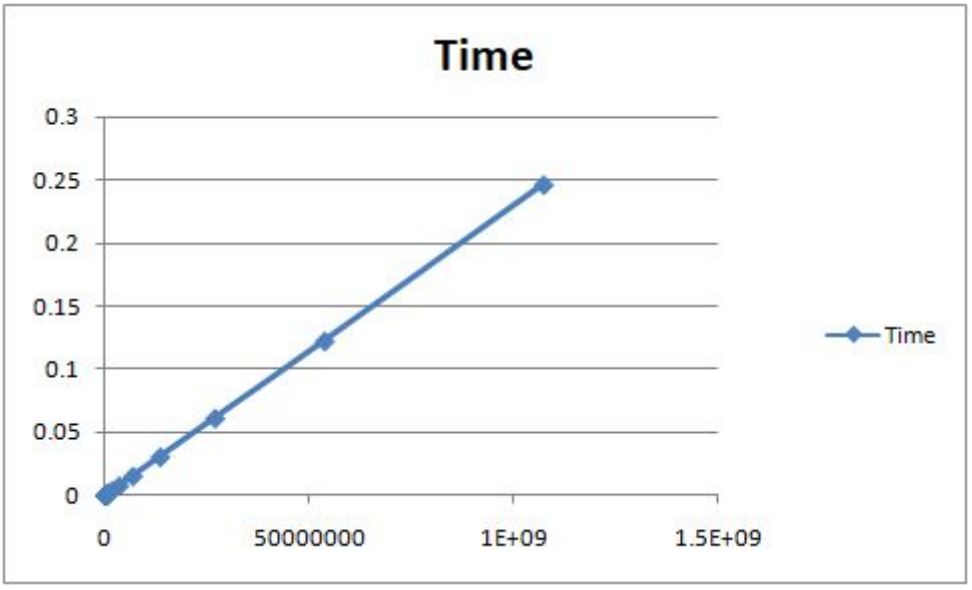
1) **Internode communication evaluation results for ping-pong**

8	0.000001423
16	0.000001471
32	0.000001571
64	0.000001562
128	0.000001414
256	0.000001450
512	0.000001526
1024	0.000002031
2048	0.000002408
4096	0.000003428
8192	0.000004761
16384	0.000005784
32768	0.000007770
65536	0.000011704
131072	0.000019596
262144	0.000035017
524288	0.000065570
1048576	0.000127184
2097152	0.000260832
4194304	0.000510094
8388608	0.001095867
16777216	0.002264237
33554432	0.004630070

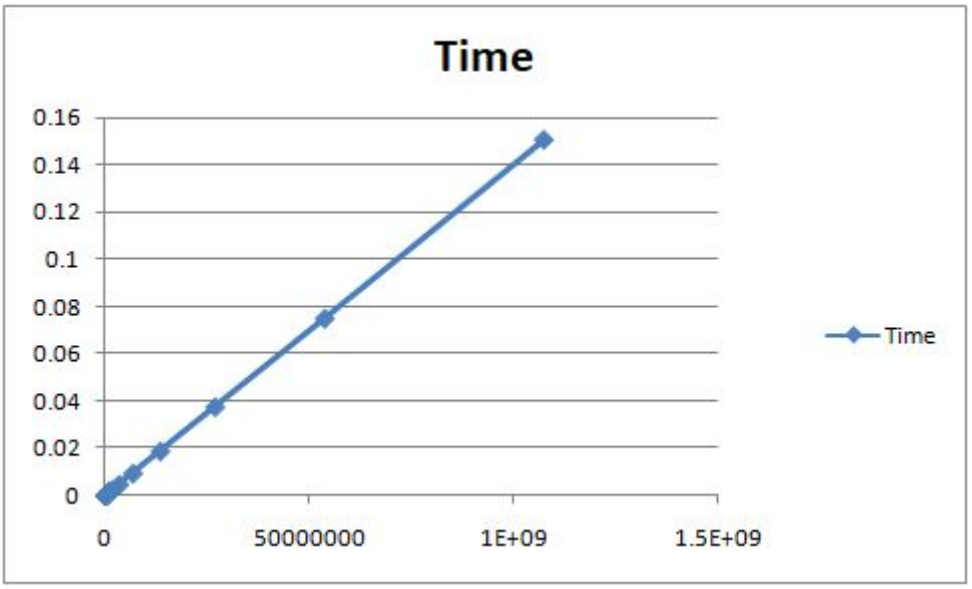
67108864	0.009325495
134217728	0.018707871
268435456	0.037425880
536870912	0.074812799
1073741824	0.150397635

2) Plot time vs messages

Intranode



Internode



3) **BestFit when calculated in python**

Latency for intranode

p2 = s = -5.40852443e-05

Not Possible since latency is negative.

