CONWAY'S "GAME OF LIFE"

Homework # 4

By

Marcelo Torres

CS 481/581 High Performance Computing
November 8, 2024

Problem Specification:

The objective of this assignment is to design and implement a simulation of Conway's "Game of Life," a cellular automaton devised by mathematician John Horton Conway. The "Game of Life" is played on a two-dimensional grid of cells, where each cell can be either alive or dead. The state of the cells evolves over discrete time steps according to a set of rules based on the states of neighboring cells. 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.

The goal of this project is to demonstrate the effect of different parallel approaches on Conway's "Game of Life" by writing a distributed memory program based on MPI and running the program with larger and larger process counts and constant iterations and field sizes. The program must be able to calculate the next game board and write results to an output file. The program must also be able to check results against the contents of an output file to verify that changing the number of processes does not affect the accuracy of the output. To that end, the program designed must use a dynamically allocated game board and accept the board size, maximum iteration count, number of threads, output directory, and test file as input.

Program Design:

In this iteration of my design, I used the optimized code from homework3 and converted it to use MPI. The resulting program runs at very similar speeds with perhaps a slight decrease in performance. All code can be found in the `homework4Submission` directory in the GitHub repository. All test results can be found in ` homework4Submission/test-results` in the GitHub repository. Approaching this project, I wanted to use my existing structure and convert it to use MPI. I knew that this would be a fast, effective method to program the project, but I also believe that this will demonstrate a good comparison of the optimization of the two systems. The conversion process was entirely centered around implementing send and receive functionality. To that end, I added a set of 1D buffers for use in communication. I converted the main 2D array to a 1D array, and I utilized the calculateRowGroups function from Homework3 to scatter the array. I used scatterv to account for overhang in the groups. I also included a row above and below the intended working area for each group to allow each section to share information with the other neighboring sections easily. When each group received its data, I converted it back into a 2D array. The implementation of the logic Conway's "Game of Life" is identical to Homework3. After the calculations for the next iteration are complete, I share the top and bottom rows of each working section with the neighboring sections. These rows are shared using MPI_Sendrecv which avoids deadlock when exchanging information. After exchanging the rows, the offset is updated, and the array is checked for changes using MPI_Allreduce. MPI_Allreduce combines all instances of the noUpdate variable using logical and. If the resulting value is true, the computation is stopped. MPI_Allreduce also acts as a barrier,

synchronizing the processes before they continue. Finally, the root process checks if no changes have occurred and outputs the iteration on which the program stopped. After the game loop is complete, the data for each section is sent back to the main process using MPI_Gather. The 2D array is converted to a 1D array for transfer. The overlapping rows are removed before sending this data, so that the combined result has no duplicate data. The received data is combined into a 1D buffer which is again converted to a 2D array, and the final array is then written to a file or tested.

The design of this program is difficult to optimize. An option for optimization could be to use asynchronous send and receive functions; however, the workload for each process is extremely similar and the processes are interdependent. In initial tests with asynchronous send and receive functions, I found very little performance changes, implying that a more complex system would be needed to optimize beyond the current program. I think it could be possible to implement a system in which processes can "get ahead" of other processes, but there is a hard limit to how far ahead a process can get, and there is no guarantee that this will even be advantageous to system performance. Another option could be to size groupings based on population density, but this method might not inherently provide any speed increase as the process of checking each cell would not be bypassed. A last option may be to mark sections to be ignored, but that approach falls outside the scope of this project and would make comparisons between different versions of the program less informative.

Testing Plan

The game logic was tested using small board sizes of 5x5 and a single iteration. The results were printed and compared to the website https://playgameoflife.com/ which provides a free interactive implementation of Conway's "Game of Life." An example is shown below:

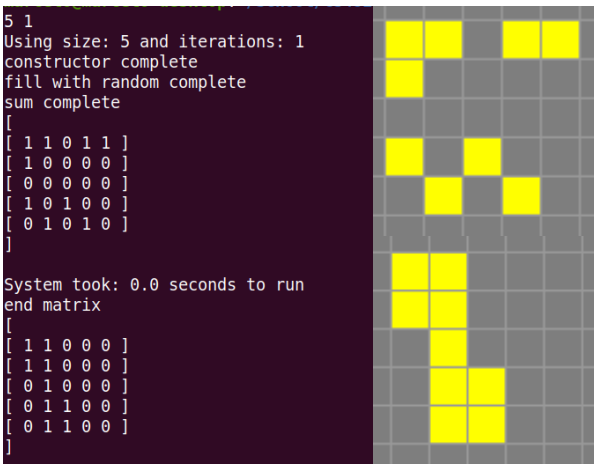

Figure 1: A validation of the game logic

The above figure depicts a printout of the game state before and after an iteration, as well as a graphic representation of the state taken from the website listed above. For this project, this testing was extrapolated to include the new code with different numbers of threads. The results of that testing were successful and can be viewed fully in `homework4Submission/test-results/testvalidityshSCRI.o207108` in the GitHub repository.

