**CITS3402 High Performance Computing**

***Assignment 1***

Cunjun Yin (22249435)

Qianhao Liu (21952083)

**Compilation of the program**

Gcc –std=c99 -fopenmp -o gol GoL\_MP.c

./golm 2048 100 1

./golm 2048 100 2

* The first parameter is the array size
* The second parameter is the number of steps want to simulate
* The thirds parameter can only be 1(execute the program using OpenMp) or 2(Sequentially)

**Problem statement**

This is the first Assignment of the unit CITS3402 High Performance Computing. The major purpose of this assignment is to see the different performance on using a single thread and multiple threads parallelism to execute the program that implemented a famous game “Game-of-Life”. The game we implemented in the problem is the normal “Game of Life” using 8-neighborhood to decide the next generation of each cell is alive or dead.

The game rules are as follow:

* A live cell dies if it has less than two live neighbours due to loneliness;
* A live cell with four live neighbours die due to overpopulation;
* A live cell with two or three live neighbours continues to live;
* Any dead cell with exactly three live neighbours becomes live due to reproduction;

**Implementation of Game of Life(GoL)**

There will be two versions of implementation of the games, one is implemented using parallelism, another one is only implemented sequentially, there will be comparison between these two-mechanism using different array size, different and different number of threads to parallelize the program.

The main function takes three parameters, the first parameter is the array size, the second, the second parameter is the number of steps that need to be simulated, and the last parameter is either 1 or 2, it controls whether execute the program with OpenMP or sequentially.

Firstly, all the alive cells are assigned by the value of 1 and the dead cells are assigned by the value of -1. Then, according to the first parameter, a 2-D array will be initialized with all the elements equal to -1 using dynamically memory allocation

int\*\*malloc2DArray(int m, int n)

After the success of memory allocation for both map and new map, the program will be executed whether using parallelism with the third parameter 1 or sequentially with third parameter 2, and the number of steps wants to simulate will depends on the second parameter.

The “detect” and “fate”, both function take number of columns and rows as parameters, function “dectet” will return an integer that records how many neighbours are around a single cell. Both functions will be used to judge a single cell on the current map whether it will be dead or alive depends on the integer returned form “detect”, then, the “fate” function will assign the modified value from the current map to the new map in the same cell location. Then, by loop through each cell and assign all the elements in the current map to the new map which is handled by “display”

int detect(int row, int col)

void fate(int row, int col)

