CS5500 HW8

Chase Mortensen A01535275

November 2019

1 Description

Make a game of life. Create a 'world' of 1024 by 1024 cells, without edge connections (i.e., the world is 'flat' with 'edges' that organisms can 'fall off' of).

The rules for the game of life are:

1. every organism that has 0 or 1 neighbors dies of loneliness.
2. every organism that has 2 or 3 neighbors survives to the next day.
3. every organism that has 4 or more neighbors dies of overcrowding.
4. every cell that has exactly three neighbors gives birth to a new organism.

Test to see if your program is working by creating a 'glider gun' at one edge of the world and tracking it as it crosses the problem space.

After you are sure that your program is working correctly, seed the entire space with organisms so that there is a 1-in-5 probability that any one cell initially contains an organism.

Be sure to correctly implement pseudo-random number generators over your processor set.

That being accomplished, make your program run as quickly as possible. Provide timing information on a 'per day' basis, and chart your program versus the number of processors used.

The idea behind my implementation is that process 0 has the master copy of the game world and redistributes it each day. Each column is computed by the i%size processor which sends the column data to process zero, which compiles all of the data, prints the world, and then redistributes.

My implementation works on one processor and implements all of the rules of the game of life. However, when it switches to multiple processors, there is an error and the world becomes empty. It was creating a seg fault, but it's closer to being correct now.

