ECE454, Fall 2020
Homework1: Profiling and Compiler Optimization
Assigned: Sept 17th, Due: Sept 27th, 11:59PM

The TAs for lab assignment 1: Jack Luo (`jack.luo@mail.utoronto.ca`)

1 Introduction

Upon graduation from Skule, and having become a high-performance program-optimizing guru, you decided to open your own high-performance code optimization consulting firm. After great success with your previous client, you have a second client: a virtual reality headset startup. The startup is co-founded by a group of hardware geeks: those who like to design electrical circuits and integrate sensors. The VR headset prototype hardware is almost ready but lacks a high performance software image rendering engine. The hardware engineers have already wrote functionally correct code in C, but wants your help to supercharge the software performance and efficiency.

The rendering engine's input is a preprocessed time-series data set representing a list of object manipulation actions. Each action is consecutively applied over a 2D object in a bitmap image such that the object appears moving in reference to the viewer. In order to generate smooth and realistic visual animations, sensor data points are over-sampled at 1500Hz or 25x normal screen refresh rate (60 frames/s).

Figure 1 shows all of the possible object manipulation action. The goal of the rendering engine is to process all of the basic object manipulation actions and output rendered images for the display at 60 frames/s.

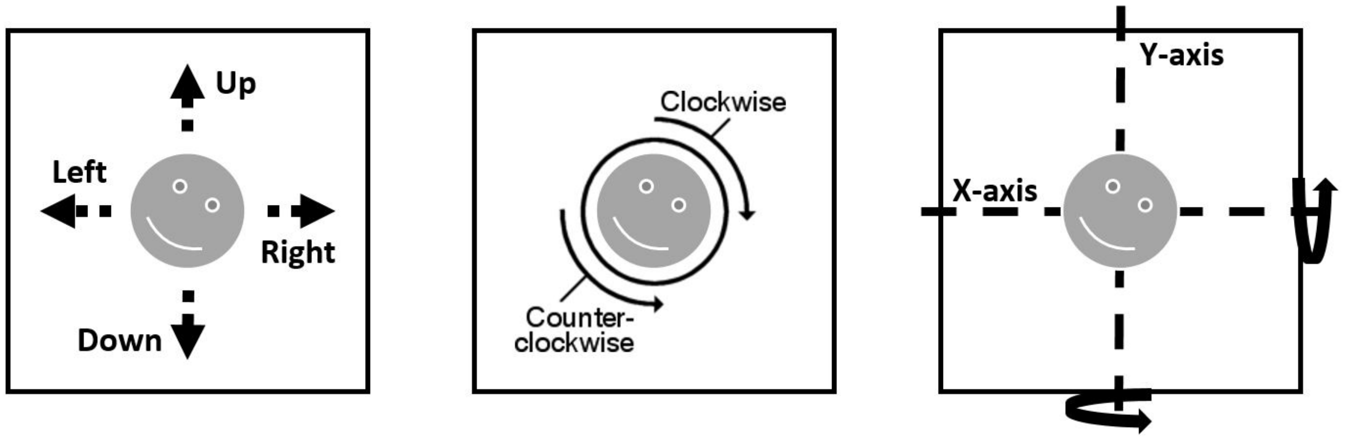

Figure 1: Basic Actions

2 Procedure

For this lab, please answer all questions below with few bullet points or short sentences. Performance measurements must be perform on UG machines.

2.1 Source Code

Start by copying the `hw2.tar.gz` file from UG shared directory
`/cad2/ece454f/hw2/hw2.tar.gz`
into a protected directory within your **UG** home directory and then run the command:
`tar xzvf hw2.tar.gz`
This will cause a number of files to be unpacked into the directory.

2.2 Build and Test

The lab assignment utilize open-source cross-platform CMake packaging system to manage the source code. Unlike the simple projects you have seen before, the Makefile is automatically generated based on your computer configuration.

Below are the instructions to compile the the project:

```
> cd <project directory>      // Navigate to the lab assignment directory you extracted
> mkdir bin && cd bin         // Make a new directory called bin, then navigate inside
> cmake ../                   // Use cmake to generate Makefile automatically
```

After the simple configuration steps, the make file is automatically generated. Simply run the Makefile and an executable named `ECE454_Lab2` should appear within the `bin` folder.