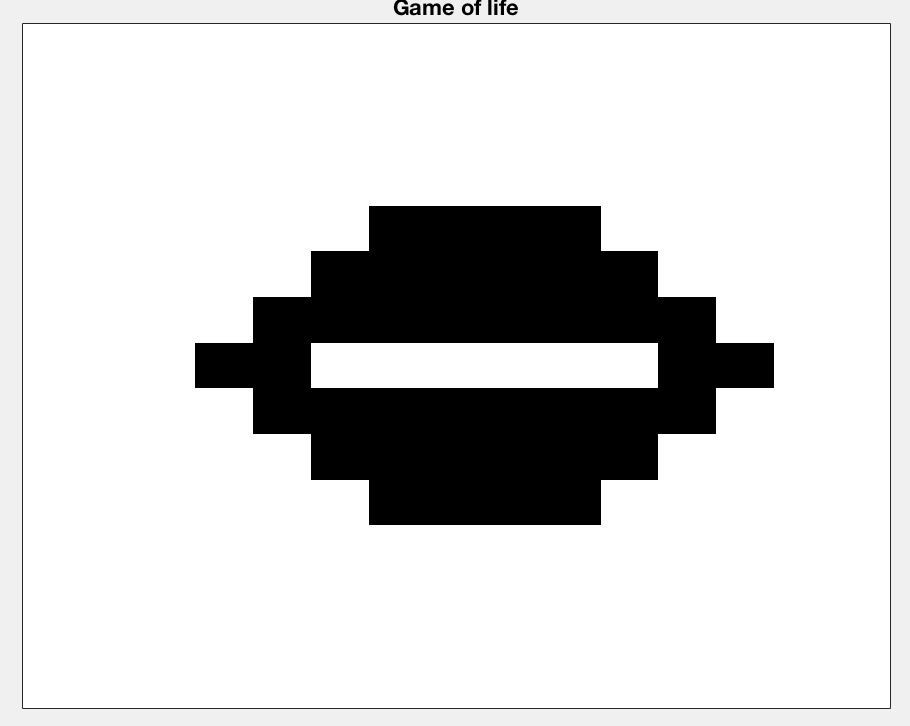
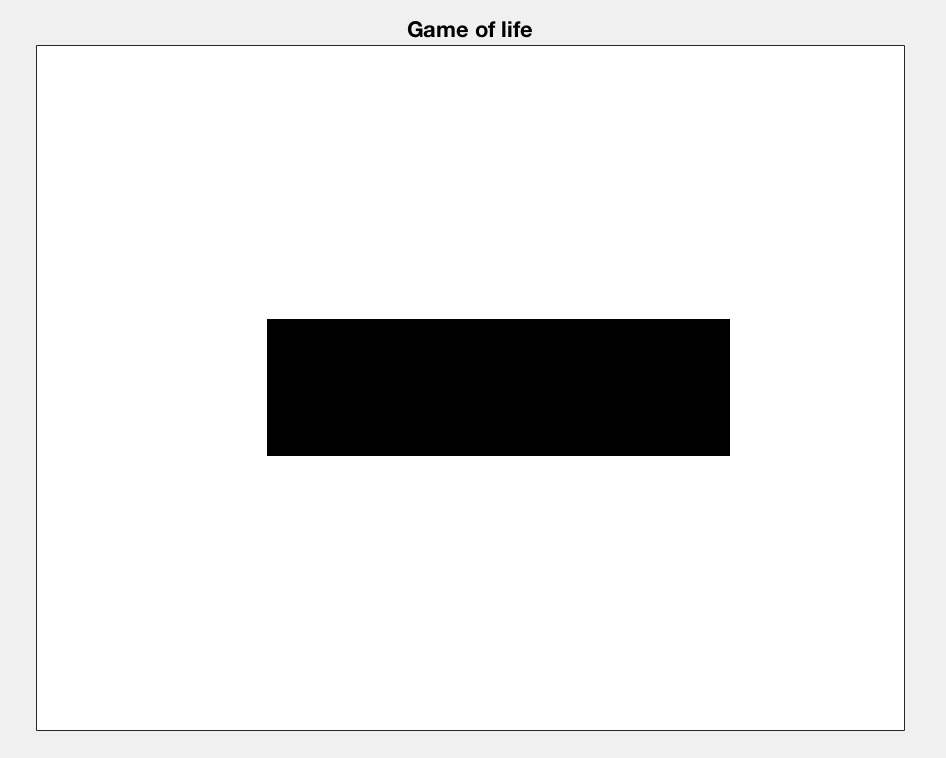
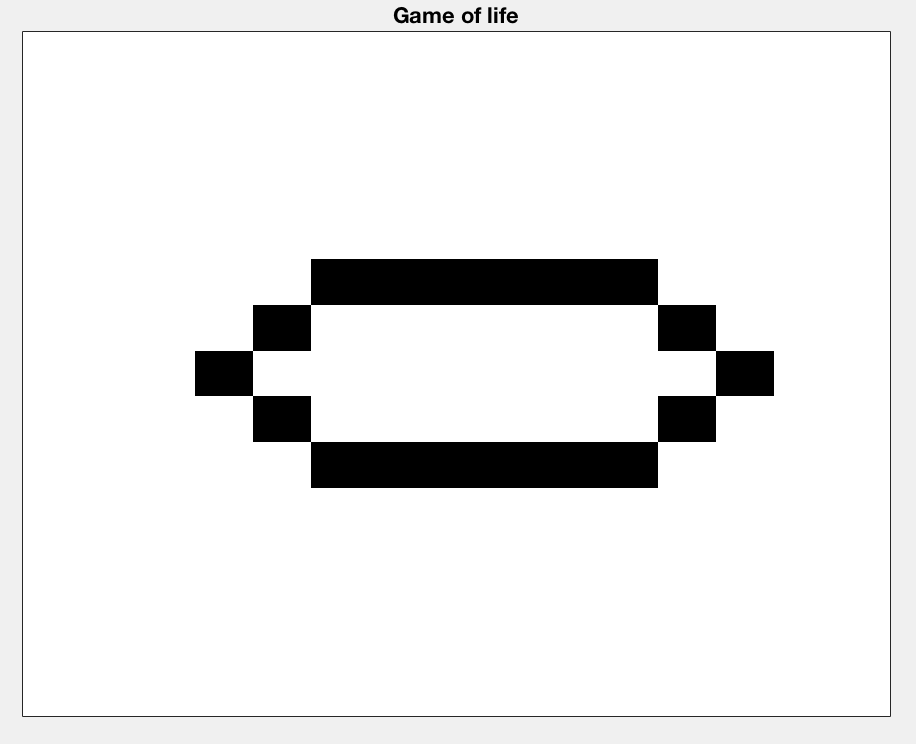
void display()

