```c
/*
 * Brandon Bell
 * csci4576
 * hw5 9-21-2016
 *
 * Homework 5: Alpha-Beta, Dense Matrix Transform.
*/

#include <stdio.h>
#include <string.h>
#include <math.h>
#include "mpi.h"

int main(int argc, char* argv[])
{
    int        my_rank;
    int        p;
    int        tag    = 0;
    double     start;
    double     end;
    int        np     = 1000;
    int        msize  = 0;
    int        maxpower = 22;
    int        maxsize = pow(2,maxpower);
    char       message[maxsize];
    double     total;
    char       name[MPI_MAX_PROCESSOR_NAME];
    int        pnamemax;
    MPI_Status status;

    // Spin-up Mpi.
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &p);
    // Get the node name for output.
    MPI_Get_processor_name(name,&pnamemax);

    // Ensure that the needed 2 processes for a ping pong test are there and no
    // more.
    if( p != 2 )
    {
        fflush(stdout);
        printf("[ %s, %d ], Please use exactly 2 processes\n", name, my_rank);
        return 1;
    }

    // Dump out each rank and the node that it's on.
    fflush(stdout);
    printf("[ %s, %d ]\n", name, my_rank);

    // Loop through a series of pingpong passes with Wtime calls on either side
    // of the loop to get the time for np passes. The Intel compiler doesn't
    // like varibale defs in the the for loop.
    int i;
    for ( i=0; i <= maxpower; i++ )
    {
        msize = pow(2,i);
        // Ensure process are synced at this point because p0 handles the output.
        MPI_Barrier(MPI_COMM_WORLD);
        // Have p0 start timing
        if( my_rank == 0 )
            start = MPI_Wtime();
        int j;
        for( j=0; j < np; j++ )
        {
            if( my_rank == 0 )
            {
                MPI_Send( &message, msize, MPI_CHAR, 1, tag, MPI_COMM_WORLD );
                MPI_Recv( &message, msize, MPI_CHAR, 1, tag, MPI_COMM_WORLD, &status );
            }
            else
            {
                MPI_Recv( &message, msize, MPI_CHAR, 0, tag, MPI_COMM_WORLD, &status );
                MPI_Send( &message, msize, MPI_CHAR, 0, tag, MPI_COMM_WORLD );
            }
        }
        // p0 ends timeing, does calculations and dumps output for the np runs.
        if( my_rank == 0 )
        {
            end = MPI_Wtime();

            // calculate average message time for np messages, then the 1/2 round
            // trip time.
            total = (end - start);
            total = total / np;
            total = total / 2;

            // Output the timing to stdout.
            printf("[ %s, %d ], %5.10f, %5.10f, %1.15f, %7d\n", name, my_rank, start, end, t
otal, msize);
        }
    }

    // Close up.
    MPI_Finalize();
    return 0;
}
```

```c
/*
 * Brandon Bell
 * csci4576
 * hw5 9-21-2016
 *
 * Homework 5: Alpha-Beta, Dense Matrix Transform.
*/

#include <stdio.h>
#include <string.h>
#include <math.h>
#include "mpi.h"

int main(int argc, char* argv[])
{
    int        my_rank;
    int        p;
    int        tag    = 0;
    double     start;
    double     end;
    int        np     = 100;
    int        msize  = 0;
    int        maxpower = 14;
    int        maxsize = pow(2,maxpower);
    char       message[maxsize];
    double     total;
    char       name[MPI_MAX_PROCESSOR_NAME];
    int        pnamemax;
    MPI_Status status;

    // Spin-up Mpi.
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &p);
    // Get the node name for output.
    MPI_Get_processor_name(name,&pnamemax);

    // Ensure that the needed 2 processes for a ping pong test are there and no
    // more.
    if( p != 2 )
    {
        fflush(stdout);
        printf("[ %s, %d ], Please use exactly 2 processes\n", name, my_rank);
        return 1;
    }

    // Dump out each rank and the node that it's on.
    fflush(stdout);
    printf("[ %s, %d ]\n", name, my_rank);

    // Loop through a series of pingpong passes with Wtime calls on either side
    // of the loop to get the time for np passes. The Intel compiler doesn't
    // like varibale defs in the the for loop.
    int i;
    for ( i=0; i <= maxsize; i++ )
    {
        msize = i;
        // Ensure process are synced at this point because p0 handles the output.
        MPI_Barrier(MPI_COMM_WORLD);
        // Have p0 start timing
        if( my_rank == 0 )
            start = MPI_Wtime();
        int j;
        for( j=0; j < np; j++ )
        {
            if( my_rank == 0 )
            {
                MPI_Send( &message, msize, MPI_CHAR, 1, tag, MPI_COMM_WORLD );
                MPI_Recv( &message, msize, MPI_CHAR, 1, tag, MPI_COMM_WORLD, &status );
            }
            else
            {
                MPI_Recv( &message, msize, MPI_CHAR, 0, tag, MPI_COMM_WORLD, &status );
                MPI_Send( &message, msize, MPI_CHAR, 0, tag, MPI_COMM_WORLD );
            }
        }
        // p0 ends timeing, does calculations and dumps output for the np runs.
        if( my_rank == 0 )
        {
            end = MPI_Wtime();

            // calculate average message time for np messages, then the 1/2 round
            // trip time.
            total = (end - start);
            total = total / np;
            total = total / 2;

            // Output the timing to stdout.
            printf("[ %s, %d ], %5.10f, %5.10f, %1.15f, %7d\n", name, my_rank, start, end, t
otal, msize);
        }
    }

    // Close up.
    MPI_Finalize();
    return 0;
}
```

```c
/*
 * Brandon Bell
 * csci4576
 * hw5 9-21-2016
 *
 * Homework 5: Alpha-Beta, Dense Matrix Transform.
