CENG478 Project Report

Berkin Kerim Konar
Student No: 2375343

Bora Kargı
Student No: 2380566

Yiğit Varlı
Student No: 2381036

1 What is the Problem?

The problem is to apply John Conway's Game of Life in a parallel computing environment adding new rules to challenge the need for communication and load balancing by using parallel computing algorithms.

There are some problems in both sequential and parallel algorithms. Since computers have finite amount of memory and Game of Life is infinite, this leads to problems when the active area encroaches on the border of the array, which may also result in accuracy drop in the existing sequential/parallel algorithm because of assumptions about other cells. In the parallel algorithm, one other problem is splitting the grid such that there is minimum performance decrease. The larger grid means the higher work load. Balancing loads among processors is a very important problem.

2 Why is it Important?

Computer simulations are used in a lot of fields. Indeed, Game of Life is also a very important and popular simulation, it can give us knowledge about the overall processes. Game of Life is undecidable, which means that given an initial pattern and a later pattern, no algorithm exists that can tell whether the later pattern is ever going to appear. Due to this property, getting faster simulation results by using parallel computing techniques makes the problem more important.

3 What are the Existing Solution Methods?

Game of Life is very old and popular cellular automaton devised by John Horton Conway. Due to its popularity, there are many kinds of sequential/parallel algorithm implementations and analysis papers for this simulation. Parallel algorithms are generally developed based on two approaches mentioned in the paper named as ”A performance Analysis of the Game of Life based on parallel algorithm [1]”.

The first one is ”one-dimensional decomposition strategy”. They decompose the grid by row or by column decomposition. Each process calculates a bar area.

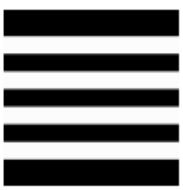


Fig. 1: By row decomposition

Fig. 2: By column decomposition

The second one is ”two-dimensional decomposition strategy”.They decompose the grid into M×N small grids.

Fig. 3: Two-dimensional decomposition

In the paper of ”Conway’s Game of Life accelerated with OpenCL [3]”, they use very interesting algorithm optimization approach. They use OpenCL two-dimensional image objects instead of simple 2D array to optimize performance. Using OpenCL image objects allows them to use the power of GPU more efficiently.

4 Rules

The universe of the Game of Life is an infinite, two-dimensional orthogonal grid of square cells, each of which is in one of two possible states, live or dead, (or populated and unpopulated, respectively). Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:

- Any live cell with two or three live neighbours survives.
- Any dead cell with three live neighbours becomes a live cell.
- All other live cells die in the next generation. Similarly, all other dead cells stay dead.

The initial pattern constitutes the seed of the system. The first generation is created by applying the above rules simultaneously to every cell in the seed; births and deaths occur simultaneously, and the discrete moment at which this happens is sometimes called a tick. Each generation is a pure function of the preceding one. The rules continue to be applied repeatedly to create further generations.

Many new different rules can be applied to Game of Life. We propose a new set of rules where we consider the neighbors of cells and the population of the grids that cells belong to. The map will be divided into grids that have an approximately equal area. We will consider these grids as underpopulated, overpopulated, or normal by counting the alive cells at the grid. An underpopulated grid G is defined as

$$\sigma_G \leq 0.15|G|$$

where σ_G denotes the number of alive cells at grid G and $|G|$ is the size of the grid. Similarly, for overpopulated grids

$$\sigma_G \geq 0.75|G|$$

If an grid is underpopulated, no death will occur in that grid and if an grid is overpopulated, there will be no births. For normal populated grids, 3 rules will be applied:

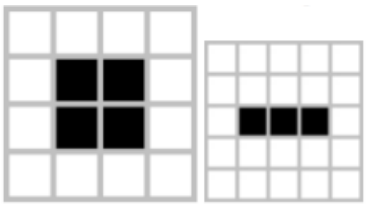
- Any live cell with two or three live neighbours survives.
- Any dead cell with three live neighbours becomes a live cell.

– All other live cells die in the next generation. Similarly, all other dead cells stay dead.

To make sure that processes communicate with each other, every grid will only have the information of cells it is going to change in its buffer.