Finally, the “swapAndClean” function takes current map and new map and number of rows and columns as parameter and copy all the elements from new map to current map and set all the cells in the new map as dead cell.

void swapAndClean(int\*\*a , int \*\*b,int row, int col)

**Test Description and Results**

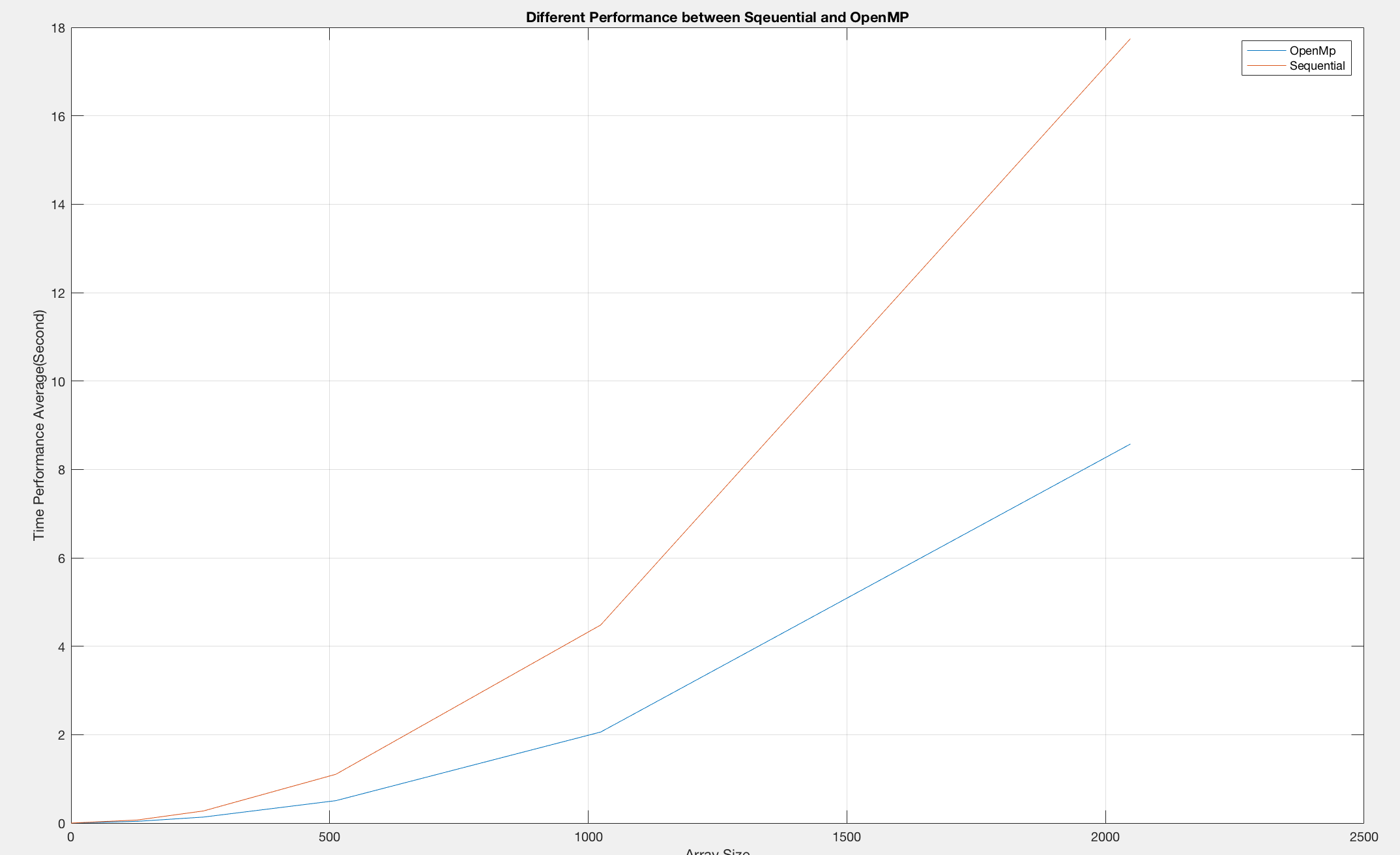
* To show the Game of Life are implemented correctly blow is the first three steps simulation:

1.  2. 3.