Bandwidth for intranode

p1 = 2.29102385e-10

1/p1/1e9 GB/s = 4.36 GB/s

Latency for internode

p2 = s = 6.72777023e-6

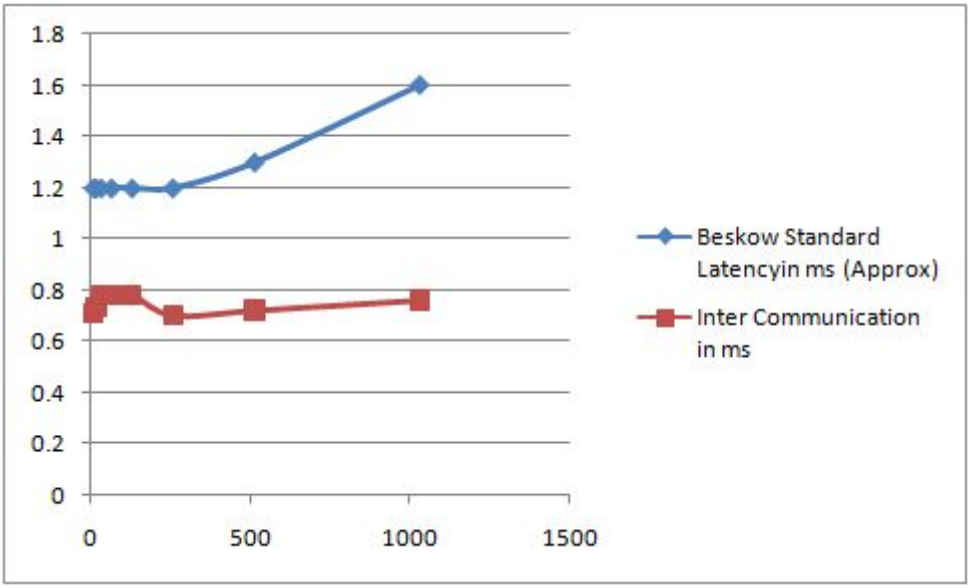
s/1e-6 ms = 6.73 ms

Bandwidth for internode

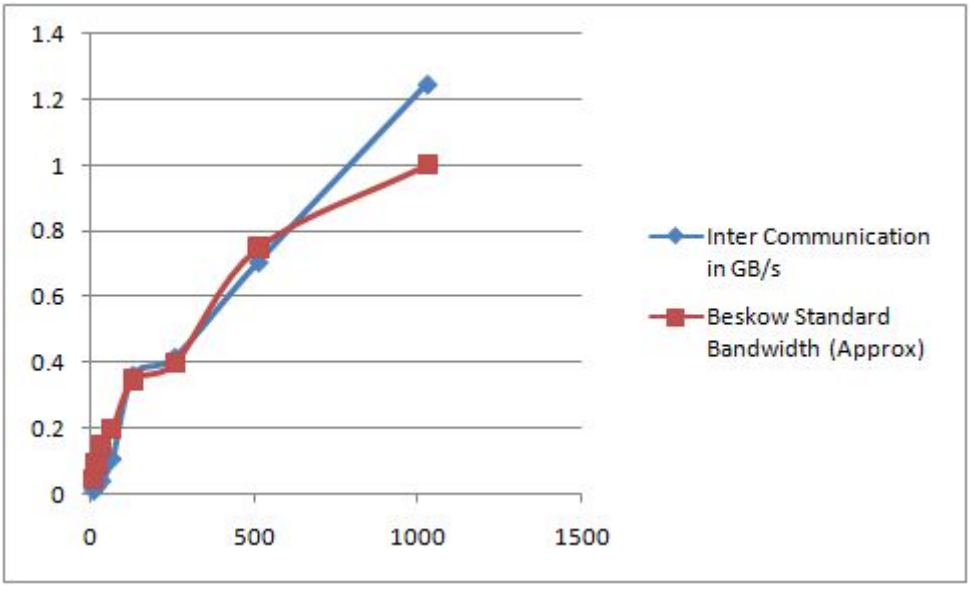
p1 = 1.13988e-10

1/p1/1e9= 8.77 GB/s

4) **Latency Calculated vs Beskow stats**



Bandwidth calculated vs Beskow stats



No major difference spotted in bandwidth and latency when compared evaluation results with https://www.cray.com/sites/default/files/resources/CrayXCNetwork.pdf .

References

[1] *Understanding MPI Reduction Algorithms*, 2020 (accessed 19th April, 2020). https://devmesh.intel.com/projects/understanding-mpi-reduction-algorithms