# ASSIGNMENT 2

**Topic:** Parallelizing Wave Diffusion with MPI and OpenMP
**Student Name:** Pooja Nadagouda

## Implementation:

Wave2D_template is the sequential program where t=1 and t>=2 is implemented using Schrodinger's wave formulae.

Wave2D_template_mpi is the parallelized version of sequential program. Here parallelization is done using MPI and OpenMP. For t>=2 the calculations are divided among 6 processors. Processors from csmpi1h to cssmpi6h, master being cssmpi1h. All processors hold z for t=0 and t=1. But for t>=2, matrix is divided into horizontal strips and assigned among all 6 processors. If the matrix size can't be equally divided between processors, then all processors with rank less than remainder receive (1+strip size), so that all the remainder is divided equally between all processors instead of assigning entire remainder to one processor.

To calculate z, Schrodinger's wave formula requires information of height of its neighboring cells in all 4 directions. To compute values at the top and lower boundary, each processor required information of neighboring cell. To do this the processor and its neighbor exchange the data with each other. To further parallelize and increase the performance, OpenMP is used. The for loop used for computation for t=1 and t>=2 is parallelized using 0 – 4 threads.

## Performance calculations:

Time taken for computation using 1 thread and 1 processor:
Time taken for computation using 1 thread and 6 processors:
Time taken for computation using 1 thread and 6 processors:

- *** Performance = 2460478 / 854027 = 2.88**
- *** Performance = 2460478 / 642833 = 3.82**

Video of sequential Wave 2D program execution using wout is attached in the zip folder – **(demo)**

