# CMPE 300 MPI Project

# Game of Life

Barış Başmak

2016400087

Submission Date : 20.12.2019

## Introduction :

The project's goal is to implement a simulation called game of life using OpenMPI.

The game consists of a 360x360 map and if there's a creature at an index that index's value is 1 else it is 0. If a creature has less than 2 or more than 3 neighbor creatures it dies. If a cell has value 0 and has 3 creature neighbor's it becomes 1. The map is formed as a toroid, so the first worker process' partition is connected to the last worker connection as if a paper was rolled to a cylinder. Moreover the rightmost and leftmost cells are also connected the same way.

These iterations will be done t times and the map is partitioned into (processor count – 1) parts.

These processors communicate with each other in order to count the boundaries' neighbors.

Master processor : Processor with rank 0. The one that distributes the data and handles I/O.

Worker processor : The ones that do the iterations and communicate with each other.

id == rank of a processor.

## Program Interface :

The user can run the program by compiling the main.cpp file by mpic++ main.cpp -o a

And then run it with n processors and t iterations by

```
mpirun -np [n] –oversubscribe ./a “input file” “output file” [t]
```

The program doesn't have any other user inputs and gives an output by writing to the outputfile.

## Program Execution :

The program can be compiled by compiling with by mpic++ main.cpp  -o a

And then run it with n processors and t iterations by

```
mpirun -np [n] –oversubscribe ./a “input file”  “output file” [t]
```

The program doesn’t have any other user inputs and gives an output by writing to the outputfile.

## Input and Output :

The program reads the input from a txt file which has 360 x 360 integers (either 1 or 0 ) with spaces in between. This file is read and translated into a matrix.

The output file is also a txt file with the same specifications.

## Program Structure :

At first the MPI variables are initialized and the ranks are named id. Id == 0  is the master processor which handles I/O and distributes data. Master processor reads the input from the input file and then distributes the data to the worker processors according to how many rows there are per worker processor. And then it waits for all of the processors to send back the changed data ( after t iterations ). After receiving it outputs the matrix’s contents to the output file.

The worker processors are the ones that have id’s (ranks) that are different from 0. They first compute how many rows they should have and the data points they’ll have. After that they wait to receive their part of data from the master processor. The data received is put into a matrix and then copied into another one with 2 spare rows that’ll later act as buffers.

Then there’s a while loop that exits if the t’th iteration is done.

Then the top and bottom rows of the processors that have odd id’s are sent to and received from the adjacent processors (first bottom communication done with id+1 and then top to id-1 with id = 1 and id = worker count as exceptions they send theirs to each other. (1 sends top to last processor and last processor sends bottom to processor 1 ) ) At the same time even numbered processors first wait for data and then send data to the top processor and then do the same for the process stationed as one below them.

Afterwards the data is checked with respect to game of life’s rules.

When the while loop exits after t iterations of communications and computations, the data is sent back to the master processor and the processor is done.

## Difficulties encountered :

I didn't encounter any major difficulties apart from a few minor changes I made while debugging.

## Improvements and Extensions :

The game could have been implemented such that it had more functions. This way since it's not a big program debugging etc. is relatively easy but it is not perfect.

Also I haven't implemented the checkered version of the project which would've been an improvement.

## Conclusion :

In conclusion I have successfully finished the project with the 20 point toroid bonus.

## Appendices :

## main.cpp

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <mpi.h>
#include <stdlib.h>

using namespace std ;

int main(int argc, char * argv[] )
{

    int id;
    int ierr;
    int processor_count ;   //how many processors there are
    ierr = MPI_Init ( &argc, &argv );

    if ( ierr != 0 )
    {
        cout << "\n";
        cout << "MULTITASK_MPI - Fatal error!\n";
        cout << "  MPI_Init returned ierr = " << ierr << "\n";
        exit ( 1 );
    }

    ierr = MPI_Comm_rank ( MPI_COMM_WORLD, &id );

    ierr = MPI_Comm_size ( MPI_COMM_WORLD, &processor_count );


