CS4000

C++ Threads and the Game of Life

due Thursday, Feb. 28th, 2019

(50 pts.)

Introduction

In the 1970's, John Conway developed a game with very simple rules that produces amazingly complex patterns — The Game of Life. Conway's original Game of Life is played on a 2-dimensional grid of cells. The game starts with some of the cells being "alive" and the remaining cells being "dead." The game is played in rounds, where each round depends on the values of the cells in the previous round.

In each successive round, the value of a cell is determined by the values of its 8 neighboring cells from the previous round. If four or more of a cell's neighbors are alive, then that cell dies because of overpopulation. If zero or one of the cell's neighbors are alive, then that cell dies of loneliness. If two or three of a cell's neighbors are alive, then that cell survives (if it was alive previously). Finally, a cell is born if exactly 3 of its neighbors are alive. These are the only rules in the Game of Life.

Conway's rules can produce strangely intricate patterns. For example, consider the action of three living cells in a row. This pattern will "blink" forever if left alone.

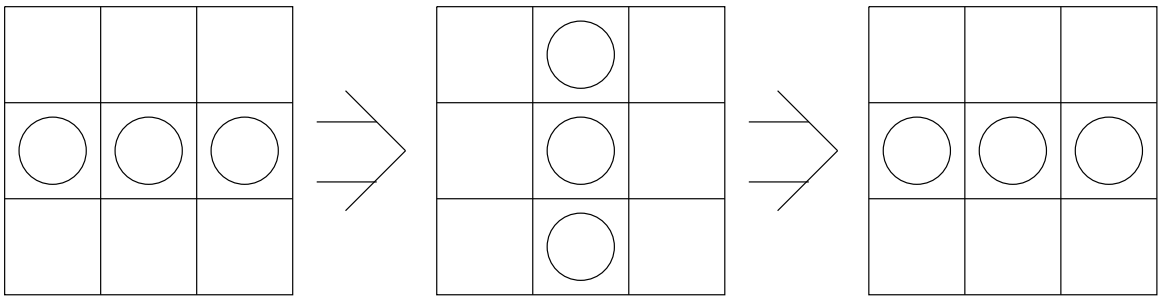

Figure 1: A blinker

Project

In Conway's game of life, the 2 dimensional world is infinite. For this project, we will simulate Conway's Game of Life over a finite 2 dimensional grid. Our grids will be finite and square $(n \times n)$. So, in this case, the 8 neighbors of any grid cell (i, j) will be

1. $(i + 1(\bmod)n, j)$
2. $(i - 1) \bmod n, j)$
3. $(i - 1) \bmod n, j + 1 \bmod n)$

4. $(i-1) \bmod n, j-1 \bmod n)$
5. $(i+1) \bmod n, j+1 \bmod n)$
6. $(i+1) \bmod n, j-1 \bmod n)$
7. $(i, j+1 \bmod n)$
8. $(i, j-1 \bmod n)$

For positive values, the meaning of x mod y is clear. For negative values (e.g., -1), the value that you want to compute is not `a%b`, but `(b-a)%b`. So, for example, `-1 % 10` = `(10-1)% 10` = 9. In general, `(n+i+/-1) mod n` will give you the value you desire.

For your project, you will create a class `GameOfLife` with at least one public member function

```
vector<vector<int> > SimulateLife(vector<vector<int> > &board, int life_cycles)
```

that returns the 2 dimensional grid (1== alive, 0 = dead) after `life_cycles` generations.

The original "board" will have some cells (with a value 2) that are eternal, i.e., they live forever. This slight modification to Conway's game makes this version a bit more interesting.

Sequential Version (20 pts)

Implement the class `GameOfLife` without parallelism. Your code should run reasonably fast. For example, my code on a 100×100 grid, using 1000 iterations (`original.dat`, `1000_iters.dat`) took 1.2 seconds.

Parallel Version Using C++ Threads (30 pts.)

Implement a second version of `GameOfLife` that uses threads in C++ to achieve a faster solution. Your solution should be at least 75% efficient on a 4 core machine. Make your code as clean as possible (consider using the `future` class, etc.). Also, this version may require process synchronization. (See lecture #10.)