COP5570 Programming Assignment 4
Multi-threaded, OpenMP and MPI implementations of the Game of Life

OBJECTIVES

- Practice parallel programming with pthread, OpenMP, and MPI

DESCRIPTION

This assignment can be done in a group of two. 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 multi-threaded, OpenMP and MPI programs for the simulation. 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 14, 2025, 11:59pm**.

- Submit all files related to the assignment including makefile, README file, and your self-grading sheet in one tar file to canvas.

  In a project directory that contains your submitted files, a 'make' command should create the executables (pthread, openMP, and MPI) in linprog. Your README file should describe how to compile and run you program, the known bugs in your program, and how to do your demo. The makefile should (1) automatically generate the executables by issuing a 'make' command under the directory, (2) compile the C/C++ files with '-Wall

-ansi -pedantic' or '-Wall -std=cxx -pedantic', (3) clean the directory by issuing 'make clean', and (4) recompile only the related files when a file is modified.

You should also write a README file that includes the basic README file header given in the website and gives information about (1) how to compile and run your program and (2) how to repeat your demo (you will design a demo for your assignment) showing that your programs are (1) correct and (2) efficient.

REQUIREMENT AND GRADING POLICY

Your program will be tested on linprog: all performance numbers assume the program runs on linprog. You must make sure that your programs meet the following requirements.

1. Your parallel versions of the program 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.
2. Your parallel program (threaded, OpenMP, and MPI) is considered correct when (1) it produces the same output as the given sequential code for any world size and (2) it achieves a speed-up of at least 1.2 over the sequential code for some problem sizes and some numbers of threads/processes. The baseline sequential code should be compiled with -O3 flag. Incorrect program will not receive any points for performance.
3. The multi-threaded (Pthread) program must use the master-worker paradigm and progressively reduce the number of iterations in each task for the worker as the program executes - the first few tasks for the workers must have at least 5 times the number of iterations of the last few tasks.
4. To be considered efficient, your Pthread, OpenMP, and MPI implementations must achieve **a speedup of more than 4.1** on linprog.
5. In the MPI implementation, each MPI process should only store a fraction of the domain (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.

Grading: (the baseline sequential code should be compiled with -O3 flag.)

- Programs with compiler errors will get 0 point. Programs that do not simulate the game of life will get 0 point. Parallel programs that do not have the same calculation as the sequential code get 0 point.
- Proper README, makefile file, etc (10 points)
- Demo that shows that your programs are correct and efficient (5 points)
- Pthread program (30 points + 5 extra points)
  - Correct with a speedup of at least 1.2 (10 points)

  - Efficient (a speedup of at least 4.1) (10 points)
  - Master-worker paradigm with progressive reduction of task size (10 points)
  - Achieve a speedup of 13.1 (extra 5 points)
- OpenMP program (20 points)
  - Correct with a speedup of at least 1.2 (10 points)
  - Efficient (a speedup of at least 4.1) (10 points)
- MPI program (35 points + 5 extra points)
  - Correct with a speedup of at least 1.2 (10 points)
  - Efficient (a speedup of at least 4.1 (10 points)
  - Each process only uses O(World_Size/P) memory (15 points)
  - Achieve a speedup of 13.1 (extra 5 points)
- -15 points for the first unknown bug
- -8 points for the first incomplete implementation
- -3 points for each compiler warning

MISCELLANEOUS

OpenMP implementation is the easiest (a few lines of code); multi-threaded implementation is also not hard; MPI implementation can be quite challenge if you have never written parallel programs before.