

# Conway's Game of Life

## Coursework

### Conway's Game of Life Explanation (Q1)

Game of Life is a cellular automata designed by Conway in 1970. It is a zero-player game and the only input provided is the initial state of the game board. The state of the board is updated at each iteration of the game in a cell-by-cell basis. The decision of the state of each cell in the next iteration depends on the number of cells that are alive in its 8-neighbourhood and on the state of each cell in particular. Therefore:

- 1) If a cell is alive and has exactly two or three neighbours alive it will be alive in the next iteration. Otherwise, it dies.
- 2) If a cell is dead and has exactly three neighbours alive it will be alive in the next iteration. Otherwise, it remains dead.

### Compile Instructions

```
$ ./compile.sh
```

### Code Organisation

The code has been organised in several folders:

- inc: contains the header files (.h or .hpp).
- src: contains the code files (.c or .cpp).
- test: contains all the required headers and code files for the unit tests (i.e. catch and test.cpp, which is the file that contains all the tests).
- doc: contains all the documentation of the project including this report.

- bin: contains the two binary files produced by this project, the app and the tests.

## Run Unit Tests

Execute the following commands in the main directory of the project in order to check the correctness of the code.

```
$ bin/GameOfLifeTest
```

## Run the Program

After the program and the tests have been compiled, execute this command to run the program:

```
$ build/src/GameOfLife -input test/input.txt -output output.txt
$ build/src/GameOfLife -random 100 -output output.txt
```

## Parallelisation

The code has been parallelised using OpenMP. The part of the code that has been parallelised is the method `evolve()` of the `Board` class since each cell is updated based exclusively on the previous state of the cells in its neighbourhood, which does not change while the other cells are being updated.

```
// Code that executes Conway's rules and updates the board
#pragma omp parallel for
for (uint32_t i = 0; i < m_rows; i++) {
    for (uint32_t j = 0; j < m_cols; j++) {
        uint32_t nAlive = aliveNeighbours(i, j);
        if (m_board[i][j].isAlive()) { // If cell is alive
            if (nAlive == 2 || nAlive == 3)
                newBoard.cell(i, j, true);
            else
                newBoard.cell(i, j, false);
        }
        else { // If cell is not alive
            if (nAlive == 3)
                newBoard.cell(i, j, true);
        }
    }
}
```

```

        else
            newBoard.cell(i, j, false);
    }
}
}

```

## Results

Video uploaded to YouTube demonstrating Game of Life:

<https://www.youtube.com/watch?v=949ciYmRyhE>

This video has been generated running this program with the following command:

```
$ ./GameOfLife -random 100 -niter 500 -output output.txt
```

Then, the file output.txt is converted into a video with the following command:

```
$ python extras/txt_to_video.py output.txt video.mp4
```

Speed tests comparing the execution time with and without OpenMP:

**OS :** MAC OS X 10.9.5

**Machine :** Intel Core i7 2.5GHz 16GB 1600MHz DDR3 (4 cores)

**Benchmark :** Unix 'time' command

**Iterations :** 100

**Board sizes tested:** 100x100, 1000x1000, 2000x2000

Each test has been executed three times.

Without OpenMP:

```
GameOfLife -r 100 -n 100 -o output.txt 0.27s user 0.02s system 99% cpu
0.294 total
```

```
GameOfLife -r 100 -n 100 -o output.txt 0.27s user 0.01s system 99% cpu
0.286 total
```

```
GameOfLife -r 100 -n 100 -o output.txt 0.28s user 0.01s system 99% cpu
0.293 total
```

GameOfLife -r 1000 -n 100 -o output.txt 26.30s user 0.37s system 99% cpu  
 26.676 total  
 GameOfLife -r 1000 -n 100 -o output.txt 26.68s user 0.39s system 99% cpu  
 27.066 total  
 GameOfLife -r 1000 -n 100 -o output.txt 26.54s user 0.39s system 99% cpu  
 26.936 total

GameOfLife -r 2000 -n 100 -o output.txt 105.65s user 1.52s system 99% cpu  
 1:47.20 total  
 GameOfLife -r 2000 -n 100 -o output.txt 106.05s user 1.57s system 99% cpu  
 1:47.67 total  
 GameOfLife -r 2000 -n 100 -o output.txt 105.97s user 1.58s system 99% cpu  
 1:47.57 total

#### With OpenMP:

GameOfLife -r 100 -n 100 -o output.txt 0.50s user 0.02s system 375% cpu  
 0.138 total  
 GameOfLife -r 100 -n 100 -o output.txt 0.50s user 0.02s system 356% cpu  
 0.147 total  
 GameOfLife -r 100 -n 100 -o output.txt 0.50s user 0.02s system 382% cpu  
 0.138 total

GameOfLife -r 1000 -n 100 -o output.txt 60.73s user 0.44s system 570% cpu  
 10.729 total  
 GameOfLife -r 1000 -n 100 -o output.txt 60.02s user 0.43s system 564% cpu  
 10.714 total  
 GameOfLife -r 1000 -n 100 -o output.txt 62.75s user 0.44s system 568% cpu  
 11.124 total

GameOfLife -r 2000 -n 100 -o output.txt 282.59s user 1.80s system 607%  
 cpu 46.848 total  
 GameOfLife -r 2000 -n 100 -o output.txt 284.56s user 1.79s system 611% cpu  
 46.810 total  
 GameOfLife -r 2000 -n 100 -o output.txt 286.03s user 1.82s system 611% cpu  
 47.105 total

#### Summary (without OpenMP):

100x100: 0.294s, 0.286s, 0.293s

1000x1000: 26.676s, 27.066s, 26.936s

2000x2000: 107.20s, 107.67s, 107.57s

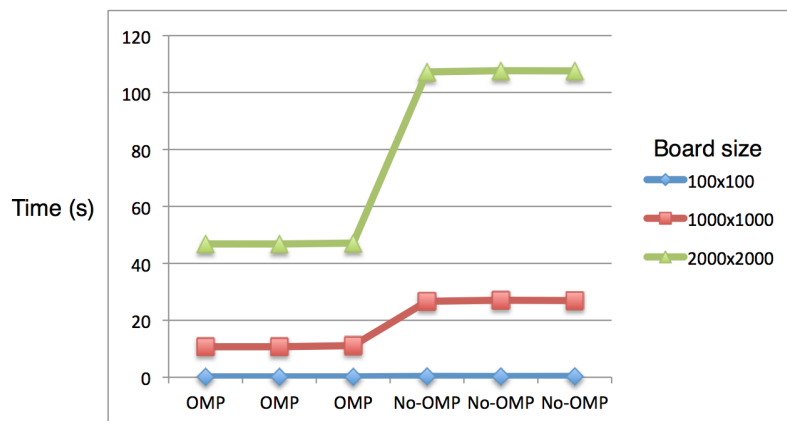
### Summary (with OpenMP):

100x100: 0.138s, 0.147s, 0.138s

1000x1000: 10.729s, 10.714s, 11.124s

2000x2000: 46.848s, 46.810s, 47.105s

	100x100	1000x1000	2000x2000
<b>OMP</b>	0.138	10.729	46.848
<b>OMP</b>	0.147	10.714	46.81
<b>OMP</b>	0.138	11.124	47.105
<b>No-OMP</b>	0.294	26.676	107.2
<b>No-OMP</b>	0.286	27.066	107.67
<b>No-OMP</b>	0.293	26.936	107.57



## Conclusions

The speedup achieved with OpenMP for a boards of 100x100, 1000x1000 and 2000x2000 are 206.38%, 299.99% and 229.07% respectively.