Final Project - Conway's Game of Life

Niklas Bergqvist
Parallel and Distributed Programming

May 24, 2018

1 Introduction

The purpose of this project is to do an MPI implement of John Conway's Game of Life. The implementation will be tested on three different grids and running on up to 16 processors and the code will be written c/c++ language

2 Problem Description

The Game of Life is a Cellular Automaton (CA) model, that is visualized as cells on a square grid, developed by John Conway in the 1970s. The cells have two states, dead or alive, also each cell has 8 neighbors (Moore neighborhood) and the rules are:

- Any live cell with fewer than two live neighbours dies, as if caused by under population.
- Any live cell with two or three live neighbors lives on to the next generation.
- Any live cell with more than three live neighbors dies, as if by overpopulation.
- Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

3 Solution Method

The parallelization of the game will be done using message passing interface (MPI) for c/c++ programming language. The solution uses an odd-even send and receive scheme with standard blocking MPI commands such as MPI_Send and MPI_Recv. The scheme is set up so that even processes send, then receive and odd processes receive first, then send. The load on the processors is sliced up row-wise on the grid.

4 Experiments

4.1 Results

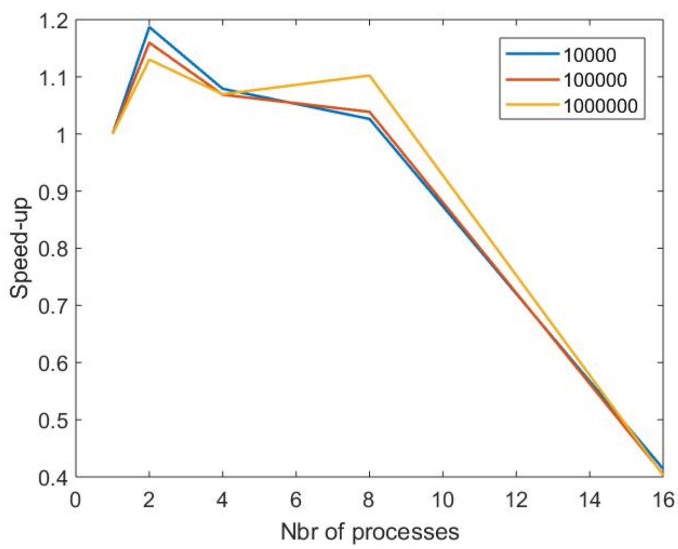

Figure 1: Plot of the Speed-up for 16x16 grid. Run in the computer lab 2510.

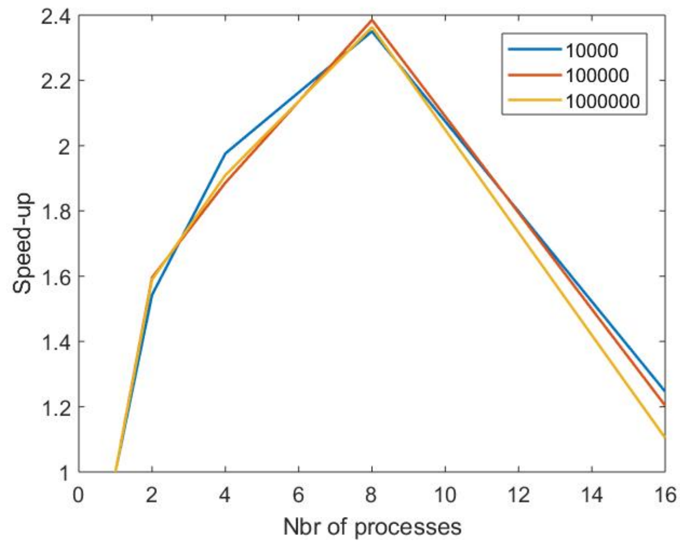

Figure 2: Plot of the Speed-up for 32x32 grid. Run in the computer lab 2510.