 * Part 5: Dense Matrix Transform.
*/

#include <stdio.h>
#include <string.h>
#include <math.h>
#include "mpi.h"

int main(int argc, char* argv[])
{
    int        my_rank;
    int        p;
    int        tag     = 0;
    int        n       = 5;
    MPI_Status status;

    // Spin-up Mpi.
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &p);

    if ( my_rank == 0 )
    {
        // Define an upper trianglur array of 1's on P0.
        int        a[n][n];
        for( int i=0; i <= n; i++ )
        {
            for( int j=0; j <= n; j++ )
            {
               if ( j < i )
                   a[i][j] = 1;
                else
                    a[i][j] = 0;
            }
        }

        // Print the matrix.
        for( int i=0; i <= n; i++ )
        {
            printf("| ");
            for( int j=0; j <= n; j++ )
            {
                printf("%d ", a[i][j]);
            }
            printf("|\n");
        }
    }

    // Close-Up MPI.
    MPI_Finalize();
}
```

```c
/*
 * Brandon Bell
 * csci4576
 * hw5 9-21-2016
 *
 * Homework 5: Alpha-Beta, Dense Matrix Transform.
*/

#include <stdio.h>
#include <string.h>
#include <math.h>
#include "mpi.h"

int main(int argc, char* argv[])
{
    int        my_rank;
    int        p;
    int        tag    = 0;
    double     start;
    double     end;
    int        np     = 1000;
    int        msize  = 0;
    int        maxpower = 22;
    int        maxsize = pow(2,maxpower);
    char       message[maxsize];
    double     total;
    char       name[MPI_MAX_PROCESSOR_NAME];
    int        pnamemax;
    MPI_Status status;

    // Spin-up Mpi.
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &p);
    // Get the node name for output.
    MPI_Get_processor_name(name,&pnamemax);

    // Ensure that the needed 2 processes for a ping pong test are there and no
    // more.
    if( p != 2 )
    {
        fflush(stdout);
        printf("[ %s, %d ], Please use exactly 2 processes\n", name, my_rank);
        return 1;
    }

    // Dump out each rank and the node that it's on.
    fflush(stdout);
    printf("[ %s, %d ]\n", name, my_rank);

    // Loop through a series of pingpong passes with Wtime calls on either side
    // of the loop to get the time for np passes.
    int i;
    for ( i=0; i <= maxpower; i++ )
    {
        msize = pow(2,i);
        // Ensure process are synced at this point because p0 handles the output.
        MPI_Barrier(MPI_COMM_WORLD);
        // Have p0 start timing
        if( my_rank == 0 )
            start = MPI_Wtime();

        int j;
        for( j=0; j < np; j++ )
        {
            if( my_rank == 0 )
            {
                MPI_Send( &message, msize, MPI_CHAR, 1, tag, MPI_COMM_WORLD );
                MPI_Recv( &message, msize, MPI_CHAR, 1, tag, MPI_COMM_WORLD, &status );
            }
            else
            {
                MPI_Recv( &message, msize, MPI_CHAR, 0, tag, MPI_COMM_WORLD, &status );
                MPI_Send( &message, msize, MPI_CHAR, 0, tag, MPI_COMM_WORLD );
            }
        }
        // p0 ends timeing, does calculations and dumps output for the np runs.
        if( my_rank == 0 )
        {
            end = MPI_Wtime();

            // calculate average message time for np messages, then the 1/2 round
            // trip time.
            total = (end - start);
            total = total / np;
            total = total / 2;

            // Output the timing to stdout.
            printf("[ %s, %d ], %5.10f, %5.10f, %1.15f, %7d\n", name, my_rank, start, end, t
otal, msize);
        }
    }

    // Close up.
    MPI_Finalize();
    return 0;
}
```

```
import numpy as np
import matplotlib.pyplot as plt


cld = "PingPong-Large-Double-Node.4342248.comet-27-13.out"
cls = "PingPong-Large-Single-Node.4342220.comet-10-51.out"
csd = "PingPong-Small-Double-Node.4345654.comet-18-29.out"

sld = "PingPong-Large-Double-Node.7635285.c557-401.out"
sls = "PingPong-Large-Single-Node.7635263.c558-402.out"
ssd = "PingPong-Small-Double-Node.7635426.c557-502.out"

fname = "test.txt"
datacld= np.genfromtxt(cld,delimiter=",",usecols=(4,5),skip_header=2)
datacls= np.genfromtxt(cls,delimiter=",",usecols=(4,5),skip_header=2)
datacsd= np.genfromtxt(csd,delimiter=",",usecols=(4,5),skip_header=2)
datasld= np.genfromtxt(sld,delimiter=",",usecols=(4,5),skip_header=2)
datasls= np.genfromtxt(sls,delimiter=",",usecols=(4,5),skip_header=2)
datassd= np.genfromtxt(ssd,delimiter=",",usecols=(4,5),skip_header=2)

fig = plt.figure(figsize=(8,3))
datasls[:,1] = datasls[:,1] / ( 2 ** 10 )
datasls[:,0] = datasls[:,0] / ( 10 ** (-6) )
plt.plot(datasls[:,1],datasls[:,0])

plt.title("Stampede 2 Nodes 0kb - 4MB")
plt.xlabel('Message Size ( kb )')
plt.ylabel(r'Time ( $\mu s$ )')
plt.xlim((datasls[0,1],datasls[-1,1]))
plt.ticklabel_format(style='plain')
plt.xscale('log')
plt.yscale('log')

plt.savefig('sls.eps',bbox_inches='tight', dpi=300)
plt.show()
```