A new neighbour system will be considered for the cells in the grids. For a cell located at (i,j)th position at a grid, we will consider it's neighbours as the cells in the neighbour grids located at the (i,j)th position of those grids.

Assume that map is divided into M×N small grids. And as we stated before, each grid will have one of the three tags respectively, overpopulated, normal, and underpopulated. We will analyze the behaviours of certain overpopulated/underpopulated grid designs.According to created shape in the map, we will try to change the circumstances of grids. (i.e.) If the shape in the map created by underpopulated grids is block, we will change the cells of diagonal grids and inside the grids, dead cells to alive cells and vice versa.



5 Proposed Method

Our Game of Life consists of newly proposed rules which we stated in section 4. The only difference is that since the number of processes to work on columns and rows are not fixed, we are not able to analyze the shape of overpopulated/underpopulated grids over the map.

Newly added rules take Game of Life to a new dimension. Our proposed solution to new extension of this problem starts with every grid will only have the information of cells it is going to change in its buffer. This is to force processes to communicate with each other. (Do not forget that in our Game of Life, for a cell located at (i,j)th position at a grid, we consider it's neighbours as the cells in the neighbour grids located at the (i,j)th position of those grids. The Map consists of M×N small grids and each grid has alive and dead cells).

Since neighbours of a cell is located in other grids, we always need a communication. There are two possible ways to look through neighbours of each cell. One way is to get information from neighbour grids of cells when it is needed. Our proposed solution uses an alternative way, we keep each neighbour grids of specific grid in its process. So that, we tried to send it as one to avoid unnecessary extra communication time.

After getting all neighbour grids, each process apply rules according to given rules in itself. Before we start to apply rules on new generation, we need to exchange information between all grids since each process has its own virtual memory which cannot be accessed from other processes.

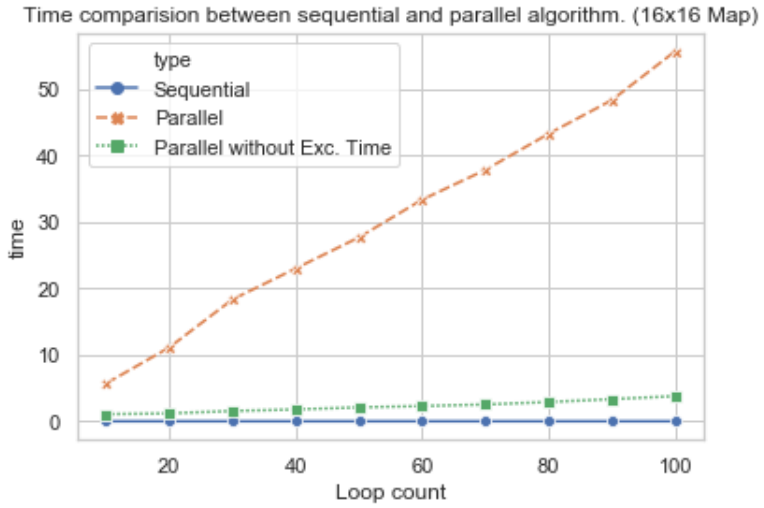
6 Computing Platform

We used our own computer as computing platform instead of Slurm HPC. The computing platform runs Windows10 as operating system. It consists of 8 cores, 2.6GHz Intel i7-4720HQ processor. It has 16GB DDR4 RAM and AMD Radeon R9 M200X series as graphic processor. We have used MPI library and C++ programming language while writing the algorithm.

