**CSE-443/543: High Performance Computing**

**Lab #04**

Max Points: 33

**Objective**: The objective of this exercise is to:

* Build experience with compiling code at the command line and constructing batch shell scripts
* Gain familiarity with the Linux perf hardware-assisted profiler.
* Learn about the concepts of *standard output* and *standard error* in Linux.
* Use information from perf to identify the cause of the performance difference between a “good” and a “bad” implementation of a simple algorithm.

Fill in answers to all of the questions. For almost all the questions you can simply copy-paste appropriate text from the shell/output window into this document. You may discuss the questions with your instructor and the TA.

## Background

For this lab you are given two very similar programs, Lab03-bad.cpp and Lab03-good.cpp. They each perform multiplication operations with a pair of matrices but the bad version runs significantly slower than the good version. The bad version contains a “rookie” mistake that is easy to make (and easy to fix).

You will be using the perf hardware-assisted profiler to identify the indicators of the performance problem with the bad code. You will need to review the two versions of the code to determine the differences between them then, as part of your analysis, explain how the output of perf is consistent with the coding differences.

Writing your shell script and running the code should take only a small part of the time in this lab. More time should be spent reviewing the code, analyzing the output of perf, and researching the descriptions of the relevant performance counters.

One other note: In Linux there are two ways to stream information to the terminal screen. “Normal” output is written to the standard output (stdout) stream while error messages and timing data are written to the standard error (stderr) stream. In this lab you will need to redirect the stderr of the perf command to a file. If you are not familiar with stdout and stderr, [this article](https://www.howtogeek.com/435903/what-are-stdin-stdout-and-stderr-on-linux/) provides an excellent overview of them.

## Experiments

1. Download the files Lab04-bad.cpp and Lab04-good.cpp from Canvas.
2. Create a shell script to run your job. It should do the following:
   1. Invoke /bin/bash as a login shell
   2. Get a single value for the CPU model name and total memory from the appropriate files in the /proc filesystem as you’ve done in previous labs.
   3. Compile the two source code files, creating executables called Lab04-bad and Lab04-good respectively. g++ should use these options (note the change from previous labs): -g –O3 –std=c++14 –Wall
   4. Use the perf stat –d –d -d command to run Lab04-bad, redirecting the standard error to a file called Lab04-bad.perfstat.
   5. Use the perf stat –d –d -d command to run Lab04-good, redirecting the standard error to a file called Lab04-good.perfstat.

# Apparatus (experimental platform)

The apparatus used for the experiments documented in this report were conducted on the following platform (fill in the Details column using information determined in your batch job):

|  |  |
| --- | --- |
| Component | Details |
| CPU Model | Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz |
| Main Memory (RAM) size | 8071544 kB |

# Observations – perf stat

The perf stat command gives you information on various counters that the hardware keeps track of while the program runs. One is the elapsed time the program took to run. Record the elapsed time for each program in the appropriate table in your Google spreadsheet using data from the \*.perfstat files you created.

Running perf stat –d –d –d gives you information from many different hardware counters. Record the values of the counters requested in the appropriate table in your Google spreadsheet.

Keeping the result for elapsed time in mind, look through the table of counters and determine which are the most likely indicators of the difference in performance between the “bad” and “good” code. Hint: Look at the ratio of the elapsed times and the other counters. For the indicators you choose, use the perf man page or other information you find online to describe what the counter means. You should give a description/definition of what the counters you chose mean.

Review the source code for the two programs and explain how the difference in the code is consistent with the program execution times and the output of perf.

|  |
| --- |
| In terms of the most egregious downside of the way the “bad” code is written for this exercise, the dTLB load misses suffers the craziest comparison. There are 238x more dTLB load misses on the bad code example than that of the good code example. Why is that? The easiest answer I think is because when we associate and store rows of memory at a time, as is built in C++, the TLB quickly saves and identifies the locations of entire rows at a time. A[0][0] to A[0][1000] will be an easy lookup with the memory locations stored away in the TLB. However, when we do the code in the manner of the bad example, searching for columns of separation (I.E A[0][0] then A[1][0] then A[2][0] and so on…), we have to go look at the VRAM pages every single time and no memory locations will be stashed effectively in the TLB in between our searches, causing a load miss nearly every single column search. |

# Submit files to Canvas

When you complete the lab, download this document from Google Drive as a Microsoft Word (.docx) file with the naming convention Lab04 - MUid.docx (example: Lab04 - ferrenam.docx).  You should save the corresponding Google spreadsheet file as a Microsoft Excel (.xlsx) file with the naming convention Lab04 – Results MUid.xlsx (example: Lab04 - Results - ferrenam.xlsx). These are the default names you’ll get from downloading so you’ll be fine as long as you don’t change anything.

Then, submit the following files to Canvas:

1. The Microsoft Word file you downloaded from Google Drive.
2. The Microsoft Excel file you downloaded from Google Drive.
3. The Lab04.bash shell script you created for this lab.
4. Your Lab04-bad.perfstat file.
5. Your Lab04-good.perfstat file.