//Master processor

    if(id == 0)
    {
        string arguments[argc] ;
        for(int i = 0 ; i < argc ; i++){
            arguments[i] = argv[i];
        }


        string input_file = arguments[1];
        string output_file = arguments[2];
        string iteration = arguments[3];

        stringstream streamer2(iteration);
        int iteration_count;

        streamer2 >> iteration_count;

        int tokens[360][360] ;

        ifstream myfile (input_file);

        if (myfile.is_open()){


          for(int i = 0 ; i < 360 ; i++){
            for(int j = 0 ; j < 360 ; j++){
                    int q;
                    myfile >> q;
                    tokens[i][j] = q ;

                }

              }
                myfile.close();
        }
        ///////////sending data to worker processors !
        int row_per_worker = 360 / (processor_count-1) ;
        int width = 360 ;
        int data_points_per_worker = width * row_per_worker ;
        for(int i = 1 ; i < processor_count ; i++)
        {
            MPI_Send(&tokens [ (i - 1) * row_per_worker ][ 0 ] , data_points_per_
worker, MPI_INT , i , 0, MPI_COMM_WORLD ) ;
        }
        //Data distributed to workers


        //Receiving the data back and wirting to file
        int received_back = 0 ;
        MPI_Status status ;
        while(received_back < ( processor_count - 1)  )
        {
            int receive[row_per_worker][width] ;

            MPI_Recv(&receive[0][0] , data_points_per_worker, MPI_INT, MPI_ANY_SO
URCE, MPI_ANY_TAG, MPI_COMM_WORLD, &status);

            int source = status.MPI_SOURCE ;
            int starting_index = (source - 1) * row_per_worker ;

            for(int i = 0 ; i < row_per_worker ; i++)
            {
                for(int j = 0 ; j < 360 ; j++)
                {
                    tokens[ starting_index + i ][j] = receive[i][j] ;
                }
            }

            received_back ++ ;
        }


        ofstream new_file( output_file ) ;
          for(int i = 0 ; i < 360 ; i++)
          {
            for(int j = 0 ; j < 360 ; j++ )
            {
                if( j != 360)
                    new_file<<tokens[i][j]<<" ";
                else
```

```
                    new_file<<tokens[i][j];

            }
            new_file<<endl;
          }
        MPI_Finalize ( ) ;
    }   // WORKER PROCESSORS
    else if(id != 0)
    {
        string arguments[argc] ;
        for(int i = 0 ; i < argc ; i++)
        {
            arguments[i] = argv[i];
        }

        int iteration_wanted = stoi(arguments[3]);
        int width = 360 ;
        int worker_count = processor_count - 1 ;
        int my_row_count = 360 / worker_count ;
        int my_data_points = my_row_count * width ;

        int work_data [ my_row_count + 2 ][ width ] ;
        int send_data [ my_row_count ][ width ] ;

        int ierr;
        MPI_Status status ;
        //Receiving data from P0
        ierr = MPI_Recv(&send_data[0][0], my_data_points, MPI_INT, 0, MPI_ANY_TAG
, MPI_COMM_WORLD, &status) ;

        if(ierr != 0)
        {
            cout<<"Error when receiving data from P0   This is processor P "<<id
<<endl ;
        }

        for(int i = 0  ;  i < my_row_count ; i++)
        {
            for(int j = 0 ; j < width ; j++)
            {
                work_data[i+1][j] = send_data[i][j] ;
            }
        }
        //Receiving data from P0
        //COMMUNNICATION BETWEEN PROCESSES

//send_datadan atmalik
        int bottom_send_index = my_row_count-1 ;
        int top_send_index = 0 ;

//worker_datasi icin
        int top_recv = 0 ;
        int bottom_recv = my_row_count+1;
       int current_iteration ;
       while(iteration_wanted != current_iteration)
       {