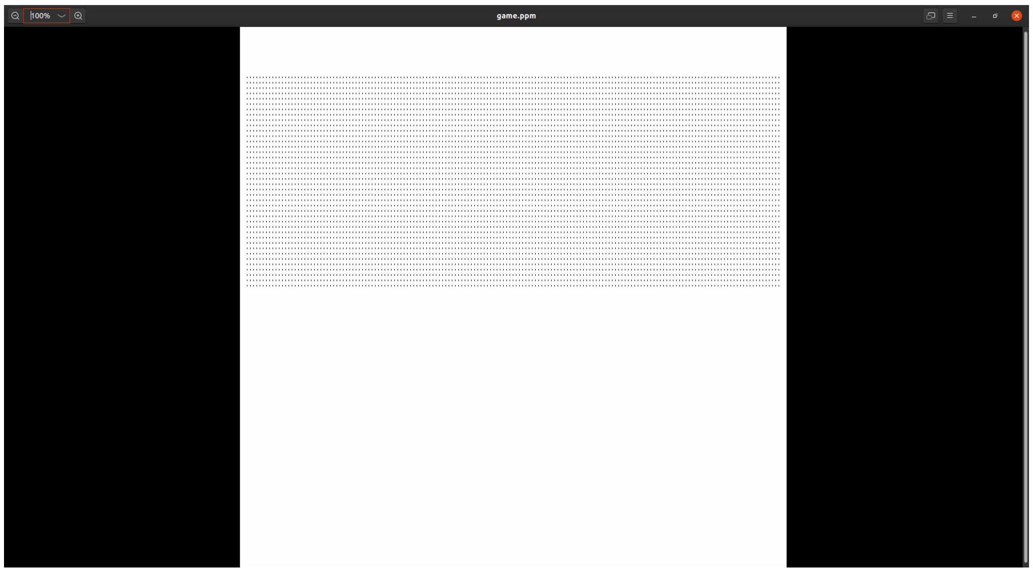
2 Program

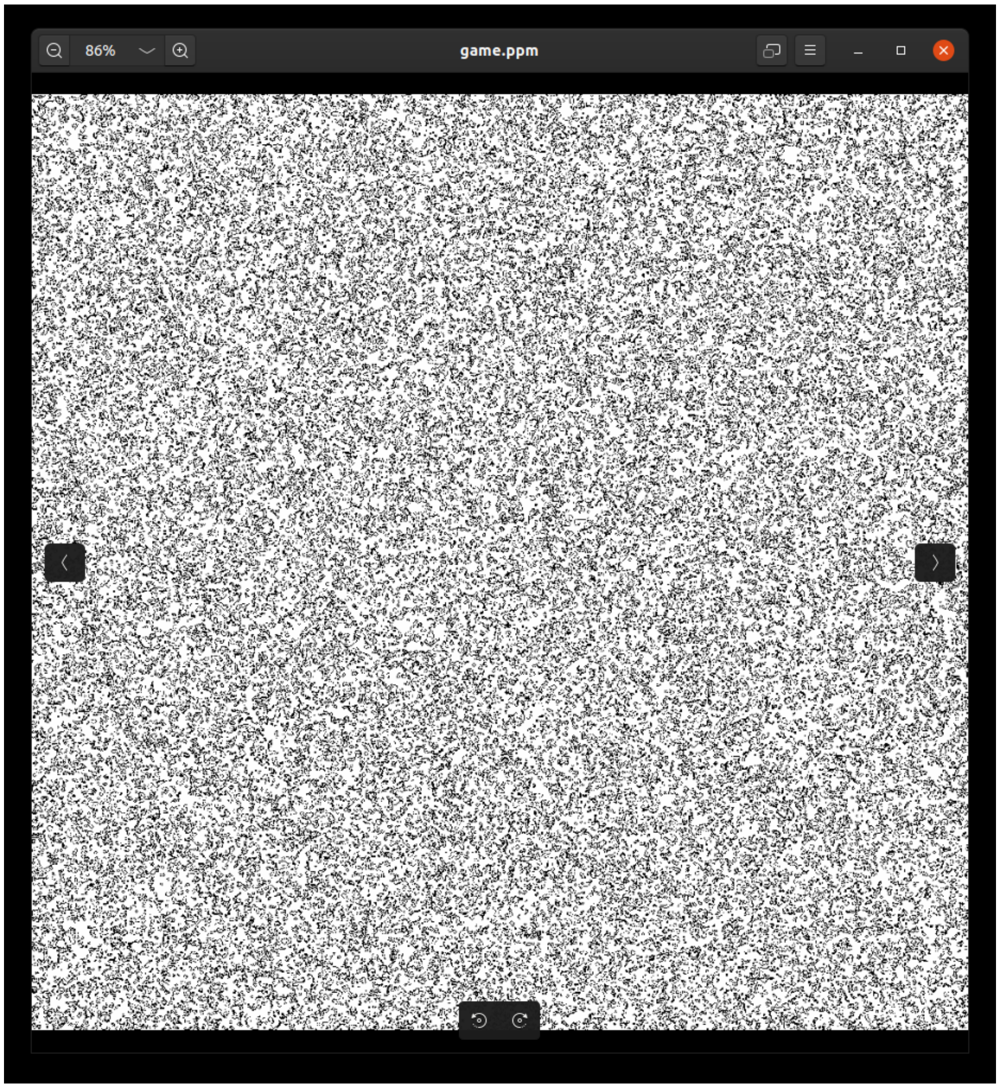
```
#include <iostream>
#include <fstream>
#include <math.h>
#include <mpi.h>
#include <vector>
#include <unistd.h>
#include <cstdlib>

#define MCW MPI_COMM_WORLD
using namespace std;

int pixels = 1024;
vector<vector<bool>> vec_0; // read
// vec_1 is for individual columns and is used for dividing tasks between processes
vector<vector<bool>> vec_2; // write
int neighbors;

bool isAlive(int i, int j, vector<vector<bool>>& v) {
    neighbors = 0;
    // cout << "isAlive " << i << " " << j << endl;
    // cout << "v.size() = " << v.size() << endl;
    // cout << "vec_0[0].size() = " << vec_0[0].size() << endl;
    // for (int x = 0; x < v.size(); x++){
    //     for (int y = 0; y < v[x].size(); y++){
    //         cout << v[x][y] << " ";
    //     }
    //     cout << endl;
    // }

    if (i > 0){
        if (v[i-1][j]) neighbors++;
        if (j > 0) {
            if (v[i-1][j-1]) neighbors++;
        }
        if (j < pixels - 1){
            if (v[i-1][j+1]) neighbors++;
        }
    }
    if (i < pixels - 1){
        if (v[i+1][j]) neighbors++;
        if (j > 0) {
            if (v[i+1][j-1]) neighbors++;
        }
        if (j < pixels - 1){
            if (v[i+1][j+1]) neighbors++;
        }
    }
    if (j > 0){
        if (v[i][j-1]) neighbors++;
    }
    if (j < pixels - 1){
        if (v[i][j+1]) neighbors++;
    }

    if (neighbors == 2 && v[i][j]) return true;
    else if (neighbors == 3) return true;
    return false;
}

void initialize(){
    int tmp;
    for (int i = 0; i < pixels; i++){
        vector<bool> v(pixels, false);
        vec_0.push_back(v);
        vec_2.push_back(v);
    }

    bool change = false;
    for (int i = 100; i < 500; i += 10){ // blinkers
        for (int j = 10; j < 1014; j++){
            if (j%3 == 0){
                change = !change;
            }
            vec_0[i][j] = change;
        }
    }

    vec_0[4][4] = true;
    vec_0[3][4] = true;
    vec_0[5][4] = true;


    // for (int i = 0; i < pixels; i++){
    //     for (int j = 0; j < pixels; j++){
    //         vec_0[i][j] = (rand()%2 == 0); // random static
    //     }
    // }

    ofstream fout("game.ppm");
    fout << "P3" << endl;
    fout << pixels << " " << pixels << endl;
    fout << "256" << endl;


    for (int i = 0; i < pixels; i++){
        for (int j = 0; j < pixels; j++) {
            tmp = 0;
            if (!vec_0[i][j]) tmp = 255;
            fout << " " << tmp << " " << tmp << " " << tmp << "  ";