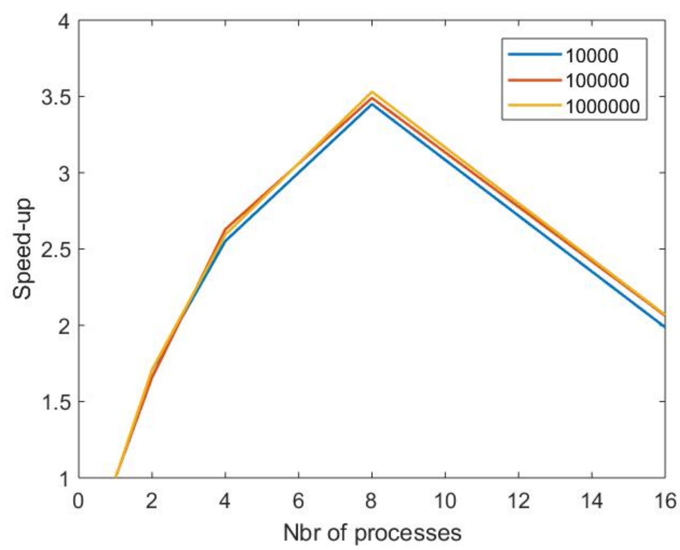

Figure 3: Plot of the Speed-up for 48x48 grid. Run in the computer lab 2510.

Figure 1 shows the speed-up of the Game of Life on a 16x16 grid for 10000, 100000 and 100000 iterations. It also shows that the speed-up is largest for 2 processors, peaking at 1.2x and after 9 processors the speed-up becomes lower than 1. In figure 2 and Figure 3, the speed-up is shown for a 32x32 and 48x48 gird respectively. For both of the grids in Figure 2 and 3 the speed-up peaks at 9 processors. For the 32x32 grid the maximum speed-up is about 2.4x and for the 48x48 grid the speed-up is about 3.5. Figure 4,

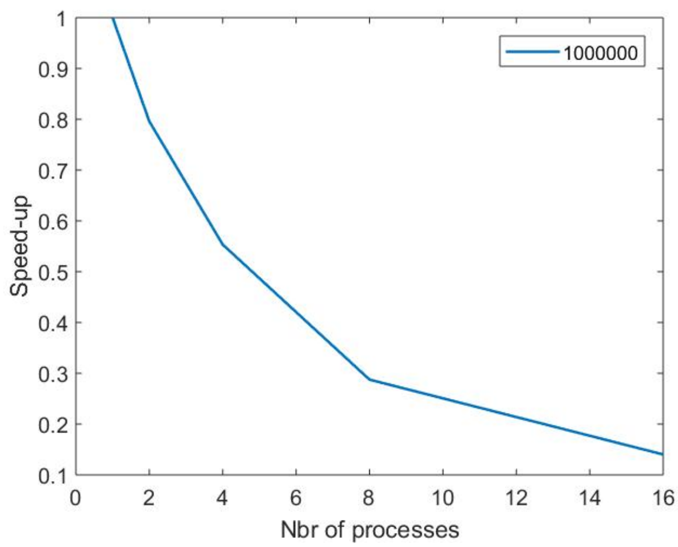

Figure 4: Plot of the Speed-up for 48x48 grid. Run on Milou cluster.

shows the speed-up when the 48x48 is run on the Milou-cluster for 100000 iterations and it shows a speed-up less than 1 throughout all of the number of processes.

4.2 Test of Solution

The graphic test output of the glider test is shown in the Appendix and it shows how the glider progresses from the initial state to state 5.
Figure 4 shows what the glider pattern progression should look like.

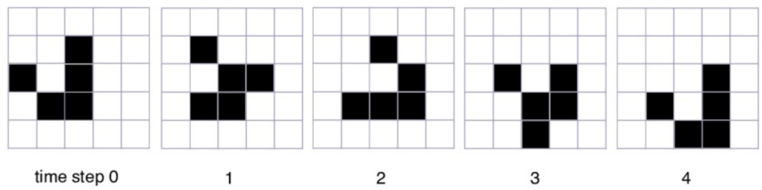

Figure 5: Diagram of the glider pattern that emerges from the rules of Game of Life.

5 Conclusions

From the results we can see that there is essentially no difference between the number of iterations in terms of speed-up. We can also conclude that the speed-up is largest for the 48x48 grid at 3.5x. It seems from Figure 2 and 3 that 9 processesors is optimal for this problem. Although while Figure 1 shows that 2 processors is optimal the 16x16 grid is too smal too say much about the problem in general. The test on the Milou-cluster shows no speed-up at all, in fact it gets slower and slower the more processors are used.