## Output screenshots:

```
Last login: Tue Oct 25 10:27:43 2022 from 10.102.100.227
[poojan26@cssmpi1h ~]$ CSSmpdboot
Preparing...
Starting Master's mpd Process...
Starting node: cssmpi2h.uwb.edu
Starting node: cssmpi3h.uwb.edu
Starting node: cssmpi4h.uwb.edu
Starting node: cssmpi5h.uwb.edu
Starting node: cssmpi6h.uwb.edu
Cluster built:
cssmpi1h.uwb.edu_35751 (10.158.82.61)
cssmpi6h.uwb.edu_39671 (10.158.82.66)
cssmpi5h.uwb.edu_39643 (10.158.82.65)
cssmpi4h.uwb.edu_33617 (10.158.82.64)
cssmpi3h.uwb.edu_45451 (10.158.82.63)
cssmpi2h.uwb.edu_46204 (10.158.82.62)
CSSmpdboot finished!
[poojan26@cssmpi1h ~]$ cd my_prog2/
[poojan26@cssmpi1h my_prog2]$ ls
compile.sh             out1.txt   Wave2D                   Wout.class
measurements2019.txt   out6.txt   Wave2D_mpi               Wout.java
measurements2020.txt   pout.txt   Wave2D_mpi.cpp
mpd.hosts              Timer.cpp  Wave2D_template.cpp
mpi_setup.txt          Timer.h    Wave2D_template_mpi.cpp
[poojan26@cssmpi1h my_prog2]$ ./compile.sh
[poojan26@cssmpi1h my_prog2]$ ./Wave2D 100 500 10 > out1.txt
Elapsed time = 249178
[poojan26@cssmpi1h my_prog2]$ mpirun -np 6 ./Wave2D_mpi 100 500 10 4 > out6.txt
rank[0]'s range 0~16
rank[1]'s range 17~33
rank[2]'s range 34~50
rank[3]'s range 51~67
rank[4]'s range 68~83
rank[5]'s range 84~99
Elapsed time = 1770760
[poojan26@cssmpi1h my_prog2]$ diff out1.txt out6.txt
[poojan26@cssmpi1h my_prog2]$ mpirun -np 1 ./Wave2D_mpi 588 500 0 1
rank[0]'s range 0~587
Elapsed time = 2460478
[poojan26@cssmpi1h my_prog2]$ mpirun -np 1 ./Wave2D_mpi 588 500 0 2
rank[0]'s range 0~587
Elapsed time = 1398516
[poojan26@cssmpi1h my_prog2]$ mpirun -np 1 ./Wave2D_mpi 588 500 0 3
rank[0]'s range 0~587
Elapsed time = 1588850
[poojan26@cssmpi1h my_prog2]$ mpirun -np 1 ./Wave2D_mpi 588 500 0 4
rank[0]'s range 0~587
Elapsed time = 1363410
[poojan26@cssmpi1h my_prog2]$ mpirun -np 2 ./Wave2D_mpi 588 500 0 1
rank[0]'s range 0~293
rank[1]'s range 294~587
Elapsed time = 1714329
[poojan26@cssmpi1h my_prog2]$ mpirun -np 2 ./Wave2D_mpi 588 500 0 2
rank[0]'s range 0~293
rank[1]'s range 294~587
Elapsed time = 1269583
[poojan26@cssmpi1h my_prog2]$ mpirun -np 2 ./Wave2D_mpi 588 500 0 3
```

```
[poojan26@cssmpi1h my_prog2]$ mpirun -np 2 ./Wave2D_mpi 588 500 0 3
rank[0]'s range 0~293
rank[1]'s range 294~587
Elapsed time = 1205320
[poojan26@cssmpi1h my_prog2]$ mpirun -np 2 ./Wave2D_mpi 588 500 0 4
rank[0]'s range 0~293
rank[1]'s range 294~587
Elapsed time = 1094968
[poojan26@cssmpi1h my_prog2]$ mpirun -np 3 ./Wave2D_mpi 588 500 0 1
rank[0]'s range 0~195
rank[1]'s range 196~391
rank[2]'s range 392~587
Elapsed time = 1321773
[poojan26@cssmpi1h my_prog2]$ mpirun -np 3 ./Wave2D_mpi 588 500 0 2
rank[0]'s range 0~195
rank[1]'s range 196~391
rank[2]'s range 392~587
Elapsed time = 867534
[poojan26@cssmpi1h my_prog2]$ mpirun -np 3 ./Wave2D_mpi 588 500 0 3
rank[0]'s range 0~195
rank[1]'s range 196~391
rank[2]'s range 392~587
Elapsed time = 943498
