COP5570 Extra Programming Assignment (7 points total)
OpenMP and MPI implementations of the Game of Life

OBJECTIVES

- Practice parallel programming with OpenMP and MPI

DESCRIPTION

This is an individual assignment. In the *Game of Life*, the world is modeled as a 2-dimensional grid `w[0..w_X-1][0..w_Y-1]` with each entry `w[i][j]` representing a live or dead cell, where `w_X` and `w_Y` specify the size of the world. The world can be represented as a C/C++ array `w[w_X][w_Y]`. Except for cells in the corners or borders, each cell `w[i][j]` has 8 neighbors: `w[i-1][j-1]`, `w[i-1][j]`, `w[i-1][j+1]`, `w[i-1][j]`, `w[i-1][j+1]`, `w[i+1][j-1]`, `w[i+1][j]`, and `w[i+1][j+1]`. Border cells have less number of neighbors. For example, `w[0][0]` only has three neighbors: `w[0][1]`, `w[1][0]`, and `w[1][1]`. Starting from an initial condition, the program will simulate the world population change in each time step. The following are the population change rules in each time step in the game:

1. Any cell with 0 or 1 living neighbor remains or becomes dead (dying out of loneliness).
2. Any cell with 2 living neighbors remains in the same state (live remains live, dead remains dead).
3. Any cell with 3 living neighbors remains or becomes alive.
4. Any cell with 4 or more living neighbors remains or becomes dead (dying out of over-population).

In this assignment, you are given a sequential program for this game. Your task is to parallelize *this particular sequential program* and develop equivalent OpenMP and MPI programs for the game. Your programs should work with any sized world allowed by the OS and any number of threads/processes allowed by the OS.

DUE DATE AND MATERIALS TO BE HANDED IN

Due: **April 22, 2024, 11:59pm** (hard deadline).

- Tar all files including makefile, README file, and submit on canvas. When the submitted files are put in one directory, type 'make' should make the OpenMP and/or MPI executables. The README file should include instructions to demonstrate that your programs are (1) correct and (2) efficient.

GRADING POLICY

Your program only needs to run on linprog. When the following conditions are met, a correct and efficient OpenMP implementation earns 2 points in the final grade, and a correct and efficient MPI implementation earns 5 points in the final grade.

1. Programs with compiler errors will get 0 point. Programs that do not simulate the game of life will get 0 point. A submission whose README file does not include sufficient instruction to demonstrate program correctness and efficiency will get at most 50% of the points even if the programs are completely correct and efficient.

2. Your parallel implementations must have the same calculation, not just the results, as the provided sequential code. A program will get 0 point if this requirement is not met. This means that you must work with the provided code. If a random Game of Life program from the Internet is submitted for grade, it will get 0 point.

3. Your OpenMP and MPI programs must correctly work with any reasonable number of threads/processes and any world size. Failing the correctness test of any case will result in 50% points deduction.

4. For the MPI implementation, each MPI process should only store a fraction of the world (the array size should be roughly equal to $w\_X \times w\_Y/P$ or $w\_X/P \times w\_Y$, where P is the number of processes). An MPI program where an MPI process stores the equivalent of the whole world (for any purpose) will receive 0 point.

5. Besides producing the same output as the sequential code, both your OpenMP and MPI program must achieve **a speedup of more than 2.1** for some configuration (world size, number of threads, etc) on linprog to be considered correct, and **a speedup of more than 4.1** to be considered efficient. The given sequential program should be used as the baseline.

MISCELLANEOUS

OpenMP implementation is simple (a few lines of code). MPI implementation can be quite challenging if you have never programmed distributed memory systems.

All submitted programs will be checked by an automatic software plagiarism detection tool.