Programming Assignment

Notes: You may discuss the problems with your peers but the submitted work must be your own work. Late assignment will be accepted with a penalty of -20 points per day. Please submit your answers through blackboard. This assignment is worth 10% of your total grade.

Game of Life

In this assignment you will implement a parallel version of the Game of Life in OpenMP and Pthreads. The universe of the Game of Life is a cellular automaton, in which cells live ona 2-dimensional world. They are born, live and die over successive generations. The world is defined as a binary-valued array, and each generation evolves according to the following rules:

- Each cell can be one of two possible states: alive or dead.
- Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically and diagonally adjacent.
- At each step in time, the following transitions occur:
  - Any live cell with fewer than two live neighbours dies, as if caused by under-population. - Any live cell with two or three live neighbours lives on to the next generation. - Any live cell with more than three live neighbours dies, as if by overcrowding. - Any dead cell with exactly three live neighbours becomes a live cell, as if by repro duction.

The first generation can be created randomly and the successive generations will follow the above rules simultaneously to every cell. Each generation is a pure function of the preceding one. The rules continue to be applied repeatedly to create further generations. Here is the pseudocode for the Game of Life:

```
for t in 0:T-1
    forall (i,j) in 1:N x 1:N
        nNeigh = number of live neighbors of World(t)[i,j]
        World(t+1)[i,j] = DEAD

        if World(t)[i,j] AND (nNeigh == 2 or nNeigh ==3) then
          World(t+1)[i,j] = LIVE
        else

          World(t+1)[i,j] = (nNeigh == 3)
        end if
    end forall
end for
```

Serial Code

We are providing you with a working serial program that implements the Game of Life. The provided code creates a random world, distributing cells uniformly according to a specified probability. It then runs the game of life for a given number of generations, sending each generation to the gnuplot plotting program.

```
The program may be run from the command prompt as follows:
./life [-n <int>] [-i <int>] [-s <int>] [-p <float>] [-t <int>] [-d] [-step]
[-g <int>]

With the arguments interpreted as follows:
-n <integer> Number of mesh points in one dimension
-i <integer> Number of generations (or iterations)
-s <integer> Random seed
-p <float> Distribution probability
-t <integer> Number of threads
-d disables display
-g <integer> selects a game type
-step pauses every iteration (for debugging)
```

Environment

- To run the program, you will need the **gnuplot plotting program** to be installed on your computer. You can download the software from http://www.gnuplot.info .
- The Makefile that we have provided includes an “arch” file that defines appropriate command line flags for the compiler.

```
To compile, type
      make
To clean the objects and executables
      make clean
Example run:

        ./life -n 500 -i 200 -p 0.2
```

- Since this program requires an interactive interface, we won’t be able to use the campus clusters because they currently allow only batch jobs. Please develop and test your implementations on your local machines or on the computer labs on campus.

Task Parallelism

In this part of the assignment, you will employ multithreading to handle graphical display. You will modify the life simulator to run with two threads. One thread (the plotter thread) should handle the display (via gnu plot) while the other thread should perform the cell updates. The two activities may take place concurrently. To synchronise the execution of the threads, mutexes or barriers can be used. Data should be shared between the threads via shared-memory and the mesh plot must appear for each iteration.

Data Parallelism

In this part of the assignment, you will introduce additional threads to share the computa tional workload in the cell update. To implement data parallelism, use a one-dimensional decomposition where the computational domain is to be divided (approximately) equally among all workers involved in the computation. You must handle the case where the num- ber of threads does not divide mesh size. Lastly, have the master thread initialize the game board. Although this approach isn't scalable because of the first touch policy, the impact on this assignment won't be noticed. Having the master initialize the game board ensures reproducible results and is extremely helpful for debugging.

You can introduce data or task parallelism in any order you like but the final implementation should use a single thread for plotting and the remaining threads to perform cell updates concurrently with the plotter thread. Starting with data parallel implementation might beeasier.

Testing

- You'll notice that the serial program we provided allows you to input a random seed via the -s parameter. If you don't specify this parameter, you obtain a seed based on the time of day. The program outputs this seed so that you will be able to reproduce a given run. In order to establish correctness, run your program on differing numbers of threads, including a single thread.
- Another approach is to run with a problem that has a known solution. The literatureis full of problems with known solutions. The simplest problems to test have the static patterns that do not change. We have provided one, which is called block still. You can find some examples on the Wikipedia page for Conways Game of Life Others include "blinkers" and gliders. A command line option -g can be used to select a game number. You can add the glider (spaceship) game if you like.
- Another command line option -step has been added to allow you to "single step" through the game, one iteration at a time. This is handy for watching moving patterns.
- Finally, another simple way of checking your results against the single thread implementation is to output the contents of all game board locations in a systematic order (say,row major order), printing a single digit (1 or 0) for each position. You can then usethe diff program to compare results. Of course, this assumes that your single processor implementation is correct!

Submission

- Document your work in a well-written report which discusses your findings.

- Your report should present a clear evaluation of the design of your code, including bottlenecks of the implementation, and describe any special coding or tuned parameters.
- Although this assignment will not address performance, we have provided a timer to allow you to measure the running time of your code. Observe the running time with and without the mesh plotter is on. You will see how much time is spent doing I/O.
- Submit both the report and source code electronically through blackboard.
- You only need to submit the final implementations for OpenMP and Pthreads if your task+data parallelism works properly.
- Please create a parent folder named after your username(s). Your parent parent folder should include a report in pdf and two subdirectories: one for OpenMP and the other for Pthreads. Include all the necessary files to compile your code. Be sure to delete all object and executable files before creating a zip file.
- GOOD LUCK.

Grading

Report (30 points), Task (15pts) and Data Parallel (20 pts) OpenMP Implementation, Task (15 pts) and Data Parallel (20 pts) Pthreads Implementation (20 points). You may lose points both for correctness (e.g. race condition) and performance bugs (e.g. unnecessary synch. point) in your implementations.

References

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

Prof. Scott Baden https://cseweb.ucsd.edu/~baden/