From the graphical test we can observe a similar glider pattern compared to the one in the provided diagram, although the glider pattern is shifted in the test case it still exibits the same characteristics as the diagram in Figure 5. That is, we can conclude that the solution behaves according to the rules of GoL and therefore the solution is correct based on this test.

6 Code

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

#define ROOT 0 //define the root process
#define N 48   //dimension for the square grid
#define GRID_SIZE 2304 //  N   N=16,32,48 GRID_SIZE = 256, 1024, 2304

int mod(int a, int b); //mod function
int main(int argc, char **argv)
{
    //set inital grid matrix.
    //Alive cell = 1, Dead cell = 0.


     /*   int global_grid[GRID_SIZE] =
        {0, 1, 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, 0, 0, 0, 0, 0,
         1, 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, 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, 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, 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, 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,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};*/


/*
  int global_grid[GRID_SIZE] =
        {0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 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,
         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, 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, 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, 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, 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, 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, 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, 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, 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, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 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,
       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, 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, 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, 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, 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, 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, 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, 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, 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 global_grid[GRID_SIZE] =
     {0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 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,
      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, 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, 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, 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, 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, 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, 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, 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, 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, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 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,
      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, 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, 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, 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, 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, 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, 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, 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, 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, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 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,
     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, 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, 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, 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, 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, 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, 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, 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, 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,


//parameters
int first_out[N];
int last_out[N];
int first_in[N];
int last_in[N];

int j;
int nGen = 0; //iteration counter
int generations; //nbr of generations

double T1; //start time

generations = atoi(argv[1]); //read the nbr of generation to run on

//MPI parameters
int my_rank;
int nprocs;
MPI_Status stat;

// MPI Setup;
MPI_Init(&argc, &argv);

MPI_Comm_size(MPI_COMM_WORLD, &nprocs); //set nprocs
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank); //get my rank


//initialize timer
if (my_rank == ROOT)
{
   T1 = MPI_Wtime();
}

//allocate memory for the local array
int*local = (int*)malloc(N*((N/nprocs) + 2)*sizeof(int));

//slice up the data in strips based on rows
for (nGen = 0; nGen < generations; nGen++)
{
    j = N;
    for (int i = my_rank*(GRID_SIZE/nprocs); i < (my_rank + 1)*(GRID_SIZE/nprocs); i++)
    {
```

```
        local[j] = global_grid[i];
        j++;
    }


    if (nprocs > 1)
    {
        //odd-even send and recieve scheme
        if (my_rank%2 == 0)
        //even
        {
            for (int i = 0; i < N; i++)
            {
                first_in[i] = local[i + N];
            }

            //Send first row using non-blocking send function
            MPI_Send(&first_in, N, MPI_INT, mod(my_rank - 1, nprocs), 1, MPI_COMM_WORLD);

            for (int i = 0; i < N; i++)
            {
                last_in[i] = local[(N*(N/nprocs)) + i];
            }
            //Send last row
            MPI_Send(&last_in, N, MPI_INT, mod(my_rank + 1, nprocs), 1, MPI_COMM_WORLD);
        }
        else
         //odd
        {
            MPI_Recv(&last_out, N, MPI_INT, mod(my_rank + 1, nprocs), 1, MPI_COMM_WORLD, &stat)
            MPI_Recv(&first_out, N, MPI_INT, mod(my_rank - 1, nprocs), 1, MPI_COMM_WORLD, &stat
        }
        if (my_rank%2 == 0)
        //even
        {
            MPI_Recv(&last_out, N, MPI_INT, mod(my_rank + 1, nprocs), 1, MPI_COMM_WORLD, &stat)
            MPI_Recv(&first_out, N, MPI_INT, mod(my_rank - 1, nprocs), 1, MPI_COMM_WORLD, &stat
        }
        else
        //odd
        {
            for (int i = 0; i < N; i++)
            {
                first_in[i] = local[i + N];
            }
            MPI_Send(&first_in, N, MPI_INT, mod(my_rank - 1, nprocs), 1, MPI_COMM_WORLD);

            for (int i = 0; i < N; i++)
            {
                last_in[i] = local[(N*(N/nprocs))+i];
            }
            MPI_Send(&last_in, N, MPI_INT, mod(my_rank + 1, nprocs), 1, MPI_COMM_WORLD);
        }

