Prog. HPC: Homework 3

CS 4080/7080

February 24, 2018

DUE:

- Code : Tuesday, 2018 March 6 @ 11:00 PM
- Report : Sunday, 2018 March 11 @ 11:00 PM

Topic: Multi-threaded programming

The goal of this homework is to:

- practice multi-threaded programming;
- understand the use threads and the trade offs from parallelization strategies;
- refine your skills writing an analysis report on your implemented solution.

You will develop a parallel, multi-threaded data processing algorithm for the problem detailed below (Homework 2). You will NOT use any distributed processing API/framework (e.g.,OpenMPI) for this homework. You MUST manage the parallelism using threads, locks, mutexes, and other parts of a threading library / framework as you see fit. For the problem, you will design your program to run a set of timing experiments with increasing parallelization and data set sizes. **You are not to use OpenMP for this project.**

Problem: ArgMax Filtering and Distance Matrices

For this program, you will complete the following processing steps:

1. Load and transform the data as you did in HW 0 and HW2.
2. Partition the data into sets of X/Y based on $argmax(vector)$.
3. For each partition with less than 5001 points, compute the Haversine distance matrix.
4. For each distance matrix, compute the average distance (normalized sum of matrix elements).

Distance Matrix

Given a list of positions or values, $p(i) \in 0 \dots N$, the distance matrix, M, is defined as:

$$M(j,k) = d(p(j), p(k)); \tag{1}$$

where $d(.)$ is some distance measure. We will use Haversine distance
https://en.wikipedia.org/wiki/Haversine_formula
for this homework. You should convert the distance to meters.

You should develop an algorithm that can compute the **argmax** of each vector in your parsed data structure. Then, all X/Y positions with $argmax(v) = 0$ should be added to a collection C_0, and $argmax(v) = 1$ to C_1, and so on. Each collection C_i should be processed to compute a haversine distance matrix in meters. Your program should provide timing statistics on data loading as well as converting the in-memory data structure into the 21 distance matrices and also the computation of the average distance of each matrix.

Multi-threaded Parallelization of the Solution

You are to determine an approach to parallelization at the data set level. Your timing experiments will first be conducted using a single thread. The subsequent timing experiments will be conducted using 2, 4, 6, and 8 threads, respectively. You should capture and analyze the timing trends as the database size grows, as well as how the parallelization increases.

Your program will take two command line arguments:

1. P : the integer level of parallelism, e.g., number of processes used to process the dataset.
2. F : the input file.

The goal is to accelerate the compute of the distance matrices from the whole dataset.

Your program must be structured as the following algorithm for generating timing results:

Load data, D, from file into Memory.
Segment D from Memory into collections, C_i, in Shared Memory.
# In Parallel, up to P:
for Each $C_i \in \{0 \dots 20\}$ **do**
    Compute M_i
    Store the computed M_i into a data structure
    Measure average(M)
end for
Report Timing Statistics.

Analysis Report

REMEMBER: The goal is to accelerate the computation of the distance matrices and average distance matrix after the data has been segmented. All your timing discussions should be in the context of the time to perform the work, using parallization level P. You should also discuss the time to load and sort the dataset from file into a data structure that facilitates parallel accelerated processing, but this is just incidental information for perspective.

Algorithm Describe your algorithm(s) to perform the computation tasks defined above. You should provide a clear discussion of your algorithm in the context of how the parallel threads are accessing the dataset. Additionally, you must provide a clear, yet concise, discussion of when and how and why resources are protected.

Timing Analysis For timing and analysis will use the following sets for P and F:

- P = 1, 2, 4, 6, 8;
- F = /data/scottgs/hpc_{N}m.csv, with $N \in [1, 9]$.

I expect the analysis to include various graphical plots. The discussion of the timing should address the trends of the timing using various values of P and N, where N is millions of data rows. Offer some interesting insights and explain which portions of your code/algorithm(s) results in the various trend characteristics. How does Ahmdahl's Law or Gustafson's Law apply to your algorithm or code performance?

Lessons Learned Conclude with a discussion of the key things you have learned during this exercise in regards to multi-process programming, parallel algorithm development, conducting analysis of code, etc.

Additional Implementation Constraints

- You must specify the build/runtime system in the submitted README file, along with usage instructions.
- If your code is excessively slow, decrease the number of timing iterations until you can get your code running effectively.
- You must implement the solutions using C or $C{+}{+}$.

- Your solution CANNOT use a multidimensional data structure library other than the STL to manage the data, you must implement your own data storage management and search aglorithms. If you would like to do **extra work** and compare your solution to a library, that is fine.
- You CAN USE ANY LIBRARIES, such as boost mutexes or locks, or any other APIs not expressly forbidden.

Data Files

The following data files are available for you to experiment with. At a very minimum, you must process the one million record file to perform your analysis experiments. Analysis along data growth trends is encouraged.

```
wc -l ‘pwd‘/hpc_?m.csv
    1000000 /data/scottgs/hpc_1m.csv
    2000000 /data/scottgs/hpc_2m.csv
    3000000 /data/scottgs/hpc_3m.csv
    4000000 /data/scottgs/hpc_4m.csv
    5000000 /data/scottgs/hpc_5m.csv
    6000000 /data/scottgs/hpc_6m.csv
    7000000 /data/scottgs/hpc_7m.csv
    8000000 /data/scottgs/hpc_8m.csv
    9000000 /data/scottgs/hpc_9m.csv
   45000000 total
```

Deliverables & Assignment Submission

You will submit two things.

1. The updated code pushed up to OSGIT using git. Be sure to first run `make clean` on the project folder. Be sure to add all needed files to the submission.
2. A *PDF* report which organized according to the topics identified in the Analysis Report section above. The report should not exceed seven pages and should identify your *Name* and *Pawprint* at the top each page. The naming convention should be: *pawprint_hw3.pdf* Where the *pawprint* is your actual pawprint id. This file will be submitted into Canvas.