[poojan26@cssmpi1h my_prog2]$ mpirun -np 3 ./Wave2D_mpi 588 500 0 4
rank[0]'s range 0~195
rank[1]'s range 196~391
rank[2]'s range 392~587
Elapsed time = 1052583
[poojan26@cssmpi1h my_prog2]$ mpirun -np 4 ./Wave2D_mpi 588 500 0 1
rank[0]'s range 0~146
rank[1]'s range 147~293
rank[2]'s range 294~440
rank[3]'s range 441~587
Elapsed time = 1065376
[poojan26@cssmpi1h my_prog2]$ mpirun -np 4 ./Wave2D_mpi 588 500 0 2
rank[0]'s range 0~146
rank[1]'s range 147~293
rank[2]'s range 294~440
rank[3]'s range 441~587
Elapsed time = 664969
[poojan26@cssmpi1h my_prog2]$ mpirun -np 4 ./Wave2D_mpi 588 500 0 3
rank[0]'s range 0~146
rank[1]'s range 147~293
rank[2]'s range 294~440
rank[3]'s range 441~587
Elapsed time = 781010
[poojan26@cssmpi1h my_prog2]$ mpirun -np 4 ./Wave2D_mpi 588 500 0 4
rank[0]'s range 0~146
rank[1]'s range 147~293
rank[2]'s range 294~440
rank[3]'s range 441~587
Elapsed time = 929024
[poojan26@cssmpi1h my_prog2]$ mpirun -np 5 ./Wave2D_mpi 588 500 0 1
rank[0]'s range 0~117
rank[1]'s range 118~235
rank[2]'s range 236~353
rank[3]'s range 354~470
rank[4]'s range 471~587
Elapsed time = 1032181
```

```
[poojan26@cssmpi1h my_prog2]$ mpirun -np 5 ./Wave2D_mpi 588 500 0 2
rank[0]'s range 0~117
rank[1]'s range 118~235
rank[2]'s range 236~353
rank[3]'s range 354~470
rank[4]'s range 471~587
Elapsed time = 625952
[poojan26@cssmpi1h my_prog2]$ mpirun -np 5 ./Wave2D_mpi 588 500 0 3
rank[0]'s range 0~117
rank[1]'s range 118~235
rank[2]'s range 236~353
rank[3]'s range 354~470
rank[4]'s range 471~587
Elapsed time = 690411
[poojan26@cssmpi1h my_prog2]$ mpirun -np 5 ./Wave2D_mpi 588 500 0 4
rank[0]'s range 0~117
rank[1]'s range 118~235
rank[2]'s range 236~353
rank[3]'s range 354~470
rank[4]'s range 471~587
Elapsed time = 646721
[poojan26@cssmpi1h my_prog2]$ mpirun -np 6 ./Wave2D_mpi 588 500 0 1
rank[0]'s range 0~97
rank[1]'s range 98~195
rank[2]'s range 196~293
rank[3]'s range 294~391
rank[4]'s range 392~489
rank[5]'s range 490~587
Elapsed time = 854027
[poojan26@cssmpi1h my_prog2]$ mpirun -np 6 ./Wave2D_mpi 588 500 0 2
rank[0]'s range 0~97
rank[1]'s range 98~195
rank[2]'s range 196~293
rank[3]'s range 294~391
rank[4]'s range 392~489
rank[5]'s range 490~587
Elapsed time = 592554
[poojan26@cssmpi1h my_prog2]$ mpirun -np 6 ./Wave2D_mpi 588 500 0 3
rank[0]'s range 0~97
rank[1]'s range 98~195
rank[2]'s range 196~293
rank[3]'s range 294~391
rank[4]'s range 392~489
rank[5]'s range 490~587
Elapsed time = 885008
```

```
[poojan26@cssmpi1h my_prog2]$ mpirun -np 6 ./Wave2D_mpi 588 500 0 4
rank[0]'s range 0~97
rank[1]'s range 98~195
rank[2]'s range 196~293
rank[3]'s range 294~391
rank[4]'s range 392~489
rank[5]'s range 490~587
Elapsed time = 615487
[poojan26@cssmpi1h my_prog2]$ mpirun -np 6 ./Wave2D_mpi 588 500 0 4
rank[0]'s range 0~97
rank[1]'s range 98~195
rank[2]'s range 196~293
rank[3]'s range 294~391
rank[4]'s range 392~489
rank[5]'s range 490~587
Elapsed time = 642833
[poojan26@cssmpi1h my_prog2]$
```