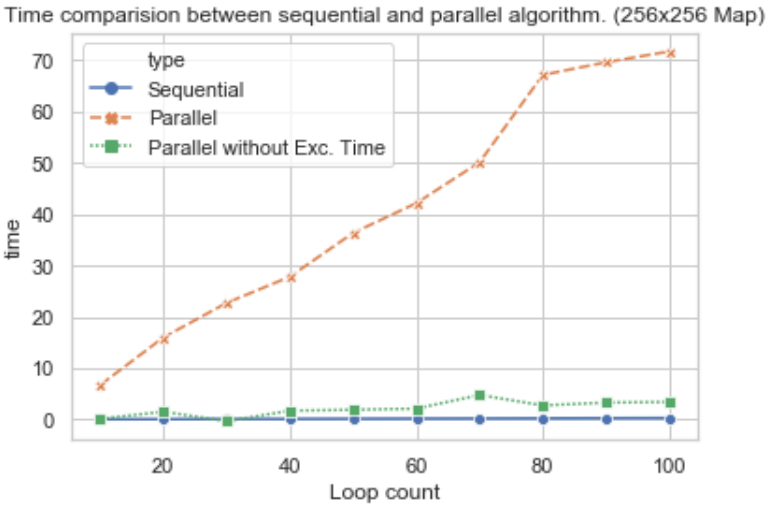
7 Results

We tried our parallel and sequential algorithms on maps of $16 \times 16,\ 256 \times 256,\ 1024 \times 1024,\ 2048 \times 2048,\ 4096 \times 4096,\ 8192 \times 8192$: (width $\times$ height) consisting cells. We used total of 16 processes (grids), 4 per row and 4 per column (Keep in mind that using 8 per row and 2 per column gives different map configuration and resulting generations due to our proposed neighbour system). We started our generation count from 10 and increased 10 by 10 until reaching to 100. For every generation count, we measured the run time of algorithms as an average of 5 runs. Due the nature of our game of life problem, we were not able to test and analyze our parallel algorithm choosing different number of processes per row and column on the same data. Also, we included run times without time spend on exchange information (other communication times are still included) for parallel algorithm in graphs.

