Homework 2: Game of Life Parallelized

Zachary Heras

April 11, 2025

1 Problem Statement

The goal of this assignment is to develop a parallel program that simulates John Conway's *Game of Life*, as part of the parallel and concurrent programming course at Rowan University. The program must accept a single input specifying the grid size, N (where the grid is N by N), another input for the maximum number of iterations the game is allowed to run, and an input for the number of threads that the program is allowed to use. The program's execution time will be evaluated based on a set of test cases provided by the course instructor, Dr. Guo. The program is to be written in C using the OpenMP library and executed on Rowan University's cluster computer.

2 Program Design

This program is based off of the serial version of the program from homework 1. With that in mind, we will only review how the program was parallelized, instead of focusing on the design of the game.

To parallelize the program, parallelization is applied to key functions to improve performance using OpenMP. In the init_board function, the initialization of non-ghost cells is parallelized using the `#pragma omp parallel for` directive, distributing the work across multiple threads. In the main function, the game loop runs inside a parallel region with `#pragma omp parallel`, enabling parallel execution of the board update and ghost cell refresh. The update_board function is parallelized by applying `#pragma omp for` to the loops that update the game board and by protecting the updated flag (which tracks if any cell's state has changed) with a `#pragma omp critical` section to prevent race conditions. The update_ghost_cells function is parallelized by dividing the tasks into independent sections with `#pragma omp sections`, where each section handles different parts of the ghost cell update (corners, leftmost, rightmost, topmost, and bottommost rows/columns). Notice that the parallel keyword is dropped from the update_ghost_cells and update_board functions' `#pragma omp` clauses. This is done to ensure that the same threads are being used in each game loop, as the parallel region is defined in the main loop. Synchronization is ensured by using `#pragma omp single` for tasks like incrementing the iteration counter and printing the board, ensuring these operations are performed by only one thread. In an earlier iteration of the code, I mistakenly used `#pragma omp critical` instead of `#pragma omp single`. This caused the game board to print multiple times and caused the iterations to increment multiple times during a single iteration. Overall, the second version leverages OpenMP constructs like parallel loops, sections, and critical sections to speed up the game. This will likely lead to faster execution times on multi-core systems like Rowan University's cluster computer.

To compile the program, use the command `gcc -fopenmp -o hw2 -DSET_SEED hw2.c`, or use the command `gcc -fopenmp -o hw2 -DSET_SEED -DDEBUG_PRINT hw2.c` to print out the game board after each iteration. To run the program, use the command `./hw1 N I T`, where `N` is the game board side size, `I` is the number of iterations, and `T` is the thread count.

To verify that the program is functioning correctly, the program was run with 1 thread and compared to the output of the program with 2 or more threads specified.

3 Test Cases

Once the program was made, the next step was to run tests to examine the execution time for each predetermined test. The tests that will be run are shown in Table 1 below. Each test will be run 3 times and the average execution time will be reported.

Board Size (N x N)	**Max Iterations**
1000 x 1000	1000
5000 x 5000	1000
5000 x 5000	5000
10000 x 10000	1000
10000 x 10000	10000

Table 1: Tests with board size and max iterations.

4 Test System Configuration

Each test was executed on the Rowan cluster computer, utilizing one of the `csm-com-[001-012]` nodes. A Slurm script was written for each test to preserve the test setup information. The scripts were stored in `.sh` files, following the naming pattern `run_hw1_x_y.sh`, where `x` represents the test case and `y` represents the test number.

Thread Count	Execution Time (s)
1	36
2	913
4	4450
8	3663
10	35571
16	35571
20	35571

Table 2: Tests showing thread count and execution time in seconds.

5 Analysis and Conclusions

Below, figure **??** and table 2 describing the resulting execution times. The figure displays the $\log_{10}$ of the execution times for each test case 1-5. From both the table and the figure, we see an exponential growth in execution time as the size of the game board increases. In the case where the maximum number of iterations was capped at 1000 for the 1000 by 1000 board, the experiment seems to have been terminated due to reaching the iteration limit, as its execution time was lower than that of the 5000 by 5000 board, which had a larger iteration limit.

6 References

- https://www.w3schools.com/c/
- https://www.openmp.org/resources/refguides/