Game of Life Simulation

By
Md Mashiur Rahman Chowdhury
CS 432/632/732 Parallel Computing
January 25, 2018

Problem Specification

The Game of Life, also known simply as Life, is a cellular automaton devised by the British mathematician John Horton Conway in 1970. It is a board game with two-dimensional array of cells. Each cell represent the state of an organism and has eight neighboring cells (left, right, top, bottom, top-left, bottom-right, top-right, bottom-left). It is assumed that, array wraps around in both dimensions which means that the left boundary is adjacent to the right boundary and the top boundary is adjacent to the bottom boundary (often referred to as periodic boundary conditions). Each cell can be in two states : alive or dead. The game starts with an initial state (alive or die) and multiply in the next iteration according the following rules :

1. If a cell is “alive” in the current generation, then depending on its neighbor’s state, in the next generation the cell will either live or die based on the following conditions:
    - Each cell with one or no neighbor dies, as if by loneliness.
    - Each cell with four or more neighbors dies, as if by overpopulation.
    - Each cell with two or three neighbors survives.
2. If a cell is “dead” in the current generation, then if there are exactly three neighbors "alive" then it will change to the "alive" state in the next generation, as if the neighboring cells gave birth to a new organism.

The above rules apply at each iteration (generation) so that the cells evolve, or change state from generation to generation. Also, all cells are affected simultaneously in a generation (i.e., for each cell you need to use the value of the neighbors in the current iteration to compute the values for the next generation).

Program Design

We have used a 2D array of size (N+2) * (N+2) to keep the state of the board. To implement the iteration we need to define another 2D array of same dimension to keep the mirror result. Though N * N array is enough to keep the board state but we have decided to keep the ghost cell for ease of calculation. We have initialized those ghost cells with the corresponding values from opposite boundary. As we have these ghost cells it was easy to calculate alive cells and make it simpler to check the cells along the boundary.

Testing Plan

We have populated the board with random values 1 and 0 to designate alive and dead. As we have taken the dimension and iteration as command line argument, we have added proper sanity check to avoid any kind of unintentional user input error. Both the board and mirror board were allocated memory and freed after the calculation. We have decided to run each generation for 3 times and then we have taken the average time. For memory allocation, deallocation and elapsed time calculation we have used c++ library. We have tested the program with small dimension first. Sometimes board were never converged and sometimes it got converged after a few iteration. In both case we have done proper testing with the small and large dimension.

Test Case :

To compile the program we have used g++ (Ubuntu 5.4.0-6ubuntu1~16.04.5) 5.4.0 20160609 version. Instructions to compile and run the program are written in the source file.

Test Case	Problem Size	Max Generations	Time Taken	Machine Configuration

1	1000*1000	1000	34.574 Seconds	OS=Centos 7 amd64
				Kernel=3.10.0-693.11.6.el7.x86_64
				CPU= 2 x Intel(R) Xeon(TM) CPU 3.20GHz
				Memory = 3.87 GB

2	5000*5000	1000	849.129 Seconds	OS= Centos 7 amd64
				Kernel = 3.10.0-693.11.6.el7.x86_64
				CPU=2 x Intel(R) Xeon(TM) CPU 3.20GHz
				Memory= 3.87 GB

3	5000*5000	5000	41.258 Minutes	OS=Ubuntu 16.04 LTS Kernel=4.4.0-109-generic CPU=Intel® Core™ i5-3340M CPU @ 2.70GHz × 4 Memory 15.5 GB
4	10000*10000	1000	34.25 Minutes	OS=Ubuntu 16.04 LTS Kernel=4.4.0-109-generic CPU=Intel® Core™ i5-3340M CPU @ 2.70GHz × 4 Memory 15.5 GB
5	10000*10000	10000	5.62 Hours	OS=Ubuntu 16.04 LTS Kernel=4.4.0-109-generic CPU=Intel® Core™ i5-3340M CPU @ 2.70GHz × 4 Memory 15.5 GB

Analysis & Conclusion

We have implemented the game of life cellular automaton. We have followed each of the steps to convergence. Those ghost cells were used for the ease of alive or dead cell calculation. We think the design goal of program was achieved for sequential program. From the experiment result we can see that it took some time to execute those programs. Specially the last part it took almost 6 hours to finish the program. Though in some of the machines there were multiple cores available but we have not take those cores into account. We have loaded the data in a single board with contiguous memory and then sequentially executed the generations. If we look back into the implementation now what we feel is that we should have design the algorithm in a way so that we can make use of those multiple cores to utilize the cpu power properly. Though we have used C++ library but we have not this in OOP style. We should have taken this into consideration at the start of design.

Reference

For writing this report we have taken help from the assignment description and wikipedia to understand the game of life methodology properly.