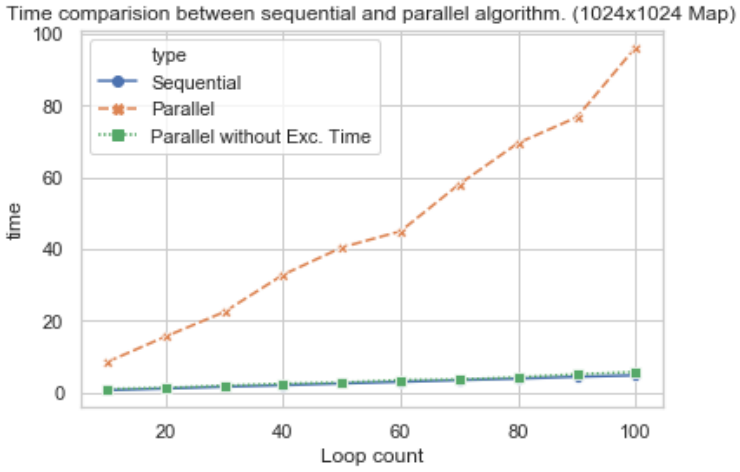
7.1 16x16 Map



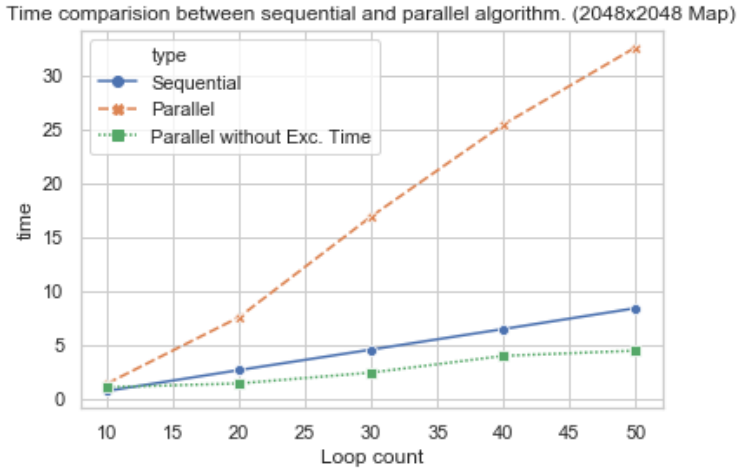
7.2 256x256 Map



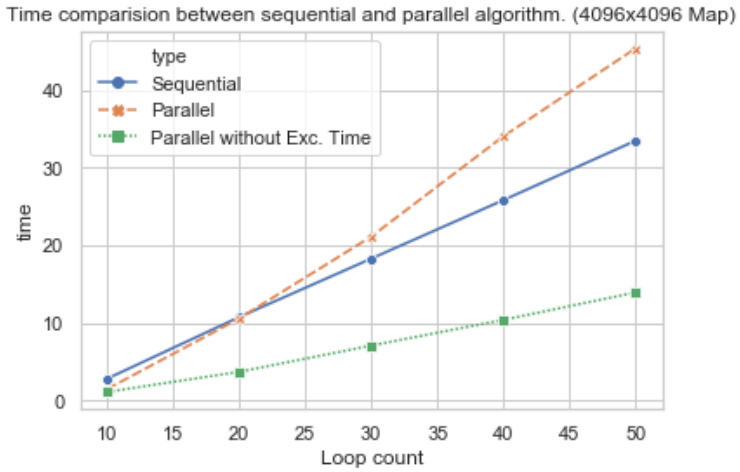
7.3 1024x1024 Map



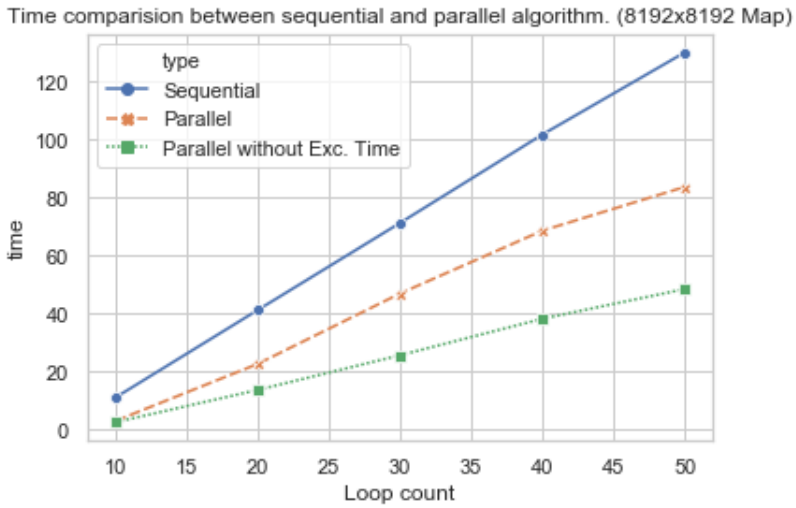
7.4 2048x2048 Map



7.5 4096x4096 Map



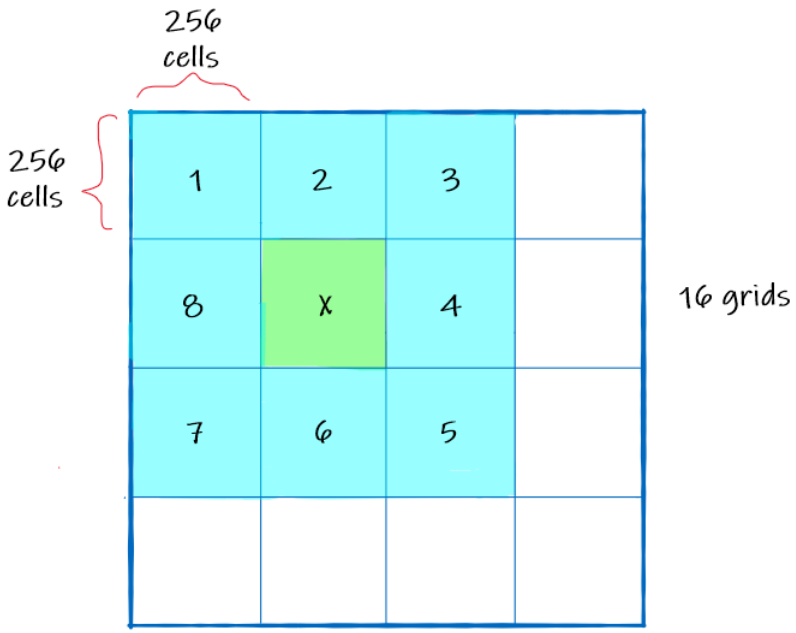
7.6 8192x8192 Map



8 Discussion

As you can see from the graphs, sequential algorithm outperforms the parallel algorithm for smaller maps. Parallel algorithm mostly suffers from high exchange time. Let us explain it with an example.

