```
/*
Beverly G Wertz     - Project 1
CS4379 - Parallel Programming Spring 2012

For my implementation I check if there is more than 1 process then get the sqroot of the
# of processes. To determine the row and column communicators I divide the process rank
by the sqroot which produces a value from 0 to 3 and then get the modulo result of the
rank from the sqroot. The communicators are then created for the row via the color - which
is the result from dividing the rank by the squareroot and the key is set as the modulo
result.  For the column communicators the color and key are reversed.

The bcasts roots are determined by dividing k by the sub matrix row and column size.
Bcasts are done for the row and column at the start of every k iteration, as long as there
are multiple processes.

Results 1024x1024:
Results with 1 process -
    cmd=mpirun -np 1 -machinefile /home/bmifflet/cs4379/project1/machinefile.1519984
    /home/bmifflet/cs4379/project1/proj1
    Elapsed time is 2.007120
Results with 4 processes -
    cmd=mpirun -np 4 -machinefile /home/bmifflet/cs4379/project1/machinefile.1519985
    /home/bmifflet/cs4379/project1/proj1
    Elapsed time is 0.733457
Results with 16 processes -
    cmd=mpirun -np 16 -machinefile /home/bmifflet/cs4379/project1/machinefile.1519986
    /home/bmifflet/cs4379/project1/proj1
    Elapsed time is 0.243832

Results 4096x4096
Results with 1 process -
    cmd=mpirun -np 1 -machinefile /home/bmifflet/cs4379/project1/machinefile.1520017
    /home/bmifflet/cs4379/project1/proj1
    Elapsed time is 145.900903
    cmd=mpirun -np 1 -machinefile /home/bmifflet/cs4379/project1/machinefile.1520020
    /home/bmifflet/cs4379/project1/proj1
    Elapsed time is 160.943575
Results with 4 processes -
    cmd=mpirun -np 4 -machinefile /home/bmifflet/cs4379/project1/machinefile.1520016
    /home/bmifflet/cs4379/project1/proj1
    Elapsed time is 59.902480
    cmd=mpirun -np 4 -machinefile /home/bmifflet/cs4379/project1/machinefile.1520021
    /home/bmifflet/cs4379/project1/proj1
    Elapsed time is 52.499395
Results with 16 processes -
    cmd=mpirun -np 16 -machinefile /home/bmifflet/cs4379/project1/machinefile.1520015
    /home/bmifflet/cs4379/project1/proj1
    Elapsed time is 31.487606
    cmd=mpirun -np 16 -machinefile /home/bmifflet/cs4379/project1/machinefile.1520022
    /home/bmifflet/cs4379/project1/proj1
    Elapsed time is 31.530095

*/
```

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <mpi.h>

int MAX = 1024;

int main(int argc, char **argv)
{
    //Initialize variables to track i, j, k, rank, size, start row and column and the
    //numerical identifier for the row and column communicator
    int rank,size,i,j,k,startrow,startcol,colComm,rowComm,root;

    //Initializing our MPI and getting the rank and size
    MPI_Init (&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    //Setup sqrt(P) as an integer to facilitate ease of later calculations
    int rootP = (int) sqrt(size);

    //Setup group/communicator variables
    MPI_Comm newCommRow, newCommCol;
    rowComm = rank/rootP;
    colComm = rank%rootP;

    //Split up the communicators in to row and column specific groups
    MPI_Comm_split(MPI_COMM_WORLD,rowComm,colComm,&newCommRow);
    MPI_Comm_split(MPI_COMM_WORLD,colComm,rowComm,&newCommCol);

    //Use the rootP to calculate the upper left corner of each submatrix
    startrow = (MAX/rootP)*(rank/rootP);
    startcol = (MAX/rootP)*(rank%rootP);

    //Initialize the random seed
    unsigned int randSeed = (unsigned int) time(NULL);
    srand (randSeed);

    //Setup MPI variables to track send and recieves.
    MPI_Status status;
    MPI_Request rowRequest, colRequest;
    double startTime = 0, endTime = 0;

    //Initialize timer on process zero
    if (rank == 0)
        startTime = MPI_Wtime();

    //Allocating space for A1, A2 and the row and column arrays.
    int **A1, **A2, *row, *col;
    A1 = (int**) malloc((MAX/rootP)*sizeof(int*));
    A2 = (int**) malloc((MAX/rootP)*sizeof(int*));
```

```
    //1d array allocation to hold the current row and col received from the comm group
    row = malloc((MAX/rootP)*sizeof(int));
    col = malloc((MAX/rootP)*sizeof(int));

    //Allocate space for the 2d arrays
    for (i = 0; i < MAX/rootP; i++) {
        A1[i] = (int*) malloc((MAX/rootP)*sizeof(int));
        A2[i] = (int*) malloc((MAX/rootP)*sizeof(int));
    }

    //Generate values for starting submatrix
    for (i = 0; i < MAX/rootP; i++)
        for (j = 0; j < MAX/rootP; j++)
            A1[i][j] = ((startrow+MAX/rootP)*(i+1)*(j+3))%15;

    //k loop which controls 1st level of iteration
    for (k = 0; k < MAX; k++) {

        //Bcast row/column to the other processes
        if (size != 1) {
            //Set the root for k communication
            root = k/(MAX/rootP);

            //First I'm broadcasting rows, then I will broadcast columns
            //First check to see if k row falls inside our own submatrix
            if (k >= startrow && k < (startrow + MAX/rootP)) {
                //If in our submatrix, copy the row to our sending array
                //first we setup an effective row variable that maps to our submatrix
                int effRow = k%(MAX/rootP);
                for (i = 0; i < MAX/rootP; i++)
                    row[i] = A1[effRow][i];
            }

            //Send/Recv the row to our Col Comm since we need to transmit the row to all
            //the columns in the communicator
            MPI_Bcast(row,MAX/rootP,MPI_INT,root,newCommCol);

            //Now I'm broadcasting columns
            //First check to see if k row falls inside our own submatrix
            if (k >= startrow && k < (startrow + MAX/rootP)) {
                //If in our submatrix, copy the row to our sending array
                //first we setup an effective row variable that maps to our submatrix
                int effCol = k%(MAX/rootP);
                for (i = 0; i < MAX/rootP; i++)
                    col[i] = A1[i][effCol];
            }

            //Send/Recv the col to our row Comm since we need to transmit the col to all
            //the processes in the same row in the communicator
            MPI_Bcast(col,MAX/rootP,MPI_INT,root,newCommRow);

        } else {
            //If we only have one process, no need to bcast, just setup the row and column
```

```
            //comparison arrays
            for (i = 0; i < MAX/rootP; i++) {
                row[i] = A1[k][i];
                col[i] = A1[i][k];
            }
        }

        //Run Comparison to see which is the smallest
        for (i = 0; i < MAX/rootP; i++)
            for (j = 0; j < MAX/rootP; j++) {
                A2[i][j] = A1[i][j];

                //Compare results with our temporary array
                int tmpComp = row[j]+col[i];
                if (tmpComp < A1[i][j])
                    A2[i][j] = tmpComp;
        }

        //Copy A2 back to A1;
        for (i = 0; i < MAX/rootP; i++) {
            for (j = 0; j < MAX/rootP; j++){
                A1[i][j] = A2[i][j];
            }
        }
    }

    //Calculate the ending time for the entire program
    if (rank == 0) {
        endTime = MPI_Wtime();
        printf("Elapsed time is %f\n",endTime-startTime);
    }

    //Close out all processes
    MPI_Finalize();
    return 0;
}
```