## Source code Wave2D sequential:

```
#include <iostream>
#include "Timer.h"
#include <stdlib.h>   // atoi
#include <stdio.h>
#include <math.h>

int default_size = 100;  // the default system size
int defaultCellWidth = 8;
double c = 1.0;      // wave speed
double dt = 0.1;     // time quantum
double dd = 2.0;     // change in system

using namespace std;

int main( int argc, char *argv[] ) {
  int my_rank;           // used by MPI
  int mpi_size;          // number of processors

  // verify arguments
  if ( argc != 4 ) {
    cerr << "usage: Wave2D size max_time interval" << endl;
    return -1;
  }

  int size = atoi( argv[1] );
  int max_time = atoi( argv[2] );
  int interval  = atoi( argv[3] );
```

```
 if ( size < 100 || max_time < 3 || interval < 0 ) {
   cerr << "usage: Wave2D size max_time interval" << endl;
   cerr << "       where size >= 100 && time >= 3 && interval >= 0" << endl;
   return -1;
 }

 // create a simulation space
 double z[3][size][size];
 for ( int p = 0; p < 3; p++ )
   for ( int i = 0; i < size; i++ )
     for ( int j = 0; j < size; j++ )
       z[p][i][j] = 0.0; // no wave

 // start a timer
 Timer time;
 time.start( );


 // time = 0;
 // initialize the simulation space: calculate z[0][][]
 int weight = size / default_size;
 for( int i = 0; i < size; i++ ) {
   for( int j = 0; j < size; j++ ) {
     if( i > 40 * weight && i < 60 * weight  && j > 40 * weight && j < 60 * weight ) {
       z[0][i][j] = 20.0;
     } else {
       z[0][i][j] = 0.0;
     }
   }
 }

 // time = 1 calculate z[1][][] – cells not on edge - IMPLEMENT BY YOURSELF !!!
 for( int i = 1; i < size-1; i++){
   for( int j = 1; j < size-1; j++){
     z[1][i][j] = z[0][i][j] +  (c * c)/2 * ((dt * dt)/(dd * dd)) * (z[0][i+1][j] + z[0][i-1][j] + z[0][i][j+1] + z[0][i][j-1] -
4.0 * z[0][i][j]);
   }
 }

 // simulate wave diffusion from time = 2 IMPLEMENT BY YOURSELF !!!
 for ( int t = 2; t < max_time; t++ ) {
   int p = t%3;
   int q = (t+2)%3;
   int r = (t+1)%3;
   for (  int i = 1; i < size - 1; i++ ) {
     for ( int j = 1; j < size - 1; j++ ) {
       z[p][i][j] = 2.0 * z[q][i][j] - z[r][i][j] + c * c * dt * dt / ( dd * dd ) * ( z[q][i + 1][j] + z[q][i - 1][j] + z[q][i][j + 1]
+ z[q][i][j - 1] - 4.0 * z[q][i][j] );
```

```
    }
   }
   if(interval != 0 && t%interval == 0){
    cout << t << endl;
    for(int i = 0; i < size; i++){
     for(int j = 0; j < size; j++){
      cout << z[p][j][i] << " ";
     }
     cout << endl;
    }
    cout << endl;
   }
  }// end of simulation
  // finish the timer
  cerr << "Elapsed time = " << time.lap( ) << endl;
  return 0;
}
```

