Assignment 6

Stuart Hopkins, A02080107

February 2020

1 Game of Life

To create the Game of Life I first start by initializing the array. In the code provided it is based off of a 1024 x 1024 table as stated in the instructions. This is done on process 0. I then create splices of the initial array and send them off to the slave processes. While the slaves are executing, process 0 also calculates its share of data.

Once the children are finished, process 0 gathers all the information and plugs it back into the game board. Finally, it prints the data to the console. It includes the processing time in each. After the set number of iterations the program terminates. It should be noted that there is currently a sleep call so that the output is readable. My code is as follows:

```
#include <iostream>
#include <mpi.h>
#include <unistd.h>
#include <stdlib.h>
#include <time.h>
#include <chrono>

#define MCW MPI_COMM_WORLD

using namespace std;
using namespace std::chrono;

void printBoard(int arr[], const int ROW_SIZE, const int ROWS, string printMsg) {
  cout << " " << printMsg << endl;
  for(int i = 0; i < ROWS; ++i) {
    for(int j = 0; j < ROW_SIZE; ++j) {
      if(arr[ROW_SIZE * i + j])
        cout << " 0";
      else
        cout << " -";
    }
    cout << endl;
  }
  cout << endl;
}
```

```
void printArr(int arr[], const int ROW_SIZE, const int ROWS, string printMsg) {
  cout << " " << printMsg << endl;
  for(int i = 0; i < ROWS; ++i) {
    for(int j = 0; j < ROW_SIZE; ++j) {
      cout << " " << arr[ROW_SIZE * i + j];
    }
    cout << endl;
  }
  cout << endl;
}


int calcNeighbors(int arr[], const int ROW_SIZE, int rows, int i, int j) {
  int neighbors = 0;

  //north 3
  if(i != 0) {
    // Upper left
    if(j != 0)
      neighbors += arr[ROW_SIZE * (i-1) + (j-1)];
    // Upper
    neighbors += arr[ROW_SIZE * (i-1) + j];
    // Upper right
    if(j+1 < ROW_SIZE)
      neighbors += arr[ROW_SIZE * (i-1) + (j+1)];
  }

  // Middle 2
  // Left
  if(j != 0)
    neighbors += arr[ROW_SIZE * (i) + (j-1)];
  // Right
  if(j+1 < ROW_SIZE)
    neighbors += arr[ROW_SIZE * (i) + (j+1)];

  // South 3
  if(i+1 < rows) {
    // Upper left
    if(j != 0)
      neighbors += arr[ROW_SIZE * (i+1) + (j-1)];
    // Upper
    neighbors += arr[ROW_SIZE * (i+1) + j];
    // Upper right
    if(j+1 < ROW_SIZE)
      neighbors += arr[ROW_SIZE * (i+1) + (j+1)];
  }
  return neighbors;
}


void makeNeighborMap(int SPLICE_SIZE, int ROW_SIZE, int graph[]) {
  int tmpArr[SPLICE_SIZE];
  for(int i = 0; i < SPLICE_SIZE; i++) {
    tmpArr[i] = graph[i];
  }

  for(int i = 0; i < SPLICE_SIZE / ROW_SIZE; ++i) {
    for(int j = 0; j < ROW_SIZE; ++j) {
      tmpArr[ROW_SIZE * i + j]= calcNeighbors(graph, ROW_SIZE, SPLICE_SIZE / ROW_SIZE, i, j);
    }
  }
  printArr(tmpArr, ROW_SIZE, SPLICE_SIZE / ROW_SIZE, "2");
}


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

  int rank, size;
  int data;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MCW, &rank);
  MPI_Comm_size(MCW, &size);
  const int ITERS = 40;
  const int ROW_SIZE = 1024;
  const int SPLICE_SIZE = ROW_SIZE * (2 + ROW_SIZE / size);


  if(!rank) {
    srand(time(0));
    int graph[(1 + ROW_SIZE) * ROW_SIZE];
    for(int i = 0; i < ROW_SIZE; ++i) {
      for(int j = 0; j < ROW_SIZE; ++j) {
        graph[ROW_SIZE * i + j] = 0;
        if(!(rand() % 5))
          graph[ROW_SIZE * i + j] = 1;
      }
    }

  // GLIDER EXAMPLE
  // int graph[(1+ROW_SIZE) * ROW_SIZE] = {0,0,0,0,0,0,0,0,
  //                                        0,0,0,0,0,0,0,0,
  //                                        0,1,0,0,0,0,0,0,
  //                                        0,0,1,1,0,0,0,0,
  //                                        0,1,1,0,0,0,0,0,
  //                                         0,0,0,0,0,0,0,0,0,
  //                                         0,0,0,0,0,0,0,0,0,
  //                                         0,0,0,0,0,0,0,0,0,
  //                                         0,0,0,0,0,0,0,0,0};

      int tmpArrZ[SPLICE_SIZE];
      printBoard(&graph[ROW_SIZE], ROW_SIZE, ROW_SIZE, "Full Board at Iteration 0:");

    for(int iters = 0; iters < ITERS; ++iters) {
      sleep(1);

      auto start = high_resolution_clock::now();
      for(int i = 0; i < ROW_SIZE; ++i) {
        graph[i] = 0;
      }

      for(int j = 0; j < SPLICE_SIZE; ++j) {
        tmpArrZ[j] = graph[j];
      }

      int tmpArr[SPLICE_SIZE];
      for(int i = 1; i < size; ++i) {
        for(int j = 0; j < SPLICE_SIZE; ++j) {
          tmpArr[j] = graph[(i*(ROW_SIZE/size)-1)*ROW_SIZE + j];
        }
        MPI_Send(tmpArr, SPLICE_SIZE, MPI_INT, i, 0, MCW);
      }

      for(int i = 0; i < (SPLICE_SIZE / ROW_SIZE); ++i) {
        for(int j = 0; j < ROW_SIZE; ++j) {
          int neighbors = calcNeighbors(graph, ROW_SIZE, SPLICE_SIZE / ROW_SIZE, i, j);

          if(neighbors == 0 || neighbors == 1) {
            tmpArrZ[ROW_SIZE * i + j] = 0;
          }
          else if(neighbors == 3){
            tmpArrZ[ROW_SIZE * i + j] = 1;
          }
          else if(neighbors > 3){
            tmpArrZ[ROW_SIZE * i + j] = 0;
          }
        }
      }

      for(int i = 0; i < SPLICE_SIZE - ROW_SIZE; ++i) {
        graph[i] = tmpArrZ[i];
      }
      for(int i = 1; i < size; ++i) {
        MPI_Recv(tmpArr, SPLICE_SIZE, MPI_INT, i, 0, MCW,MPI_STATUS_IGNORE);

        for(int j = ROW_SIZE; j < SPLICE_SIZE; ++j) {
          graph[(i*(ROW_SIZE/size)-1)*ROW_SIZE + j] = tmpArr[j];
        }
      }

      auto stop = high_resolution_clock::now();
      auto duration = duration_cast<microseconds>(stop - start);
      cout << " TIME: " << duration.count() << " microseconds" << endl;

      string printMsg = "Full Board at iteration " + to_string(iters+1) + ":";
      printBoard(&graph[ROW_SIZE], ROW_SIZE, ROW_SIZE, printMsg);
    }
  }

  // <<<<<<<<<<<<<<<< SLAVES >>>>>>>>>>>>>>>>>>>>>>
  if(rank) {
    for(int iters = 0; iters < ITERS; ++iters){
      int arr[SPLICE_SIZE];
      int tmpArr[SPLICE_SIZE];
      MPI_Recv(arr, SPLICE_SIZE, MPI_INT, MPI_ANY_SOURCE, 0, MCW,MPI_STATUS_IGNORE);

      for(int i = 0; i < SPLICE_SIZE; ++i) {
        tmpArr[i] = arr[i];
      }

      for(int i = 0; i < SPLICE_SIZE / ROW_SIZE; ++i) {
        for(int j = 0; j < ROW_SIZE; ++j) {
          int neighbors = calcNeighbors(arr, ROW_SIZE, SPLICE_SIZE / ROW_SIZE, i, j);

          if(neighbors == 0 || neighbors == 1) {
            tmpArr[ROW_SIZE * i + j] = 0;
          }
          else if(neighbors == 3) {
            tmpArr[ROW_SIZE * i + j] = 1;
          }
          else if(neighbors > 3) {
            tmpArr[ROW_SIZE * i + j] = 0;
          }
        }
      }

      MPI_Send(tmpArr, SPLICE_SIZE, MPI_INT, 0, 0, MCW);
    }

  }
```

```
  MPI_Finalize();
  return 0;
}
```