        for (int i = 0; i < N; i++)
        {
            local[i] = first_out[i];
```

```
            local[(N*((N/nprocs)+1))+i] = last_out[i];
        }
    }

    else
    //if nprocs = 1, no sending
    {
        for (int i = 0; i < N; i++)
        {
            local[i + GRID_SIZE + N] = global_grid[i];
        }
        for (int i = GRID_SIZE; i < GRID_SIZE + N; i++)
        {
            local[i - GRID_SIZE] = global_grid[i - N];
        }
    }
    //allocate memory for the temporary array
    int*temp = (int*)malloc(N*((N/nprocs))*sizeof(int));

    //run Game of Life
    for (int k = N; k < N*((N/nprocs)+1); k++)
    {
       int rows = N*(N/nprocs)+2;
       int r = k/N;
       int c = k%N;

       //we need to know the previous and the next row and column
       int p_row = mod(r - 1, rows); //previous row
       int p_col = mod(c - 1, N);    //previous column
       int n_row = mod(r + 1, rows); //next row
       int n_col = mod(c + 1, N);    //next column

       //count the nbr of neighbors of the current cell
       int sum = local[p_row*N+p_col]+local[p_row*N+c]+local[p_row*N+n_col]+local[r*N+p_col]+lo

       //apply the rules of GoL
       if (local[k] == 1) //Alive cell = 1
       {
            if (sum < 2)
            {
                temp[k - N] = 0; //any alive cell with fewer than 2 alive neighbors dies
            }
            else if (sum > 3)
            {
                temp[k - N] = 0; //any alive cell with more than 3 alive neighbors dies
            }
            else
            {
                temp[k - N] = 1; //any dead cell with exactly 3 alive neighbors becomes an aliv
            }
       }
       else
       {
            if (sum == 3)
            {
                temp[k - N] = 1; //cell lives on
            }
```

```
                else
                {
                    temp[k - N] = 0;
                }
            }
        }

        j = 0;
        for (int i = my_rank*(GRID_SIZE/nprocs); i < (my_rank + 1)*(GRID_SIZE/nprocs); i++)
        {
            global_grid[i] = temp[j];
            j++;
        }

        //Gather grid back to ROOT
        MPI_Gather(temp, N*(N/nprocs), MPI_INT, &global_grid, N*(N/nprocs), MPI_INT,ROOT, MPI_COMM_

        // Output the updated grid state
        if (my_rank == ROOT)
        {
           //  printf("\nIteration %d: temp grid:\n", nGen);
            for (j = 0; j < GRID_SIZE; j++)
            {
                if (j%N == 0)
                {
                //    printf("\n");
                }
              //  printf("%d  ", global_grid[j]);
            }
            //printf("\n");
        }
    }

      if (my_rank == ROOT)
      {
            //Print the wall time after the last generation
            printf("Parallel␣game␣over␣after:␣%lf␣seconds\n", MPI_Wtime() - T1);
      }

    free(local);    //Free up memory
    MPI_Finalize(); //Finalize MPI
}

//Modulus function
int mod (int a, int b)
{
   if(b < 0)
  {
     return mod(a, -b);
  }
   int res = a%b;
   if(res < 0)
   {
     res+=b;
   }
   return res;
```

```
}
```

7 Appendix

7.1 The command prompt output

```
16x16

nibe7223@hedenius:~/Documents$ mpirun -np 1 ./gameoflife 10000
Parallel game over after: 0.050105 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 2 ./gameoflife 10000
Parallel game over after: 0.042209 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 4 ./gameoflife 10000
Parallel game over after: 0.046437 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 8 ./gameoflife 10000
Parallel game over after: 0.048822 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 16 ./gameoflife 10000
Parallel game over after: 0.120987 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 1 ./gameoflife 100000
Parallel game over after: 0.483419 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 2 ./gameoflife 100000
Parallel game over after: 0.416835 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 4 ./gameoflife 100000
Parallel game over after: 0.452337 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 8 ./gameoflife 100000
Parallel game over after: 0.465415 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 16 ./gameoflife 100000
Parallel game over after: 1.195265 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 1 ./gameoflife 1000000
Parallel game over after: 4.836294 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 2 ./gameoflife 1000000
Parallel game over after: 4.279489 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 4 ./gameoflife 1000000
Parallel game over after: 4.521821 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 8 ./gameoflife 1000000
Parallel game over after: 4.387478 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 16 ./gameoflife 1000000
Parallel game over after: 12.004856 seconds
```