## Source code Wave2D parallelized MPI and OpenMP:

```
#include <iostream>
#include "Timer.h"
#include "mpi.h"
#include <stdlib.h>   // atoi
#include <stdio.h>
#include <math.h>
#include <omp.h>

int default_size = 100;  // the default system size
int defaultCellWidth = 8;
double c = 1.0;      // wave speed
double dt = 0.1;     // time quantum
double dd = 2.0;     // change in system

using namespace std;

int main( int argc, char *argv[] ) {
  int my_rank=0;           // used by MPI
  int mpi_size;          // number of processors

  // verify arguments
  if ( argc != 5 ) {
    cerr << "usage: Wave2D size max_time interval" << endl;
    return -1;
  }
```

```
int size = atoi( argv[1] );
int max_time = atoi( argv[2] );
int interval  = atoi( argv[3] );
int nThreads = atoi( argv[4] );


if ( size < 100 || max_time < 3 || interval < 0 ) {
  cerr << "usage: Wave2D size max_time interval" << endl;
  cerr << "       where size >= 100 && time >= 3 && interval >= 0" << endl;
  return -1;
}

omp_set_num_threads(nThreads);
MPI_Init(&argc, &argv); // start MPI
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &mpi_size);
MPI_Status status;

// create a simulation space
double z[3][size][size];
for ( int p = 0; p < 3; p++ )
  for ( int i = 0; i < size; i++ )
    for ( int j = 0; j < size; j++ )
      z[p][i][j] = 0.0; // no wave

// start a timer
Timer time;
time.start( );

// time = 0;
// initialize the simulation space: calculate z[0][][]
int weight = size / default_size;
for( int i = 0; i < size; i++ ) {
  for( int j = 0; j < size; j++ ) {
    if( i > 40 * weight && i < 60 * weight  &&j > 40 * weight && j < 60 * weight ) {
      z[0][i][j] = 20.0;
    } else {
      z[0][i][j] = 0.0;
    }
  }
}


// time = 1
// calculate z[1][][], cells not on edge
// IMPLEMENT BY YOURSELF !!!
#pragma omp parallel for
for( int i = 1; i < size-1; i++){
```

```
   for( int j = 1; j < size-1; j++){
     z[1][i][j] = z[0][i][j] +  (c * c)/2 * ((dt * dt)/(dd * dd)) * (z[0][i+1][j] + z[0][i-1][j] + z[0][i][j+1] + z[0][i][j-1] -
4.0 * z[0][i][j]);
    }
  }

  int remainder = size%mpi_size;
  int stripes[mpi_size];
  // assigning stripe size for each processor
  for (int i=0; i<mpi_size; i++){
   if (i<remainder){
    stripes[i] = stripe+1;
   }else{
    stripes[i] = stripe;
   }
  }

  // simulate wave diffusion from time = 2
  // IMPLEMENT BY YOURSELF !!!
  for ( int t = 2; t < max_time; t++ ) {
   int p = t%3;
   int q = (t+2)%3;
   int r = (t+1)%3;

    //for remainder case:
    if (my_rank==0){    //master exchanges only right data to (rank+1)
       if (mpi_size>1){ //checking if more than 1 processors exist
         MPI_Send(*(*(z+q) + stripes[my_rank]-1) , size, MPI_DOUBLE, my_rank+1, 0, MPI_COMM_WORLD );
//sending data of its last lower boundry
         MPI_Recv(*(*(z+q) + stripes[my_rank]), size, MPI_DOUBLE, my_rank+1, 0, MPI_COMM_WORLD,
&status ); //recieving data for its lower boundry+1
       }

    }else if (my_rank==mpi_size-1){ //last processor exchanges its left boundry data to (rank-1)
       if (mpi_size>1){ //checking if more than 1 processors exist
          MPI_Send(*(*(z+q) + (my_rank*stripes[my_rank])+remainder) , size, MPI_DOUBLE, my_rank-1, 0,
MPI_COMM_WORLD ); //sending data of upper first boundry
          MPI_Recv(*(*(z+q) + (my_rank*stripes[my_rank])+remainder-1), size, MPI_DOUBLE, my_rank-1, 0,
MPI_COMM_WORLD, &status ); //recieving data for its upper boundry-1
       }

    }else{ //all other processors exchange their left and right boundry data to (rank-1) and (rank+1)
       if (mpi_size>1){ //checking if more than 1 processors exist
         if (my_rank<remainder){
          MPI_Send(*(*(z+q) + stripes[my_rank]*my_rank), size, MPI_DOUBLE, my_rank-1, 0,
MPI_COMM_WORLD ); //sending its first upper boundry data
