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

**Lab #05**

Max Points: 48

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

* Build experience with compiling code at the command line and constructing batch shell scripts
* Continue to build experience with the Linux perf hardware-assisted profiler.
* Gain experience with loop unrolling as a technique for reducing hazards in a program that can impede instruction level parallelism
* Learn about the g++ -funroll-loops command-line option to enable automated unrolling of loops.
* Gain experience the Linux nohup and & commands.
* Learn about the concept of *relative error*.

Fill in answers to all of the questions here and in the corresponding spreadsheet. You may discuss the questions with your instructor and the TA. **Note that this lab has a question that only CSE543 students must answer.**

## Background

Download the program unroll.cpp from Canvas and save it to your lab computer. This program was adapted from the software optimization guide provided by AMD and is used to demonstrate how to resolve control hazards that cause stalls in the pipeline for small loops.

The supplied unroll.cpp program uses 4x4 matrix multiplication to transform a point. The program performs a standard 3-D transformation (specified in 4x4 matrix m) to a given point (v on the x, y, and z planes) resulting in a new point (r) via the following matrix multiplication operation: r = v \* m; where

* m is a 4x4 matrix that is setup to do a graphics operation (such as: translation, rotation, scale, etc.)
* v is a 1x4 array containing the x, y, and z value for a given point.
* r is a 1x4 array that will contain the result of applying the transformation (m) to the specified point (v) through matrix multiplication.

Study the unroll.cpp program, in particular the method standard(). This function provides the default/reference implementation, which is typically used by almost all programmers to implement simple matrix multiplication. The exact nature of the operations being performed is not as important. What is important is to note that the nested for-loops in this method constantly require branching back to the top of the loops. This introduces a control hazard in the pipeline causing the pipeline to stall.

On the other hand, the unrolled() method in unroll.cpp takes an aggressive approach by completely unrolling the loop. As we discussed in class last session, loop unrolling is a standard programming technique that is used to minimize the number of control hazards in looping constructs by simply repeating the body of the loop several times.

Manually unrolling loops can benefit performance, especially if the loop body is small, which makes the loop overhead significant. Review the unrolled() function that aggressively unrolls the nest for-loops.

One downside of manually unrolling loops is that the resulting code is more difficult to update or maintain as nearly-identical pieces of code get repeated. If a change is needed in that code, it must be propagated to multiple places resulting in additional effort and an increased chance of introducing errors to the program. Because of this, compilers have been updated with the ability to unroll loops at compile time if given the proper command line option or optimization level. In this lab we will be comparing the effectiveness of this automatic unrolling to manual unrolling of loops. Note that you will be compiling your code with optimization level O2 instead of O3 because automatic unrolling is part of level 3 optimization.

One additional concept added into this lab is that of *relative error*. It is defined as the standard error divided by the average value. This is a dimensionless measure of how large the error is and it is useful when comparing the measured precision of quantities whose average values vary significantly.

## Experiment

1. You should write a bash shell script called Lab05.bash that accomplishes the items below. You can certainly refer to previous shell scripts as many of the commands will be similar. You will be uploading this shell script as part of your submission for this assignment so please **make sure to follow these instructions completely!**
   1. Execute /bin/bash as a login shell
   2. Get a single line of information about the model of the CPU you’re running on and its speed
   3. Get information about the total amount of memory available on the system you’re running on.
   4. Compile the program unroll.cpp with the following options and name the resulting executable unroll

-g -O2 -Wall -std=c++14

* 1. Compile the program unroll.cpp with the following options and name the resulting executable unroll-with-funroll

-g -O2 -Wall -std=c++14 -funroll-loops

* 1. Run unroll with the command line option standard five times and profile it with perf stat, appending standard error to the file Lab05-standard.perfstat. You should capture at least the four events requested in the “observations” tables in your spreadsheet.
  2. Run unroll with the command line option unrolled five times and profile it with perf stat, appending standard error to the file Lab05-unrolled.perfstat. You should capture at least the four events requested in the “observations” tables in your spreadsheet.
  3. Run unroll-with-funroll with the command line option standard five times and profile it with perf stat, appending standard error to the file Lab05-standard-with-funroll.perfstat. You should capture at least the four events requested in the “observations” tables in your spreadsheet.

1. Run your Lab05.bash shell script with the following command line:

nohup ./Lab05.bash &

The ‘&’ runs the command in the background and the nohup command will keep the command running even if the terminal window is closed. It will write standard output and standard error of the command to the file nohup.out, unless they have been redirected elsewhere, as is the case with standard error in this experiment.

# Apparatus (platform for experiment)

The experiment documented in this report was conducted on the following platform (fill in three lines of 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

Result of the calculation. It should be the same for all 15 runs. Enter the numerical result in the box below:

|  |
| --- |
| -9071744077459551616 |

The perf stat command gives you information on various counters that the hardware keeps track of while the program runs. Record the values of the counters requested in the Observations section of the Lab05 – Results spreadsheet.

# Analysis

Using data from your observations, compute the average, standard error and relative error of the quantities requested in your Lab05 – Results spreadsheet. Then answer the questions below.

Which approach was more effective in speeding up the program: Manually unrolling the loop or letting the compiler do it? Based on your observed data, which events (Branches, Branch-Misses, or L1-dcache-load-misses) are the most likely indicators of the difference in the observed Task-clock time? Be quantitative, using results from your observations to explain/justify your answer.

|  |
| --- |
| For my personal results, the manually unrolled operation outperformed both the compiler (Standard-with-funroll) and the standard, rolled up method calls. Branches are ineligible as our speed up criteria as there were actually more branches created by the unrolled solution. This leaves us with branch-misses and L1-dcache-load-misses as possibilities. There are a lot less L1-dcache-load-misses for the unrolled option compared to the standard with funroll option, and with a smaller relative error, I believe this is the stronger contender for the cause of the slowdown. |

**For CSE543 students only:** Based on information from Session 08, identify one additional change that could have been made to improve the performance of the compiled code. Hints:

1. You could make the change via a compiler option or by modifying the code
2. The correct answer is **NOT** increasing the optimization level from –O2 to –O3

|  |
| --- |
|  |

# 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 Lab05 - MUid.docx (example: Lab05 - ferrenam.docx).  You should save the corresponding Google spreadsheet file as a Microsoft Excel (.xlsx) file with the naming convention Lab05 – Results MUid.xlsx (example: Lab05 - Results - ferrenam.xlsx)

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 Lab05.bash shell script you created for this lab.
4. Your Lab05-standard.perfstat file.
5. Your Lab05-unrolled.perfstat file.
6. Your Lab05-standard-with-funroll.perfstat file.