ECE 565 - Fall 2019

Assignment #5

Parallel Programming with Pthreads

General Instructions

1. **This is assignment is to be done in groups of 2. Each group must work independently.**
2. This assignment consists of 2 sub-problems. In each sub-problem, you will parallelize existing code using OpenMP and conduct performance experiments that will be written up into a report.
3. **Performance experiments:** Your performance experiments *must* be done using an 8-core VM (running Ubuntu Linux) that you will have access to for assignment #4 and #5. You should run your performance experiments.
4. **Program development:** Your code development, testing, debugging, etc. may be performed either on your own machine, the Intel server that you were given access to previously (kraken.egr.duke.edu or leviathan.egr.duke.edu), or the 8-core Ubuntu VM.
5. Note, as you work through these sub-problems, that **bugs in parallel code are often timing dependent,** i.e. incorrect parallel code may (quite often) result in correct execution due to the absence of certain timing conditions in which the bugs can manifest. Therefore, manual code correctness analysis is very important, and **your code will be graded based on code correctness rather than on output correctness from single test runs**.
6. Please follow the spec carefully and turn in everything that is asked. **Be sure to start early and make steady progress.**
7. Parallel programming requires significant design effort although the amount of code you will write will not be extremely large. You should not underestimate the effort and time to get to the right solution. The best way to do well in this project is to start early.
8. As you work through the two sub-problems of this assignment you will create a writeup (named **report.pdf**). You will submit this report as well as modified code as described in the submission instructions. **Be sure to submit all source code files and report.pdf inside of a hw4.zip file exactly as described or points will be deducted.**

Shared Memory Parallel Programming with Threads

For this part of the project, you will:

1. Create (in C or C++) a sequential program to solve a particular problem.

2. Create a parallel version of the program using pthreads.

3. Write up results as described below for this part into an overall report (**must be named report.pdf**), and submit your code in your single submission zip (**must be named hw5.zip**) file under a directory called rainfall/. Extracting your hw5.zip file should extract report.pdf and the rainfall/ code directory into the current directory. Follow these naming conventions.

Please refer to the course notes and examples, as well as the pthreads user documentation or man pages for help on compiling with pthreads as well as creating, managing, and synchronizing threads that have possibly conflicting reads and writes to shared memory.

Rainfall Simulation

The following is a description of the rainfall simulation problem that your program will solve. In this problem, we have a 2-dimensional landscape, which is essentially an NxN grid of points. Each point has an integer elevation associated with it that your program will read from an input file. In the rainfall simulation, drops of rain will fall onto the points of the 2D landscape. As those rain drops fall, the water will move in two ways: (1) rain drops will absorb into the ground at a point at a specified rate and (2) rain drops will trickle from a point in the landscape to their neighboring point(s) (north/south/east/west) that have the lowest elevation. Your program will perform time step simulation, simulating what happens across the 2D landscape in each time step as rain drops fall onto the points, trickle to lower elevation points, and get absorbed into the ground.

Here are some more specific characteristics of the problem. Within a timestep of the rainfall simulation program, there are 3 things that can happen in the following sequence: (1) a drop of rain may fall onto a point in the landscape, (2) if there exists any rain drops on a point at a given timestep, some amount will be absorbed into the ground, and (3) if there remains any rain drops on a point at a given timestep, some amount will trickle away to the neighboring point(s) with the lowest elevation.

- Your program will take 5 command line arguments:
  - `./rainfall <P> <M> <A> <N> <elevation_file>`
  - P = # of parallel threads to use.
  - M = # of simulation time steps during which a rain drop will fall on each landscape point. In other words, 1 rain drop falls on each point during the first M steps of the simulation.
  - A = absorption rate (specified as a floating point number). The amount of raindrops that are absorbed into the ground at a point during a timestep.

    - N = dimension of the landscape (NxN)
    - elevation_file = name of input file that specifies the elevation of each point.
- Rules for flow of rain drops to lower elevations:
    - 1 full drop of rain at each timestep will flow from a point to its north/south/east/west neighbor in the 2D landscape that has the lowest elevation.
    - If two or more of the 4 neighbors have the same lowest elevation, then the drop will be divided evenly, and a fractional drop will flow to each of the neighbors involved in the tie.
    - If a point has an equal or lesser elevation than any of its 4 neighbors, then no water will trickle off of that point. It will absorb into the ground at the specified rate.
- Sample input for a 4x4 landscape:

```
4 8 7 3
5 4 3 2
7 6 9 2
1 2 3 8
```

- Your program should output two things at the end of the rainfall simulation: (1) the number of timesteps it took for all rain drops to drain into the ground and (2) the number of rain drops absorbed by each point in the landscape as a whitespace-separated 2D grid of output. **It should print to stdout, and match this output format (with the exception of the numbers which will depend on the input).** Sample output for the above sample input when run with the following command:

```
Rainfall simulation completed in 170 time steps
Runtime = 0.037 seconds

The following grid shows the number of raindrops absorbed at each
point:

    17.5     2.5     2.5       3
     2.5    3.75    5.75    42.5
     2.5     2.5     2.5      25
   37.25    5.25     2.5     2.5
```

And some more clarification about the rainfall simulation:
In order to ensure that the rainfall simulation produces a deterministic result, no matter what order the landscape points are traversed and processed, you will need to use the following execution flow:

- Traverse over all landscape points
    - 1) Receive a new raindrop (if it is still raining) for each point.
    - 2) If there are raindrops on a point, absorb water into the point
    - 3a) Calculate the number of raindrops that will next trickle to the lowest neighbor(s)
- Make a second traversal over all landscape points
    - 3b) For each point, use the calculated number of raindrops that will trickle to the lowest neighbor(s) to update the number of raindrops at each lowest neighbor, if applicable.

For this problem, you should:

1. Develop a sequential version of the code to solve this rainfall simulation. Your sequential code can ignore the "P" command line argument. Include this code in the rainfall/ directory of your submission zip file.
2. Develop a parallel version of the code using pthreads. Include this code in the rainfall/ directory of your submission zip file.
3. Include a single Makefile that compiles the sequential and parallel versions of the rainfall code into executables named **rainfall_seq** and **rainfall_pt**, respectively with the 'make' command.
4. In your project report, describe your parallelization strategy, including:
    - Describe your sequential algorithm and data structures for the rainfall simulation.
    - Discuss what parts of your simulation code you parallelized.
    - Why did you choose those parts to parallelize in comparison to other parallelization strategies you considered?
    - What types of synchronization across threads did you use?
5. In your project report, measure and present the performance of your parallel version of the code when run with 1, 2, 4, and 8 threads, compared to the sequential code. Discuss your results. Were you able to obtain speedups? In what ways did they match or not match what you expected. Describe possible reasons for why your results did or did not match expectations.

Your writeup for questions 4 and 5 should be clear and thorough. Try to impress here.

You need to Submit

You will submit this project as a single zip file named **hw5.zip** to your Sakai drop box. You are responsible to submit the following contents in the zip file:

1) The report document (named **report.pdf**) that addresses all parts described above.
2) All your source code in a rainfall/ subdirectory as described above.