CONWAY'S "GAME OF LIFE": HPC EDITION

Homework # 2

By

Marcelo Torres

CS 481/581 High Performance Computing
September 26, 2024

Problem Specification:

The objective of this assignment is to run the program submitted in homework 1 that simulates Conway's "Game of Life" on the Alabama HPC system. The goal of these tests is to become familiar with the HPC system and determine if there is a noticeable performance difference between the HPC and our computers. We are also establishing the bench line for performance on a single thread system for later testing. A short description of the game of life is as follows:

> Conway's "Game of Life" is a cellular automaton played on a two-dimensional grid of cells, which are dead or alive. The state of the cells evolves over discrete time steps according to a set of rules. The rules are applied differently to living and dead cells. For living cells, if a cell has less than two living neighbors, it dies by underpopulation. If a cell has two or three live neighbors, it remains alive. Finally, if a live cell has more than three neighbors, it dies by overpopulation. Dead cells, on the other hand, remain dead unless they have exactly three live neighbors, in which case they become live cells by reproduction.

Program Design:

The program I used in this assignment is a modified version of that used in the first homework assignment. It can be found in this GitHub repository: https://github.com/Player1ce/CS481-HPC.git/ in the folder labeled Homework2Submission. The system uses a 2D array of unsigned 8-bit integers to store the game board. However, the update method has been modified to remove the threadpool used in the first assignment. This has resulted in a decrease in speed that makes direct comparison with the first set of results inaccurate. This change was made after discussing the program with the class instructor. Since the goal of the project is to compare multithreaded and single threaded approaches, establishing a good single threaded baseline is prioritized over using a multithreaded approach. The board is still populated using a randomly seeded uniform distribution, and updates are performed using the same code and rules in a single threaded approach.

Testing Plan

The code was tested on the Alabama HPC system using one core on the asaxg005.asc.edu master node and variable amounts of ram as needed. The code was compiled using the Intel ICPX compiler and CMake. The code was built in CMake release mode with compiler optimizations enabled.

Validation of the simulation was not necessary in this test, since the logic from the first report was reused. However, in some small test cases that I will not list here, the program performed accurately. The results will be compared below to the results in the first report.

Test Cases

As mentioned above, all tests were run using a random, uniform distribution to populate the board. In measurements on boards of 1000x1000 cells, the distribution produced board that were 50% living cells with a tolerance of ±0.05%. All tests ran one core on the Alabama

HPC. Three runs were performed for each test. The 10000x10000 cell grid with 5000 iterations was not tested because the estimated time for each set of iterations was 5.2 hours. All other tests were run and can be seen below. Those results can be seen in table 1 below:

Test Case #	Problem Size	Max. Generations	Time Taken (s) Run 1	Time Taken (s) Run 2	Time Taken (s) Run 3
1	1000x1000	1000	40.752	40.731	41.690
2	1000x1000	5000	203.936	205.541	197.755
3	5000x5000	1000	953.912	919.290	918.250
4	5000x5000	5000	4563.60	4389.459	4020.96
5	10000x10000	1000	3746.980	3575.27	3268.33
6	10000x10000	5000	N/A	N/A	N/A

Table 1: test results on HPC

A comparison with the results of the first report can be seen below.

Test Case #	Problem Size	Max. Generations	Avg. Time Taken Report 1 (s)	Avg. Time Taken Report 2 (s)
1	1000x1000	1000	5.495667	41.06
2	1000x1000	5000	28.553	202.41
3	5000x5000	1000	132.419	930.48
4	5000x5000	5000	661.316	4544.00
5	10000x10000	1000	466.5505	3652.35
6	10000x10000	5000	2359.622	N/A

Table 2: test results from report one compared to results from this report

Analysis and Conclusions

The results from the HPC tests are significantly slower than the results generated in the first report. This difference is because the program has been shifted from a multithreaded to a single-threaded approach. However, some conclusions can still be made. The tests in the first report were all run using 10 threads. After comparing the results, the runtimes from the first report were scaled by an average of 7.14. The adjusted times can be seen below.

Test Case #	Problem Size	Max. Generations	Avg. Time Taken Report 1 Scaled (s)	Avg. Time Taken Report 2 (s)
1	1000x1000	1000	39.23	41.06
2	1000x1000	5000	203.83	202.41
3	5000x5000	1000	945.28	930.48
4	5000x5000	5000	4720.84	4544.00
5	10000x10000	1000	3330.49	3652.35
6	10000x10000	5000	16844.28132	N/A

Table 3: A comparison of the times from both reports with the times from report 1 scaled by 7.14 to account for multi-threading

When the numbers in the second report are adjusted, the comparison becomes much closer. If we divide the average scaling factor by the expected scaling factor of 10, we find that the threading process introduced overhead and caused the program to be only 71.39% as efficient when multithreaded. Unfortunately, I do not have a benchmark to compare this efficiency to, but I will be interested to see how it compares to the efficiency of OMP when

applied to the same program. I am also interested in finding ways to bring this efficiency closer to 100%. Overall, I think this project has been informative, allowing me to learn how to use the HPC and see the weaknesses in some of my code.

Sources

Homework1-Report.docx by Marcelo Torres (URL not available)

https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life