        }
    }
    fout.close();
}

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

    double t1, t2;
    int rank, size;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MCW, &rank);
    MPI_Comm_size(MCW, &size);


    vector<bool> vec_1(pixels, false);

    bool pixel;
    bool tmp;
    int print_pixel;

    initialize();

    while(true){

        MPI_Barrier(MCW);


        if (!rank){
            std::ofstream ofs ("tmp.ppm", std::ios::out | std::ios::trunc);
            ofs.close();
        }

        ofstream fout("tmp.ppm");
        if (!rank){
            fout << "P3" << endl;
            fout << pixels << " " << pixels << endl;
            fout << "256" << endl;
        }

        if (!rank){
            for (int i = 0; i < pixels; i++){
                for (int j = 1; j < size; j++){
                    MPI_Send(&vec_0[i],1,MPI_C_BOOL,j,0,MCW);
                }
            }
        }
        if (rank){
            for (int i = 0; i < pixels; i++){
                for (int j = 1; j < size; j++){
                    MPI_Recv(&vec_0[i],1,MPI_C_BOOL,0,0,MCW,MPI_STATUS_IGNORE);
                }
            }
        }

        for (int i = 0; i < pixels; i++){

            MPI_Barrier(MCW);

            if ((i % size) == rank) {

                for (int j = 0; j < pixels; j++){
                    tmp = isAlive(i,j,vec_0);
                    vec_1[j] = tmp;
                }
                if (rank > 0) {
                    MPI_Send(&vec_1,1,MPI_C_BOOL,0,0,MCW);
                }
            }

            if (!rank) {
                if ((i % size) != 0) {
                    MPI_Recv(&vec_1,1,MPI_C_BOOL,MPI_ANY_SOURCE,0,MCW,MPI_STATUS_IGNORE);
                }
                for (int k = 0; k < pixels; k++) {
                    if (vec_1[k]){
                        print_pixel = 0;
                    }
                    else print_pixel = 255;

                    vec_2[i][k] = vec_1[k];
```

```
                    fout << " " << print_pixel << " " << print_pixel << " " << print_pixel
                }
                fout << endl;
            }

        }

        if (!rank)
           vec_0 = vec_2;
        fout.close();
        usleep(1000000);

        if (!rank){
            std::ifstream  src("tmp.ppm", std::ios::binary);
            std::ofstream  dst("game.ppm", std::ios::binary);
            dst << src.rdbuf();
        }
    }

    MPI_Finalize();

    return 0;
}
```

3 Output

```
$ mpic++ game_of_life.cpp
$ mpirun -np 1 ./a.out
```


100%
game.ppm


86%
game.ppm