1)

Implement the hot potato test (as taught in class and shown in lecture notes) for measuring communication time using MPI on the Stampede. Use N number of processors and S data size (your program should run with any N and any S). Use 100 repetition to increase accuracy. Measure the communication time, using N = 0, 4, 8 and 16 and S = 64 integers, and draw a curve. Repeat the same and draw another curve using N = 0, 4, 8 and 16 and S = 512 integers. Submit your source code and data curves.

#include "mpi.h"

#include <stdio.h>

int main (int argc, char \*\*argv)

{

int i, rank, size, sz=64,dest;

int msg[512];

double start\_time, end, time\_diff, finalTime;

double commTime[128],sum[128], overhead;

for (i=0; i<512; i++)

{

msg[i] = 2;

}

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank );

MPI\_Comm\_size( MPI\_COMM\_WORLD, &size );

sum[0]=0;

for (i=0; i<100; i++)

{

if (rank == 0)

{

time\_diff = MPI\_Wtime();

start\_time = MPI\_Wtime();

MPI\_Send(msg, sz, MPI\_CHAR,(rand()%(size-1))+1, 1, MPI\_COMM\_WORLD);

MPI\_Recv(msg, sz, MPI\_CHAR, MPI\_ANY\_SOURCE, 1, MPI\_COMM\_WORLD,MPI\_STATUS\_IGNORE);

end = MPI\_Wtime();

overhead = start\_time - time\_diff ;

finalTime = end - start\_time - overhead ;

commTime[i] = finalTime / size;

sum[i+1]=sum[i]+commTime[i];

}

else

{

MPI\_Recv(msg, sz, MPI\_CHAR,MPI\_ANY\_SOURCE ,1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

MPI\_Send(msg, sz, MPI\_CHAR,0, 1, MPI\_COMM\_WORLD);

}

}

printf("Average time=%f\n",sum[i-1]/100);

MPI\_Finalize();

return 0;

}

NOTE: The value of N has to be changed in the batch file and the value of S has to altered in the code in order to get the respective results.

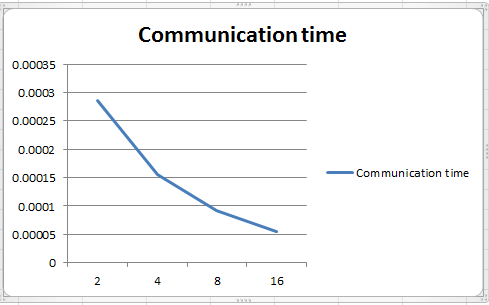
OUTPUT TABLE:

|  |  |  |
| --- | --- | --- |
| N | S | Communication Time |
| 2 | 64 | 0.000286 |
| 4 | 64 | 0.000155 |
| 8 | 64 | 0.000091 |
| 16 | 64 | 0.000054 |
| 2 | 512 | 0.000254 |
| 4 | 512 | 0.000160 |
| 8 | 512 | 0.000106 |
| 16 | 512 | 0.000059 |

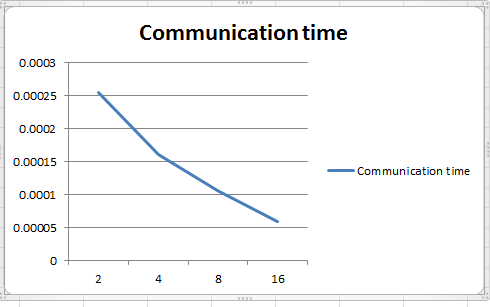
**Graphs**

**Number of Processors in x axis and Communication time in y axis**

For size = 64 integers.



For size =512 integers.



2)

Implement the program for measuring the all-to-all exchange collection communication operation (from the MPI library) using MPI on the Stampede. Use N number of processors and S data size (your program should run with any N and any S). Use 100 repetition to increase accuracy. Submit your source code and data curves. Measure the collective communication time, using N = 0, 4, 8 and 16 and S = 64 integers, and draw a curve. Repeat the same and draw another curve using N = 0, 4, 8 and 16 and S = 128 integers.

#include "mpi.h"

#include <stdio.h>

int main (int argc, char \*\*argv)

{

int i, rank, size, sz=64;

int msg[128];

int msg1[128];

double start, finish, time\_diff, finalTime;

double commTime[128],sum[128], overhead;

for (i=0; i<128; i++)

{

msg[i] = 2;

}

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank );

MPI\_Comm\_size( MPI\_COMM\_WORLD, &size );

sum[0]=0;

for (i=0; i<100; i++)

{

time\_diff = MPI\_Wtime();

start = MPI\_Wtime();

MPI\_Alltoall(msg,8,MPI\_INT,msg1,8,MPI\_INT,MPI\_COMM\_WORLD);

finish = MPI\_Wtime();

overhead = start - time\_diff ;

finalTime = finish - start - overhead ;

commTime[i] = finalTime / size ;

sum[i+1]=sum[i]+commTime[i];