To verify it works here is the command line arguments and output It worked, here is the code:

```
$ mpic++ parallel.cpp
$ mpirun -np 8 -oversubscribe a.out

 TIME: 14776 microseconds
 ...
 TIME: 13204 microseconds
 ...
 TIME: 13237 microseconds
 ...
 TIME: 12819 microseconds
 ...
```

The '...' is where the board is shown, but it is too large to show in this report. For better understanding we will show an 8x8 board.

```
$ mpic++ parallel.cpp
$ mpirun -np 8 -oversubscribe a.out

 Full Board at Iteration 0:
 - - - - - - - -
 - O - - - - - -
 - - O O - - - -
 - O O - - - - -
 - - - - - - - -
 - - - - - - - -
 - - - - - - - -
 - - - - - - - -

 TIME: 1299 microseconds
 Full Board at iteration 1:
 - - - - - - - -
 - - O - - - - -
 - - - O - - - -
 - O O O - - - -
 - - - - - - - -
 - - - - - - - -
 - - - - - - - -
 - - - - - - - -

 TIME: 81 microseconds
 Full Board at iteration 2:
 - - - - - - - -
 - - - - - - - -
 - 0 - 0 - - - -
 - - 0 0 - - - -
 - - 0 - - - - -
 - - - - - - - -
 - - - - - - - -
 - - - - - - - -

 TIME: 52 microseconds
 Full Board at iteration 3:
 - - - - - - - -
 - - - - - - - -
 - - - 0 - - - -
 - 0 - 0 - - - -
 - - 0 0 - - - -
 - - - - - - - -
 - - - - - - - -
 - - - - - - - -
```

As you can see the glider is working across the 8 processor splices. We have created a parallel Game of Life.