          MPI_Recv(*(*(z+q) + (stripes[my_rank]*my_rank)-1), size, MPI_DOUBLE, my_rank-1, 0,
MPI_COMM_WORLD, &status); //Recieve data for its upper boundery-1
```

```
        MPI_Send(*(*(z+q) + (stripes[my_rank]*(my_rank+1))-1) , size, MPI_DOUBLE, my_rank+1, 0,
MPI_COMM_WORLD ); //sending its last lower boundry data
        MPI_Recv(*(*(z+q) + stripes[my_rank]*(my_rank+1)) , size, MPI_DOUBLE, my_rank+1, 0,
MPI_COMM_WORLD, &status); //Recieve data for its lower boundery+1
       }else{
        MPI_Send(*(*(z+q) + (stripes[my_rank]*my_rank)+remainder), size, MPI_DOUBLE, my_rank-1, 0,
MPI_COMM_WORLD ); //sending its first upper boundry data
        MPI_Recv(*(*(z+q) + (stripes[my_rank]*my_rank)+remainder-1), size, MPI_DOUBLE, my_rank-1, 0,
MPI_COMM_WORLD, &status); //Recieve data for its upper boundery-1
        MPI_Send(*(*(z+q) + (stripes[my_rank]*(my_rank+1))+remainder-1) , size, MPI_DOUBLE, my_rank+1, 0,
MPI_COMM_WORLD ); //sending its last lower boundry data
        MPI_Recv(*(*(z+q) + (stripes[my_rank]*(my_rank+1))+remainder) , size, MPI_DOUBLE, my_rank+1, 0,
MPI_COMM_WORLD, &status); //Recieve data for its lower boundery+1
       }
      }
   }

   //computing z data for current t
   if (my_rank<remainder){
     #pragma omp parallel for
     for (int i = (my_rank*stripes[my_rank]); i < (my_rank+1)*stripes[my_rank]; i++ ) {
     if (i==0 || i==size-1){
       continue;
      }
      for ( int j = 1; j < size - 1; j++ ) {
       z[p][i][j] = 2.0 * z[q][i][j] - z[r][i][j] + c * c * dt * dt / ( dd * dd ) * ( z[q][i + 1][j] + z[q][i - 1][j] + z[q][i][j +
1] + z[q][i][j - 1] - 4.0 * z[q][i][j] );
      }
     }
   }else{
     #pragma omp parallel for
     for (int i = ((my_rank*stripes[my_rank])+remainder); i < (((my_rank+1)*stripes[my_rank])+remainder); i++ ) {
     if (i==0 || i==size-1){
       continue;
      }
      for ( int j = 1; j < size - 1; j++ ) {
       z[p][i][j] = 2.0 * z[q][i][j] - z[r][i][j] + c * c * dt * dt / ( dd * dd ) * ( z[q][i + 1][j] + z[q][i - 1][j] + z[q][i][j +
1] + z[q][i][j - 1] - 4.0 * z[q][i][j] );
      }
     }
   }

   //printing the data for the interval
   if(interval != 0 && t%interval == 0){
   //all workers send their z's data to master.
     if (my_rank!=0){
       if (my_rank<remainder){
```

```
        MPI_Send(*(*(z+p)+(stripes[my_rank]*my_rank)), stripes[my_rank]*size, MPI_DOUBLE, 0, 0,
MPI_COMM_WORLD );
       }else{
        MPI_Send(*(*(z+p)+(stripes[my_rank]*my_rank)+remainder), stripes[my_rank]*size, MPI_DOUBLE, 0, 0,
MPI_COMM_WORLD );
       }
     }else{
       //master recieves data from workers
       for (int rank=1; rank<mpi_size; rank++){
         if (rank<remainder){
           MPI_Recv(*(*(z+p) + rank*stripes[rank]), stripes[rank]*size, MPI_DOUBLE, rank, 0,
MPI_COMM_WORLD, &status );
         }else{
           MPI_Recv(*(*(z+p) + rank*stripes[rank]+remainder), stripes[rank]*size, MPI_DOUBLE, rank, 0,
MPI_COMM_WORLD, &status );
         }
       }
       //printing z data for current time
       cout << t << endl;
       for(int i = 0; i < size; i++){
          for(int j = 0; j < size; j++){
             cout << z[p][j][i] << " ";
          }
          cout << endl;
       }
       cout << endl;
     }
    }
   }
   // end of simulation

  MPI_Finalize( ); // shut down MPI

  // finish the timer
  if(my_rank == 0){
    for (int rank=0; rank<mpi_size; rank++){
      if (rank<remainder){
        cerr << "rank[" << rank << "]'s range " << rank*stripes[rank] << "~" << (rank+1)*stripes[rank]-1 << endl;
      }else{
        cerr << "rank[" << rank << "]'s range " << rank*stripes[rank]+remainder << "~" <<
(rank+1)*stripes[rank]+remainder-1 << endl;
      }
    }
    cerr << "Elapsed time = " << time.lap( ) << endl;
  }
  return 0;

}
```

