CS633: Assignment 1

Kajal Deep: 200483
Kumar Arpit: 200532

1 Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"
int main( int argc, char *argv[])
{
    int myrank, size;
    double start_time, time, max_time;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank( MPI_COMM_WORLD, &myrank );
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    if(size!=2)
    {
        printf("Run the code for 2 processes. Exiting.\n");
        return 1;
    }
    //argv[1] is the size of data to be sent
    int BUFSIZE = (atoi(argv[1])/sizeof(int));
    int sendarr[BUFSIZE];
    MPI_Status status;
    int recvarr[BUFSIZE];
    //Tnitializing the array for the sending process.
    if(!myrank)
    {
        for(int i=0; i<BUFSIZE; i++)
            sendarr[i]=i;
    }
    //Sending from rank 0 to rank 1
    start_time = MPI_Wtime ();
    if (!myrank)
        MPI_Send(sendarr, BUFSIZE, MPI_INT, 1, 0, MPI_COMM_WORLD);
    else
        MPI_Recv(recvarr, BUFSIZE, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
    time = MPI_Wtime () - start_time;
    MPI_Reduce (&time, &max_time, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);
    if (!myrank)
        printf ("%lf\n", max_time);
    MPI_Finalize();
    return 0;
}
```

- The code is essentially a slightly modified version of the p2p.c file provided taking the data size to be sent as a parameter. It also includes a check to see if it has been run for exactly 2 processes or not.
- To run:

  First compile using: mpicc -o code.x code.c

  Execute as: mpirun -np 2 –hosts csews1, csews2 ./code.x datasize

2 Run Script:

```
import os
import sys
if(len(sys.argv)!=4):
    print("Usage:")
    print("python .\run.py host1 host2 datasize")
    print("Enter -1 as datasize for all required sizes")
    exit()
if(int(sys.argv[3])==-1):
    sizes=[1000,100000,1000000,8000000,32000000,64000000,128000000]
else:
    sizes=[int(sys.argv[3])]
hostpairs=[[int(sys.argv[1]),int(sys.argv[2])]]
for i in hostpairs:
    print("Hosts: csews",i[0]," csews",i[1])
    for j in sizes:
        print("DataSize: ",j)
        for k in range(0,10):
            cmd = "mpirun -np 2 -hosts csews"+str(i[0])+",csews"+str(i[1])+" ./code.x "+str(j)
            os.system(cmd)
```

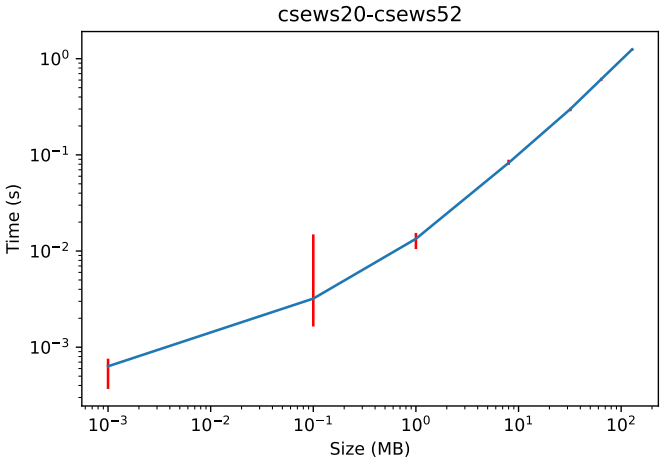
To run:
python ./run.py host1 host2 datasize
(datasize=-1 to run for all the required datasizes).

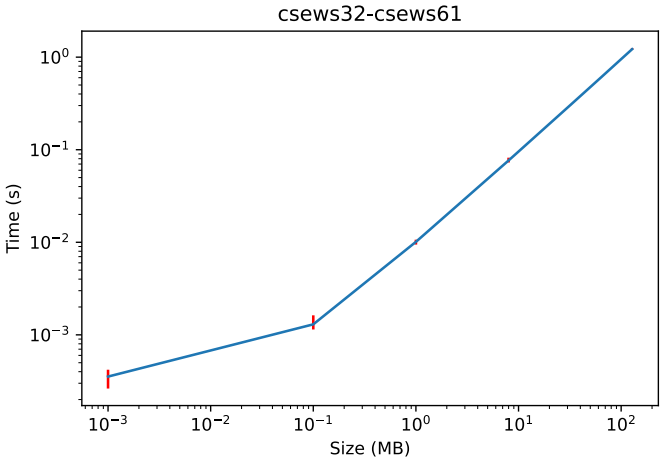
3 Experimentation methodology:

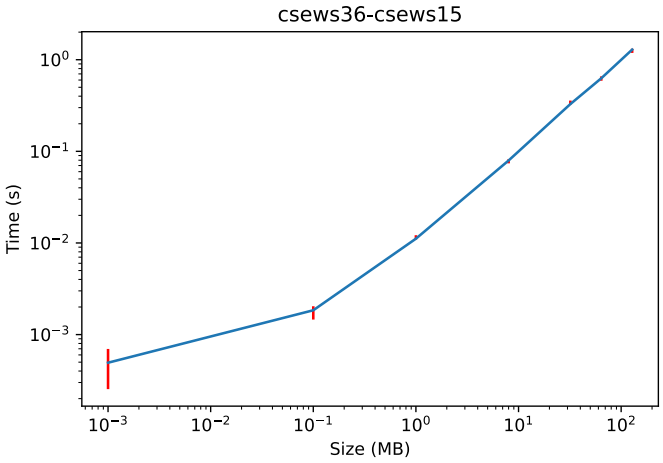
- We run the above code for 3 distinct pair of nodes, csews20-csews52, csews32-csews61 and csews36-csews51, 10 times for each pair, for each data size.
- Before recording the values of an execution, we first run a single transfer which we don't record to ensure there are no unnecessary delays since if we avoid it, the first run tends to produce slightly (10-20%) larger time than other runs.
- The observations were taken late at night when the servers should be under low load to ensure consistency. Also, if the run had some obviously off outliers, we did not consider that while preparing the plots.
- We chose the obove pair of nodes arbitrarily from the working nodes since the topology map was not available to us. For the same reason, we ran the code 10 times each on the different pairs.
- We plot the observed values (Time taken against size) for each pair distinctly using matplotlib library from python. We choose to plot the values on a log-log scale since the data sizes increased exponentially. We note that the linear nature of a graph is preserved on log-log scale and only the axes get scaled.

4 Plots:

The following are the observed Time(s) vs Size(MB) plots on a log-log scale for the 3 pairs of nodes.







5 Observations:

- We observe from the above plots that the time grows almost linearly with size for each node pairs for data sizes from 100KB.
- However, the behaviour for the small data sizes, in particular 1KB is different. We also plotted a TIME/SIZE vs SIZE graph (not shown above) for better insights.
- From the above graph, we see that the datapoints for sizes from 1MB almost lie on a single line. This is expected as for the larger data size, the startup time is negligible in comparison to the total word transfer time which grows linearly with datasize assuming the Bandwidth remains constant.
- The TIME for 1KB data size in particular is much off compared to others (by around an order of magnitude) because the startup time is the major part of the actual communication time. Also, the effective bandwith is low,
- Similarly, 100KB observations are also slightly off line from the other, larger observations but by a much shorter margin.
- The deviations in TIME relative to the SIZE between multiple runs is also much much larger for the smaller data sizes. This is again because of the stratup time dominating.
- The error bars, however, in general were negligbly small especially for the larger datas, possibly suggesting a stable network.
- Also, the plots and the values themselves are much similar for each pair of nodes.