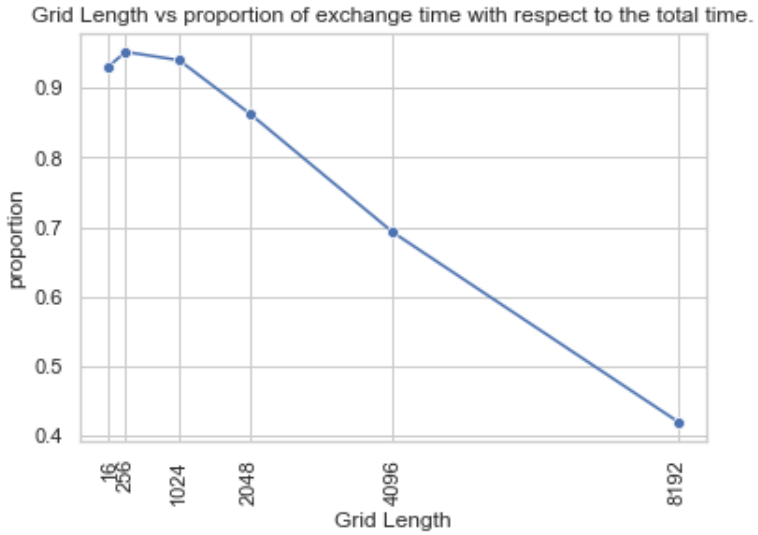
Assume that we have a map of 1024×1024 with processes per row is 4 and processes per column is 4.



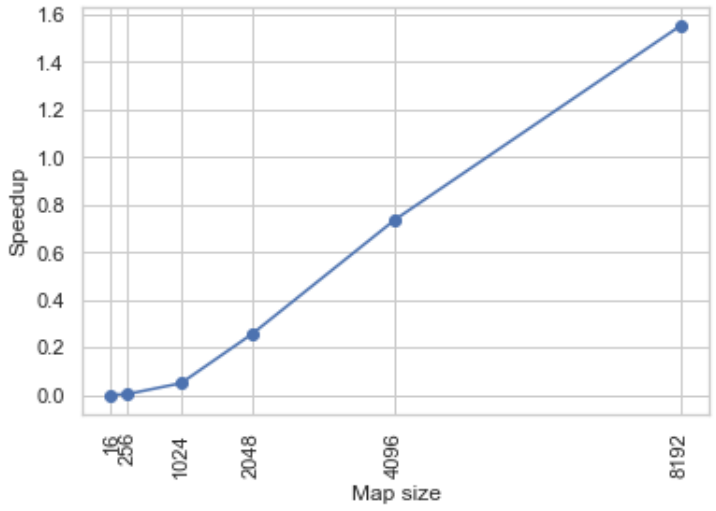
Map consists total number of 16 grids. Every grid (even at corners and at edges) has 8 neighbour grids as shown in the upper figure. Due to our proposed neighbour system and absence of common memory among processes in MPI, each grid have to get content of neighbour grids, which means we are sending total number of $16 \times 8 \times 256 \times 256 \times$

generation count cells in the system. On the other hand in sequential algorithm, we can directly access the contents of neighbour grids, which saves us from high communication time for smaller maps.

To see the effect of exchange information at the end of each generation, we considered looking proportion of time spent on exchange information with respect to total time.



As you can see from the upper figure, when we increase the size of map, proportion of exchange time with respect to the total time decreases. We mostly suffer from high exchange time for smaller maps. Our parallel algorithm outperforms sequential algorithm when map size is increased. This can be easily concluded from the speedup graph.



Also, we need to keep in mind that there are some performance degradation due to overheating of the computing platform while the running parallel algorithm.

References

1. Longfei Ma, Xue Chen, and Zhouxiang Meng. A performance Analysis of the Game of Life based on parallel algorithm. *CoRR*, abs/1209.4408, 2012.

2. Prasad, S., Gupta, A., Rosenberg, A., Sussman, A., Weems, C. (2015). Topics in Parallel and Distributed Computing. Morgan Kaufmann.

3. Rumpf, T. Conways Game of Life accelerated with OpenCL. In Proceedings of the Eleventh International Conference on Membrane Computing (CMC11), p. 459. book-on-demand. de, 2010