* First experiment is about comparing time consuming between and sequential on different array size with 100 steps simulation:

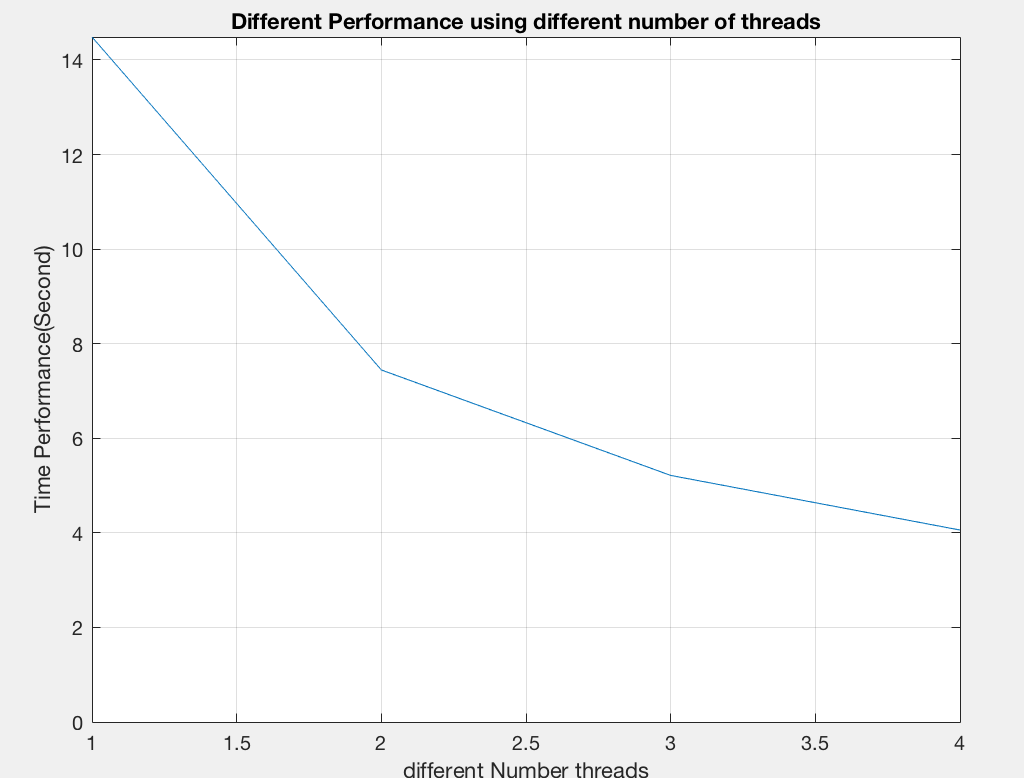
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Array Size | 128 | 256 | 512 | 1024 | 2048 |
| parallelism (Sec) | 0.0385180 | 0.13421000 | 0.51832600 | 2.11335100 | 8.9652980000 |
| Sequential  (sec) | 0.0690080 | 0.274613000 | 1.117904000 | 4.60452100 | 17.673390000 |

* The graph blew shows time spend between parallelism and sequential
* (Red is for sequential, blue is for Parallelizing)



* Second experiment is about to see the difference on time consuming using different number of threads

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of threads | 1 | 2 | 3 | 4 |
| Time spend(Sec) using array size of 2048 | 14.4851200000 | 7.4444410000 | 5.2141410000 | 4.0594830000 |

* The graph shows as the number of threads increase the time consuming is decreasing.

**Conclusion**

The goal of the assignment is to study the speed up by using OpenMp, OpenMp is an application programming interface(API) that support multiprocessing in C, C++ and Fortran, OpenMp can speed up the execution time by using parallel processing that is the processing of the programme by dividing them among multiple processor. Therefore, Theoretically, the more processor a computer has, the faster the execution time on the computer. But according to the second experiment we did, as the thread number increasing, the decreasing rate of the execution time is decreasing as well. Therefore, as a result, there will be a limit on the number of threads using in parallelism, when the limit is reached, there will be no more decreasing on execution time.

Also, to maximum the advantage of OpenMp parallelism, the parts where need to be paralleled is the place that take huge amount of computational cost. Therefore, in the implementation of the Game of Life, the part is paralleled is where the function decides each single cell is alive or dead in the next generation and swapping the new map and old map. Also, the variable that are using during in parallelism need to be controlled as well, those variables needs to be set to private, first-Private, last-Private etc. depending on the situation. In our programming, variables are set to be private during parallelism, so each thread have a copy of the variable and don’t have to wait to access the variable.

Therefore, OpenMp is a powerful API, but the way using parallelism provided by OpenMP are important, if incorrect method is used in the programme, it will be even slower than sequential.