CST8219 – C++ Programming
Lab 9

Introduction:
The goal of this lab is to practice parallel programming using OpenMP. This will let you compute your workload much faster than on a single processor.

Reference:
Week 9 Powerpoint materials on Brightspace. There are many reference websites at the end of the powerpoint slides.

Steps:

1. Take your work from Lab 8 and convert it to a git repository. That means go to the directory where your files are saved and type: git init.

2. Then add your source files to the git tracking list:
   a. git add *.java
   b. git add *.cpp
   c. git add *.h
   d. git add CMakeLists.txt
   e. git commit -m “initial commit”

3. Now create a branch for your week 9 work: git branch Week9. Add the line to CMakeLists.txt to enable support for OpenMP:
   set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} /openmp")

4. Open your .cpp file that has the two functions for calculating the standard deviation and mean of the numbers. Modify the code so that it uses a parallel for loop to iterate over the array. Make sure that your for loop has a guard to wait for all the threads to finish before returning the result to java.

5. In Java, add code to measure the time before calling the function:

```
long timeBefore = System.currentTimeMillis();
        calculateMean();
        calculateSTDdev();
long timeAfter = System.currentTimeMillis();
```

   Then add a println statement to your program to indicate how long it took to compute the mean and standard deviation. If your computer has an intel i7 processor, then you probably have hyperthreading turned on. Look at the notes on how to use only half of the processors so that hyperthreading doesn’t slow down your computation. You should see a difference in the compute time when you start working with a few million samples in the computation when compared to using a single processor. To see the difference,

set the number of threads to 1 and then run your program to see how long it takes to run.

Submission: Create a zip file containing everything in your week8 directory and submit it on Brightspace. Make sure it includes your .git directory, the Java class, the .cpp and .h, and your CMakeLists.txt file

Marks: (total of 5)

- The Java Gui has a field for the time it took to do the computation +1
- Your C++ file has a #pragma omp parallel with { } to show where multi-threading is enabled +1
- The results are computed correctly, and returns only when all threads have finished +1
- Your code checks if the number of processors available is more than 7, in which case hyperthreading is probably enabled. +1
- The standard deviation and mean are still shown correctly in the gui +1