## Lab 2: Matrix multiplication

Here the multiplication process is parallelized by dividing matrix A into horizontal strips and sharing them between all processors included for MPI. Entire matrix B is shared with all processors. Each processor computes and shares the horizontal strips of matrix C with master. Master receives the chunks of matrix C from all worker processors and displays the final output matrix.

## Output screenshots:

```
[poojan26@cssmpi1h pooja_lab2]$ mpirun -np 2 ./matrix_mpi_new 2 y
elapsed time = 1593
c[0][0] = 1
c[0][1] = 0
c[1][0] = 2
c[1][1] = -1
[poojan26@cssmpi1h pooja_lab2]$ mpirun -np 3 ./matrix_mpi_new 3 y
elapsed time = 4100
c[0][0] = 5
c[0][1] = 2
c[0][2] = -1
c[1][0] = 8
c[1][1] = 2
c[1][2] = -4
c[2][0] = 11
c[2][1] = 2
c[2][2] = -7
[poojan26@cssmpi1h pooja_lab2]$ mpirun -np 4 ./matrix_mpi_new 4 y
elapsed time = 7691
c[0][0] = 14
c[0][1] = 8
c[0][2] = 2
c[0][3] = -4
c[1][0] = 20
c[1][1] = 10
c[1][2] = 0
c[1][3] = -10
c[2][0] = 26
c[2][1] = 12
c[2][2] = -2
c[2][3] = -16
c[3][0] = 32
c[3][1] = 14
c[3][2] = -4
c[3][3] = -22
```

```
[poojan26@cssmpi1h pooja_lab2]$ mpirun -np 4 ./matrix_mpi_new 8 y
elapsed time = 5251
c[0][0] = 140
c[0][1] = 112
c[0][2] = 84
c[0][3] = 56
c[0][4] = 28
c[0][5] = 0
c[0][6] = -28
c[0][7] = -56
c[1][0] = 168
c[1][1] = 132
c[1][2] = 96
c[1][3] = 60
c[1][4] = 24
c[1][5] = -12
c[1][6] = -48
c[1][7] = -84
c[2][0] = 196
c[2][1] = 152
c[2][2] = 108
c[2][3] = 64
c[2][4] = 20
c[2][5] = -24
c[2][6] = -68
c[2][7] = -112
c[3][0] = 224
c[3][1] = 172
c[3][2] = 120
c[3][3] = 68
c[3][4] = 16
c[3][5] = -36
c[3][6] = -88
c[3][7] = -140
c[4][0] = 252
c[4][1] = 192
c[4][2] = 132
c[4][3] = 72
c[4][4] = 12
c[4][5] = -48
c[4][6] = -108
c[4][7] = -168
c[5][0] = 280
c[5][1] = 212
c[5][2] = 144
c[5][3] = 76
c[5][4] = 8
c[5][5] = -60
c[5][6] = -128
c[5][7] = -196
c[6][0] = 308
c[6][1] = 232
c[6][2] = 156
c[6][3] = 80
c[6][4] = 4
c[6][5] = -72

c[6][6] = -148
c[6][7] = -224
c[7][0] = 336
c[7][1] = 252
c[7][2] = 168
c[7][3] = 84
c[7][4] = 0
c[7][5] = -84
c[7][6] = -168
c[7][7] = -252
[poojan26@cssmpi1h pooja_lab2]$
```