Validation for early stopping was tested using a 5x5 identity matrix. These tests were performed for different numbers of threads, and the results were successful. The results of the test can be found in `homework4Submission/test-results/ testearlystopshSCR.o207111`. This file contains the results of the diffs with the single thread version as well as the iterations performed on the board.

Test Cases

All Tests were run using a random, uniform distribution with a constant seed, which yields 50.0036% population, to populate the board. Tests were computed in sets of 3s on a 5000x5000 board with varying numbers of processes. The tests were all run using mpirun and a release version of the code with the –03 compiler flag. The code was compiled using the mpicxx compiler and CMake library provided on the HPC and run using 24 cores on the asax002.asc.edu master node with 5gb of memory. The results for each test were stored in an output file and compared to an output file generated using the already tested OpenMP implementation.

There were no early stops during the entire run. Additionally, there were no mismatches between the code's output and the test file. Results were identical for every run. The full result output can be found here: `homework4Submission/test-results/test-hw4.o207112` and here `homework4Submission/test-results/test-hw4-20only.o207375` in the GitHub repository. The 20 processor runs in the first test failed due to a lack of sufficient resources, so the second file test-hw4-20only reruns those tests successfully. The results for each run are shown below:

Number of Threads	Run 1	Run 2	Run 3	Average Time (s)
1	352.190	351.913	352.190	352.098
2	177.698	177.182	177.295	177.392
4	100.790	100.733	101.920	101.148
8	49.889	50.139	50.808	50.279
10	39.605	40.904	39.550	40.020
16	26.530	26.469	26.458	26.486
20	17.620	17.460	17.610	17.563

Table 1: Runtimes for each test

Analysis

This version of the program significantly outperformed the OpenMP version of the program. Efficiency remained high over all process counts with some variance. I suspect that the high efficiency on the last 20 runs was due to fortunate grid layouts or unusually fast runs, but the average efficiency for all runs was 92%. The efficiency for each number of processes can be observed below:

Number of Threads	Average Time (s)	Speedup	Efficiency
1	352.098	1.000	100%
2	177.392	1.985	99%
4	101.148	3.481	87%

8	50.279	7.003	88%
10	40.020	8.798	88%
16	26.486	13.294	83%
20	17.563	20.047	100%

Table 2: Average efficiency of each set of runs

The speedup and efficiency for the MPI-based program were both significantly better than that of the OpenMP program. A graph of speedup for Homework3 and Homework 4 is below:

