Homework 3: Game of Life MPI

Zachary Heras

May 13, 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. A blocking and non-blocking implementation will be developed. The program's execution time will be evaluated and compare based on a set of test cases. The program is to be written in C using the OpenMPI API. All files associated with this project can be found on GitHub.

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 compile and run the blocking program, use the command `mpicc -o hw3_blocking hw3_blocking.c; mpiexec -n P ./hw3_blocking N I`, or use the command `mpicc -o hw3_blocking -DDEBUG_PRINT hw3_blocking.c; mpiexec -n P ./hw3_blocking N I` to print out the game board after each iteration. To compile and run the non-blocking program, use the command `mpicc -o hw3_nonblocking hw3_nonblocking.c; mpiexec -n P ./hw3_nonblocking N I`, or use the command `mpicc -o hw3_nonblocking -DDEBUG_PRINT hw3_nonblocking.c; mpiexec -n P ./hw3_nonblocking N I` to print out the game board after each iteration. `N` is the game board side size, `I` is the number of iterations, and `P` is the number of processes.

To verify that the program was functioning correctly, the game board was traced until completion with a game board size of 5.

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 is run three times and the average value is taken to ensure consistency.

4 Test System Configuration

Each test was executed on my local machine, utilizing a set of my 6 processors. For this test suite, a board size of 5000, and a max iterations of 500 were used. This was done in lieu of setting a random seed; Each experiment ran for the maximum amount of iterations, which ensures that each test was equal in computation.

5 Analysis and Conclusions

Below, figure **??**, figure **??** and table 1 describe the resulting execution times and other metrics.

The speedup (s) is defined as the ratio of the serial execution time (t_{serial}) to the parallel execution time (t_{parallel}):

$$s = \frac{t_{\text{serial}}}{t_{\text{parallel}}}$$

The efficiency (η) is the ratio of Speedup to the number of processors (P):

$$\eta = \frac{s}{p}$$

From table, we see a decrease in execution time as the thread count increases up until 8 threads. After 8 threads the execution time begins to decrease. This was unexpected, and is likely because of a more complicated cache miss issue. This unexpected outcome affected the results of the speedup and efficiency as well. The speed up began to sharply decrease after 8 threads was exceeded, and efficiency got significantly worse after 8 threads as a result. One thread can be thought of as sequential, and as seen it is slower in execution time than all of the larger thread count execution times.

Method	**Thread Count**	**Execution Time (s)**	**Speedup**	**Efficiency**
Serial Program	1			
OpenMP	2			
OpenMP	4			
OpenMP	6			
MPI Blocking	2	131		
MPI Blocking	4	67		
MPI Blocking	6	45		
MPI Non-Blocking	2	131		
MPI Non-Blocking	4			
MPI Non-Blocking	6			

Table 1: Tests showing thread count, execution time in seconds, speedup, and efficiency.

6 References

- https://www.w3schools.com/c/
- https://www-lb.open-mpi.org/doc/v4.0/