EE155 / CS122: Parallel Computing
Lab 1: Histogram Generation with C++ threads

In this homework, your goal is to write a program that processes input data and builds a histogram. You want to take advantage of multiple threads to run as fast as possible.

We noted in class that if the threads spend most of their time in a critical section, then the handoffs involved will be very slow. Thus, you want to somehow ensure that most of the work in the threads does not need to be in a critical section. How you do this is up to you.

The program provided to you accepts the size of the input data set as an argument. It first creates a randomly-initialized data set and computes the histogram with a known-correct single-threaded implementation (which is already coded). It then re-processes the same input data using a multi-threaded implementation (which you must write), and checks the answer.

Use an input-dataset size of 1 billion pieces of data. Collect multi-threaded data with 1, 2, 4, 8, 16, 32 and 64 threads (which the program will do automatically).

Write a short report explaining how you shared the work between threads. Include your results, and answer the questions below. A correctly-running program and a two-sentence description of how you did the sharing are worth 50 points; the questions below are worth 50. Note that we do not grade you on how fast your code runs – except that if it runs so slowly as to impact our grading software (e.g., about a half hour or more) then we'll take off a few points.

Logistics:

- The lab assignment is due at midnight on the date that the class calendar specifies. You must work on this assignment on your own. Submit your project via **provide**. You should submit a copy of your report as a PDF, as well as your version of histogram_user.cxx.
- Copy the files histogram.cxx, histogram_user.cxx, ee155_utils.cxx and ee155_utils.hxx into your own work directory. You will eventually turn in only histogram_user.cxx; the other files are part of the infrastructure.
- Edit the function compute_multithread() within the file histogram_user.cxx to complete the functionality of histogram generation. You may add other new functions to the file as needed. Do not change the source code elsewhere.
- Compile with the line **c++ -pthread -std=c++11 -O2 histogram.cxx ee155_utils.cxx**. The **-O2** will make your program run much faster, and you should use that for your final results collection. However, for initial debug you may want to skip it, and add **-g** (which creates information for debugging).
- Run with **./a.out 1000000000** (to, e.g., get the desired 1 billion data entries).

Resources:

- Pages 66-70 of "An Introduction to Parallel Programming" cover the histogram problem.
- Use any Linux system in 074 for your development and benchmarking. If you are going to collect benchmark data, then reboot the machine to be sure that nobody else is using it from a previous session.
- It is possible to log into the systems remotely. They are named lab120a – lab120v. However, these machines are not exposed to the general internet. Instead, you must first **ssh** into a publically-visible machine (e.g., linux.eecs.tufts.edu), and then **ssh** into a lab

machine. To use X-windows tools on the lab machine, the second **ssh** should be "**ssh -X"**. Also, note that you will not be able to reset the machine remotely. Finally, you should use **top** or something similar to check if another person is actively using the machine – obviously that could greatly affect your runtimes.

- You can try to debug on your own laptop if you install a C++ compiler. However, note that ee155_utils.cxx uses Linux-specific functionality in the function assign_to_core(), so your code may not compile under Windows. You can just comment it out for this lab if you like; however, we will use it in the next lab.

***Reference results* (to be used for all questions below except as noted for question #1):**

# of threads	1	2	4	8	16	32	64	128
time	520ms	270ms	270ms	220ms	240m	250ms	500ms	1500ms

Questions:

1. Compare your results to the reference results. If they differ substantially, then why do you think that is? (*Ungraded*).
2. Given the input-data size of 10^9 values, how much space would that data take up? Which memory level does it fit in? (*10 points*)
3. You will note that using two threads makes the computation take roughly half as long as just one thread, and that 2-32 threads all take roughly the same amount of time. Let's try to explain this (*20 points*).
    a) First explain why the main-memory bandwidth number is the correct memory bandwidth to use (as opposed to, e.g., the L3-cache bandwidth).
    b) For simplicity, assume that the main-memory bandwidth is 32GB/sec. How many integers/sec can main memory supply to the CPU?
    c) How many integers/second would each core have to be capable of consuming to make our data above reasonable?
    d) Assume that the processor runs at 4GHz. What assumption would you have to make about how many cycles, on average, the inner loop of the calculation takes? Is this "mostly" reasonable (given the picture on archreview_2 / slide #3)?
4. You will note that starting with 64 threads, throwing more threads at the problem makes runtime substantially *longer* rather than shorter for the reference data. We noticed this type of behavior in class (archreview1_caches slides #15-22 and lec3_cpp_threads slides #20-25). We've not told you exactly how the reference implementation is coded – but…
    a) is the first (cache-striding) explanation compatible with the reference data? Why or why not? (*8 points*)
    b) the second (spin-lock) explanation? Why or why not? (*8 points*)
    c) can you propose another hypothesis? (*4 points*)