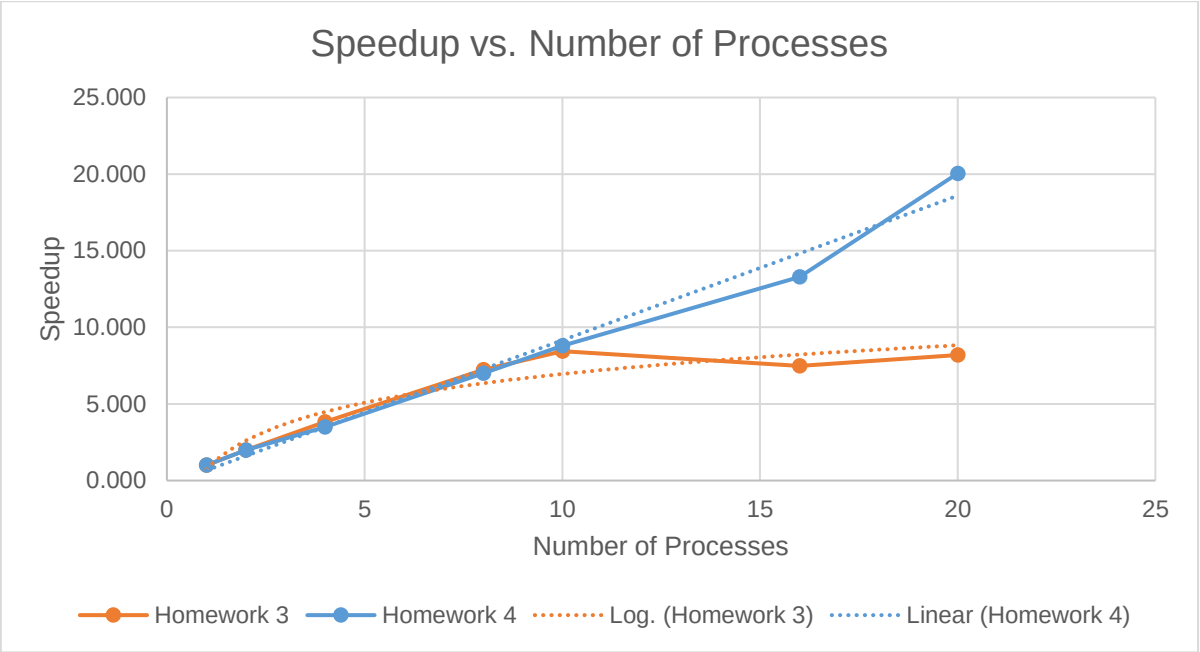

Table 3: Speedup of Homeworks 3 and 4 over number of processes

As can be seen, the speedup for this assignment, Homework 4, follows a linear trendline and is significantly better than the speedup for OpenMP as the number of processes increases.

Efficiency follows a very similar pattern, with Homework 4 performing significantly better than Homework3 at larger process numbers. That chart can be seen below:

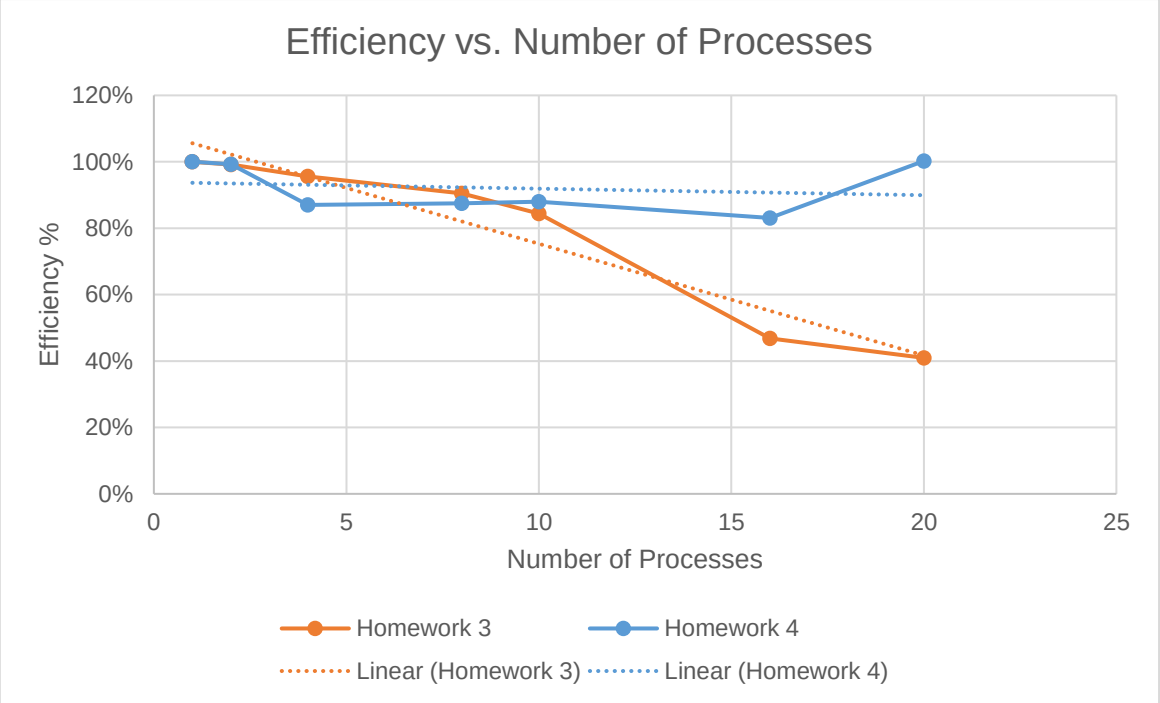

Table 4: Efficiency vs. Number of Processes for Homeworks 3 and 4

The efficiency and speedup charts above depict linear an almost linear relationship between program performance and number of processes. I would be interested in investigating the program's performance on a higher number of processes.

Conclusions

If I were to try to improve this system further, I would explore ways to track cells that do not need to be changed. I would also investigate a non-blocking version of the MPI approach. I believe that the approach of converting the present code to use a new method is valid, and the results illustrate the efficiency and speed of MPI in comparison to OpenMP. Overall, I am satisfied with the work put forward in this project, and I am satisfied with what I have learned.

Sources

GitHub repo: https://github.com/Player1ce/CS481-HPC

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

Homework3-Report.docx by Marcelo Torres (URL not available)

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

https://playgameoflife.com