```

32x32

nibe7223@hedenius:~/Documents$ mpirun -np 1 ./gameoflife 10000
Parallel game over after: 0.191365 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 2 ./gameoflife 10000
Parallel game over after: 0.124039 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 4 ./gameoflife 10000
Parallel game over after: 0.096842 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 8 ./gameoflife 10000
Parallel game over after: 0.081426 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 16 ./gameoflife 10000
Parallel game over after: 0.153559 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 1 ./gameoflife 100000
Parallel game over after: 1.912307 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 2 ./gameoflife 100000
Parallel game over after: 1.198324 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 4 ./gameoflife 100000
Parallel game over after: 1.013816 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 8 ./gameoflife 100000
Parallel game over after: 0.801756 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 16 ./gameoflife 100000
Parallel game over after: 1.589086 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 1 ./gameoflife 1000000
Parallel game over after: 19.085168 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 2 ./gameoflife 1000000
Parallel game over after: 12.008185 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 4 ./gameoflife 1000000
Parallel game over after: 9.994661 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 8 ./gameoflife 1000000
Parallel game over after: 8.076402 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 16 ./gameoflife 1000000
Parallel game over after: 17.280834 seconds

48x48

nibe7223@hedenius:~/Documents$ mpirun -np 1 ./gameoflife 10000
Parallel game over after: 0.507366 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 2 ./gameoflife 10000
Parallel game over after: 0.297148 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 4 ./gameoflife 10000
```

```
Parallel game over after: 0.198794 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 8 ./gameoflife 10000
Parallel game over after: 0.147087 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 16 ./gameoflife 10000
Parallel game over after: 0.255395 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 1 ./gameoflife 100000
Parallel game over after: 5.053464 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 2 ./gameoflife 100000
Parallel game over after: 3.048049 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 4 ./gameoflife 100000
Parallel game over after: 1.922138 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 8 ./gameoflife 100000
Parallel game over after: 1.448096 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 16 ./gameoflife 100000
Parallel game over after: 2.449751 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 1 ./gameoflife 1000000
Parallel game over after: 50.493111 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 2 ./gameoflife 1000000
Parallel game over after: 29.532493 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 4 ./gameoflife 1000000
Parallel game over after: 19.468946 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 8 ./gameoflife 1000000
Parallel game over after: 14.302638 seconds

nibe7223@hedenius:~/Documents$ mpirun -np 16 ./gameoflife 1000000
Parallel game over after: 24.407654 seconds
```

Output on milou

```
[ohnahhh@milou1 ~]$ mpicc -O3 -o gameoflife gameoflife.c -lm -std=c99
[ohnahhh@milou1 ~]$ sbatch gameoflife.sh
Submitted batch job 12235743
[ohnahhh@milou1 ~]$ mpirun -np 1 --map-by core ./gameoflife 1000000
Parallel game over after: 46.102480 seconds
[ohnahhh@milou1 ~]$ mpirun -np 2 --map-by core ./gameoflife 1000000
Parallel game over after: 57.935312 seconds
[ohnahhh@milou1 ~]$ mpirun -np 4 --map-by core ./gameoflife 1000000
Parallel game over after: 83.313487 seconds
[ohnahhh@milou1 ~]$ mpirun -np 8 --map-by core ./gameoflife 1000000
Parallel game over after: 160.351554 seconds
[ohnahhh@milou1 ~]$ mpirun -np 16 --map-by core ./gameoflife 1000000
Parallel game over after: 328.793513 seconds
```

7.2 Output of the glider pattern test

Initial grid

```
        {0, 1, 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, 0, 0, 0, 0, 0,
         1, 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, 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, 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, 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, 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,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
```

Iteration 0

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 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 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 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 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 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 0
```

Iteration 1

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Iteration 2

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
4	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Iteration 3

```
0 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 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 1 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 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 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 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 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
```

References

[1] http://faculty.ycp.edu/ dhovemey/spring2011/cs365/lecture/lecture7.html

[2] https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

[3] https://en.wikipedia.org/wiki/Glider_(Conway%27s_Life)