COP5570 Project No. 3
Multi-process, multi-thread, and OpenMP implementations of the Game of Life

OBJECTIVES

- Practice Process, thread, OpenMP programming

DESCRIPTION

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, your program will simulate the world population change. The following are the population change rules in each time step in the game:

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

In this project, you will be given a sequential program for this game. Your task is to parallelize **this particular sequential program** and develop equivalent multi-process, multi-thread, and OpenMP implementations for the game. Your program should work with any sized world as long as the OS allows you to have the arrays and any number of threads/processes as long as the OS allows you to create the threads/processes.

DEADLINES AND MATERIALS TO BE HANDED IN

Due date: **July 8, 2015**. The starting of the demo is the due time.

- Project Demo (15 mins)
- Submit the hard-copy of your code, README, and makefile to the TA before the project demo.
- Email the TA your program package that contains all code and related files that allow for repeating your demo.

REQUIREMENT AND GRADING POLICY

Your program only needs to run 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. All programs must have > 1.1 speed-ups over the sequential code in order to receive more than 0 point grade.
3. Your Pthread and OpenMP programs must correctly work with any reasonable number of threads and any world size. Incorrect programs will not get speed-up points.
4. Your multi-process program should use pipe as the communication mechanism among processes and must correctly work with any reasonable number of processes and any world size. No shared memory or communication through files is allowed in this implementation. Incorrect programs will not get speed-up points.
5. Your thread and OpenMP implementations must achieve **a speedup of more than 4.1** on linprog. You are required to use the given sequential program as the base, and to keep the computation complexity in your parallel code as mentioned earlier.
6. Your multi-process program must achieve **a speedup of more than 2.1** on linprog.
7. In the process 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 (10 points for this feature).

The grading for each part of the project is as follows.

- Proper README, makefile file (10)
- OpenMP implementation correct results and >1.1 speed-up (10)
- OpenMP implementation > 4.1 speed-up (10)
- Pthread implementation correct results and >1.1 speed-up (15)
- Pthread implementation > 4.1 speed-up (15)
- Multi-process implementation correct results and > 1.1 speed-up (10)
- Multi-process implementation > 2.1 speed-up (10)
- Partitioned array in multi-process implementation (10)

- Self-guided Demo, showing the correctness at all conditions and the speed-up for one particular case (10)
- -20 points for the first unknown bug
- -10 points for incomplete implementation
- minus all points for programs whose code structures are different from the provided sequential code.
- +3 points for being the first to report a bug in the sample code

MISCELLANEOUS

OpenMP implementation is the easiest (a few lines of coding); multi-thread implementation is also simple; multi-process implementation resembles programming for distributed memory paradigm, and is a challenge (will discuss some tricks in class).

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