}

MPI\_Finalize();

printf("Average time =%f\n",sum[i-1]/100);

return 0;

}

Output table:

|  |  |  |
| --- | --- | --- |
| N | S | Collective communication time |
| 2 | 64 | 0.000340 |
| 4 | 64 | 0.000193 |
| 8 | 64 | 0.000165 |
| 16 | 64 | 0.000066 |
| 2 | 128 | 0.000359 |
| 4 | 128 | 0.000178 |
| 8 | 128 | 0.000102 |
| 16 | 128 | 0.000064 |

Note :

The maximum value of the output corresponding to the number of processors is considered, which is according to the collective communication algorithm.

The value of N has to be changed in the batch file and the value of S has to altered in the code in order to get the respective results. Also the value of the 2nd and the 5th parameter of MPI\_ALLtoall function has to be calculated and placed using the formula = (Size/Number of processors) corresponding to processors and the size used in order to get the respective results.

OUTPUT: Maximum of communication times of n local processors are found at first and are highlighted in Red colour.

**Output obtained for N=2 and S= 64**

Average time =0.000340

Average time =0.000016

**Output obtained for N=4 and S= 64**

Average time =0.000193

Average time =0.000056

Average time =0.000042

Average time =0.000052

**Output obtained for N=8 and S= 64**

Average time =0.000165

Average time =0.000103

Average time =0.000092

Average time =0.000097

Average time =0.000092

Average time =0.000095

Average time =0.000095

Average time =0.000023

**Output obtained for N=16 and S= 64**

Average time =0.000066

Average time =0.000030

Average time =0.000012

Average time =0.000029

Average time =0.000032

Average time =0.000031

Average time =0.000033

Average time =0.000028

Average time =0.000030

Average time =0.000027

Average time =0.000028

Average time =0.000027

Average time =0.000029

Average time =0.000030

Average time =0.000033

Average time =0.000033

**Output obtained for N=2 and S= 128**

Average time =0.000359

Average time =0.000074

**Output obtained for N=4 and S= 128**

Average time =0.000178

Average time =0.000033

Average time =0.000041

Average time =0.000029

**Output obtained for N=8 and S= 128**

Average time =0.000102

Average time =0.000024

Average time =0.000025

Average time =0.000034

Average time =0.000036

Average time =0.000039

Average time =0.000037

Average time =0.000027

**Output obtained for N=16 and S= 128**

Average time =0.000064

Average time =0.000029

Average time =0.000028

Average time =0.000026

Average time =0.000030

Average time =0.000032

Average time =0.000030

Average time =0.000037

Average time =0.000028

Average time =0.000027

Average time =0.000027

Average time =0.000029

Average time =0.000026

Average time =0.000003

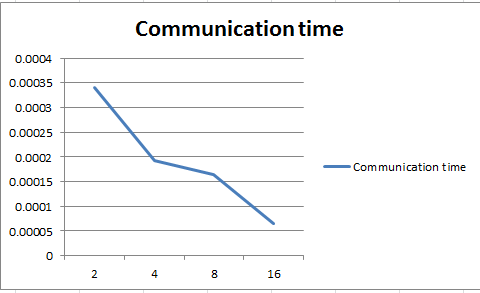
Average time =0.000028

Average time =0.000023

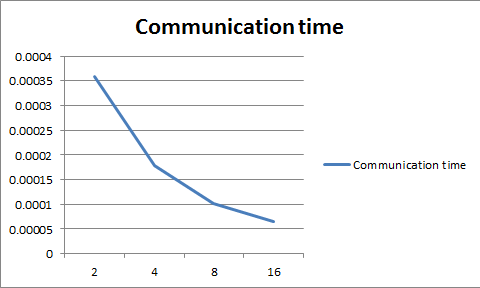
**Graphs**

**Number of Processors in x axis and Communication time in y axis**

For s= 64 integers.



For s=128 integers



**Batch file:**

#!/bin/bash

#SBATCH -J p1 # job name

#SBATCH -o p1.o # output and error file name (%j expands to jobID)

#SBATCH -n 16 # total number of mpi tasks requested

#SBATCH -N 16 # total number of mpi tasks requested

#SBATCH -p development # queue (partition) -- normal, development, etc.

#SBATCH -t 00:05:00 # run time (hh:mm:ss) - 1.5 hours

#SBATCH --mail-user=cxh4131@tacc.utexas.edu

#SBATCH --mail-type=begin # email me when the job starts

#SBATCH --mail-type=end # email me when the job finishes

ibrun p1 # run the MPI executable named a.out

~

**Running Steps:**

1. Compiling -> mpicc –o filename filename.c
2. Sbatch batchfilename.txt
3. Cat output.o