## Source code: Matrix Multiplication

```
#include "mpi.h"
#include <stdlib.h> // atoi
#include <iostream> // cerr
#include "Timer.h"
using namespace std;

void init( double *matrix, int size, char op ) {
 for ( int i = 0; i < size; i++ )
  for ( int j = 0; j < size; j++ )
   matrix[i * size + j]
 = ( op == '+' ) ? i + j :
 ( ( op == '-' ) ? i - j : 0 );
}

void print( double *matrix, int size, char id ) {
 for ( int i = 0; i < size; i++ )
  for ( int j = 0; j < size; j++ )
   cout << id << "[" << i << "][" << j << "] = " << matrix[i * size + j] << endl;
}

void multiplication( double *a, double *b, double *c, int stripe, int size ) {
 for ( int k = 0; k < size; k++ )
  for ( int i = 0; i < stripe; i++ )
   for ( int j = 0; j < size; j++ )
 c[i * size + k] += a[i * size + j] * b[j * size + k];   // c[i][k] += a[i][j] * b[j][k];
}

int main( int argc, char* argv[] ) {
 int my_rank;          // used by MPI
 int mpi_size;         // used by MPI
 int size = 400;           // array size
 bool print_option = false;  // print out c[] if it is true
 Timer timer;

 // variables verification
 if ( argc == 3 ) {
  if ( argv[2][0] == 'y' )
   print_option = true;
 }

 if ( argc == 2 || argc == 3 ) {
  size = atoi( argv[1] );
 }
 else {
  cerr << "usage:   matrix size [y|n]" << endl;
  cerr << "example: matrix 400   y" << endl;
  return -1;
 }

 MPI_Init( &argc, &argv ); // start MPI
 MPI_Comm_rank( MPI_COMM_WORLD, &my_rank );
 MPI_Comm_size( MPI_COMM_WORLD, &mpi_size );
 MPI_Status status;

 // matrix initialization
 double *a = new double[size * size];
```

```
double *b = new double[size * size];
double *c = new double[size * size];

if ( my_rank == 0 ) { // master initializes all matrices
  init( a, size, '+' );
  init( b, size, '-' );
  init( c, size, '0' );

  if ( false ) {       // print initial values
    print( a, size, 'a' );
    print( b, size, 'b' );
  }
  timer.start( ); // start a timer
}
else {              // slavs zero-initializes all matrices
  init( a, size, '0' );
  init( b, size, '0' );
  init( c, size, '0' );
}

MPI_Bcast( &size, 1, MPI_INT, 0, MPI_COMM_WORLD ); //broadcast the matrix size to all.

int stripe = size / mpi_size;    // partitioned stripe

if ( my_rank == 0 ) {
  for ( int rank = 1; rank < mpi_size; rank++ ) {
    // master sends each partition of a[] to a different slave
    MPI_Send( a + stripe * size * rank, stripe * size, MPI_DOUBLE, rank, 0, MPI_COMM_WORLD );
    // master also sends b[] to all slaves
    MPI_Send( b, size * size, MPI_DOUBLE, rank, 0, MPI_COMM_WORLD );
  }
}
else {
  MPI_Recv( a, stripe * size, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, &status );
  MPI_Recv( b, size * size, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, &status );
}

//all ranks should compute multiplication
multiplication( a, b, c, stripe, size );

//master receives each partition of c[] from a different slave
if (my_rank != 0 ) {
    MPI_Send(c, stripe * size, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD );
}
else {
  for ( int rank = 1; rank < mpi_size; rank++ ) {
    MPI_Recv( c + stripe * size * rank, stripe * size, MPI_DOUBLE, rank, 0, MPI_COMM_WORLD, &status );
  }
}

if ( my_rank == 0 )
  cout << "elapsed time = " << timer.lap( ) << endl;    // stop the timer

// results
if ( print_option && my_rank == 0 )
  print( c, size, 'c' );
MPI_Finalize( ); // shut down MPI
}
```