            if( id % 2 == 1)
            {
                if(id != worker_count)
                {//Asagiya atis ve asagidan recv
                    MPI_Send(&send_data[bottom_send_index][0], width, MPI_INT, (i
d + 1), 0, MPI_COMM_WORLD );
                    MPI_Recv(&work_data[bottom_recv][0], width , MPI_INT, id + 1
 ,MPI_ANY_TAG, MPI_COMM_WORLD, &status) ;
                }
                //Project assumption --> EVEN NUMBER OF WORKER PROCESSORS
                if(id != 1)
                {
                    MPI_Send(&send_data[top_send_index][0] , width, MPI_INT , id
-1 , 0 , MPI_COMM_WORLD);
                    MPI_Recv(&work_data[top_recv][0], width , MPI_INT, id - 1  ,M
PI_ANY_TAG, MPI_COMM_WORLD, &status) ;
                }
                else
                {
                    MPI_Send(&send_data[top_send_index][0] , width, MPI_INT , wor
ker_count , 0 , MPI_COMM_WORLD);
                    MPI_Recv(&work_data[top_recv][0], width , MPI_INT, worker_cou
nt  ,MPI_ANY_TAG, MPI_COMM_WORLD, &status) ;
                }
            }

            else if(id % 2 == 0)
            {
                MPI_Recv(&work_data[top_recv][0],  width , MPI_INT, id - 1  ,MPI_
ANY_TAG, MPI_COMM_WORLD, &status) ;
```

```
        MPI_Send(&send_data[top_send_index][0] , width, MPI_INT , id -
1 , 0 , MPI_COMM_WORLD);

        if(id != worker_count)
        {
          MPI_Recv(&work_data[bottom_recv][0],  width , MPI_INT, id + 1
 ,MPI_ANY_TAG, MPI_COMM_WORLD, &status) ;
          MPI_Send(&send_data[bottom_send_index][0] , width, MPI_INT ,
id + 1 , 0 , MPI_COMM_WORLD);
        }else
        {
          MPI_Recv(&work_data[bottom_recv][0],  width , MPI_INT, 1  ,MP
I_ANY_TAG, MPI_COMM_WORLD, &status) ;
          MPI_Send(&send_data[bottom_send_index][0] , width, MPI_INT ,
1 , 0 , MPI_COMM_WORLD);
        }

      }

      int helper[8][2] = {-1, -1, 0, -1, 1, -1, -1, 0, 1, 0 , -
1 , 1, 0, 1, 1, 1} ;
      for(int i = 1 ; i < my_row_count + 1 ; i++)
      {
        for(int j = 0 ; j < width ; j++)
        {
          int current = work_data[i][j] ;
          int count = 0 ;
          bool rightmost = ( j == 359 ) ;
          bool leftmost  = ( j == 0 )   ;

          if(rightmost)
          {

            for(int x = 0 ; x < 8 ; x++)
            {
              if( x < 5 && work_data[i + helper[x][0] ][ j + helper
[x][1] ] == 1)
                count++ ;

            }
            if (work_data[i - 1][0] == 1)
                count ++;
            if( work_data[i][0] == 1)
              count++ ;
            if(work_data[i + 1][0] == 1)
```

```
                            count++ ;
                    }
                    if(leftmost)
                    {

                        for(int x = 0 ; x < 8 ; x++)
                        {
                            if( x > 2 && work_data[i + helper[x][0] ][ j + helper
[x][1] ] == 1)

                                count++ ;

                        }
                        if (work_data[i - 1][359] == 1)
                                count ++;
                        if( work_data[i][359] == 1)
                            count++ ;
                        if(work_data[i + 1][359] == 1)
                            count++ ;
                    }
                    if(!leftmost && !rightmost)
                    {

                        for(int x = 0 ; x < 8 ; x++)
                        {
                            if( work_data[i + helper[x][0] ][ j + helper[x][1] ]
== 1)

                                count++ ;
                        }
                    }

                    if(count < 2 )
                    {
                        send_data[i-1][j] = 0 ;
                    }
                    if(count > 3)
                    {
                        send_data[i-1][j] = 0 ;
                    }
                    if(count == 3 && send_data[i-1][j] == 0)
                    {
                        send_data[i-1][j] = 1 ;
                    }

                }
```

```
            }
            for(int k = 0 ; k < my_row_count ; k++ )
            {
                for(int m = 0 ; m < width ; m++)
                {
                    work_data[k+1][m] = send_data[k][m];
                }
            }

            current_iteration++ ;
        }

        MPI_Send(&send_data[0][0], my_data_points, MPI_INT, 0, 0, MPI_COMM_WORLD)
 ;


        MPI_Finalize ( ) ;
    }

    return 0 ;
}
```