When you run ECE454_Lab2 binary using the command below, you should receive similar output.

```
ugXXX:~/ECE454-Lab2/bin% ./ECE454_Lab2 -g -f ../lab1.csv -i ../lab1.bmp
Loading input sensor input from file: ../lab1.csv
Loading initial 2D object bmp image from file: ../lab1.bmp
*************************************************************************************
Team Information:
         team_name: default-name
         student_first_name: john
         student_last_name: doe
         student_student_number: 0000000000
*************************************************************************************
Performance Results:
         Number of cpu cycles consumed by the reference implementation: 33492345978
         Number of cpu cycles consumed by your implementation: 33170344755
         Optimization Speedup Ratio (nearest integer): 1
*************************************************************************************
```

Tip: You will be investigating and profiling lab2's source code in lab1. You will not be modifying the source code in lab1 thus not required to understand what the code is doing exactly. However, it may be a good idea to use this opportunity to explore around and think about how you can optimize it in lab 2 while you work on lab 1.

Q1 (1 mark) List the function you think might be important to optimize to in lab2's source code? (freebie)

2.3 Measuring Compilation Time

In this assignment you will use /usr/bin/time to measure compilation and performance. In the output, note that the number that ends in "user" is runtime in seconds for "user-mode", the time to use for this report, except Section 3.4. Note that since you are measuring real systems, measurements are a little different each time due to system variability. Try to measure on an unloaded machine. For every timing measurement always do 5 runs and average them (please only report the final average).

To build the gprof version, use the flags: -g -pg -no-pie

To build the gcov version, use the flags: -g -fprofile-arcs -ftest-coverage

Measure the compilation time of the gprof, gcov, -g, -O2, -O3, and -Os compilation flags. Be sure to **regenerate** the make file and run "make clean" in between each build to ensure that all files are rebuilt properly.

Note: We intentially left out details on how to add compiler flags to CMake to encourage you to practice reading and researching CMake's documentation or stackoverflow pages.

Q2 (1 mark) Report the 6 measurements using the slowest method of compilation as a baseline, report the speedup for each of the five measurements. Eg., If gcov was the slowest, and -g was twice as fast as gcov, then the speedup for -g relative to gcov would be 2.0.

Q3 (1 mark) Which is the slowest and why?

Q4 (1 mark) Which is the fastest and why?

Q5 (1 mark) Which of gprof and gcov is faster and why?

2.4 Measuring Program Size

Use "ls -l" to measure the size of each version of the binary from the previous section.

Q6 (1 mark) Report the six measurements using the smallest method of compilation as a baseline, report the relative size increase for each of the six measurements. Eg., if -g was the smallest, and gprof was twice the size of -g, then the relative size increase for gprof relative to -g would be 2.0

Q7 (1 mark) Which is the smallest and why?

Q8 (1 mark) Which is the largest and why?

Q9 (1 mark) Which of gprof and gcov is smaller and why?

2.5 Measuring Performance

Measure the run-time for all six versions compiled in the previous section.

Q10 (1 mark) Report the six measurements using the slowest measurement as a baseline, also report the speedup for each version.

Q11 (1 mark) Which is the slowest and why?

Q12 (1 mark) Which is the fastest and why?

Q13 (1 mark) Which of grof and gcov is faster and why?

2.6 Profiling with gprof

Compile gprof support for the -g, -O2, and -O3 versions, by using flags "-g -pg", "-O2 -pg" and "-O3 -pg" respectively; run each of these versions to collect the gprof result; you don't have to time any of this.

Q14 (1 mark) For each version, list the top 3 functions (give function name and percentage execution time)

Q15 (1 mark) For the "number-one" function for -O3 (the one with the greatest percentage execution time), how does its percentage execution time compare with the percentage execution time for the same function in the -g version? How is this possible? What transformation did the compiler do and to which functions?

2.7 Insepection Assembly

Use objdump to list the assembly for the -g and -O3 versions (eg., run "objdump -d OBJ/main.o" to examine the assembly instructions for the file main.c).

Q16 (1 mark) Count the instructions for the function ”number-one” function identified in the previous question and report the counts, as well as the reduction (reported as a ratio) in number of instructions for the -O3 version (ie., if the -O3 version has half as many instructions as the -g version, the reduction is 2.0x).

2.8 Profiling with gcov

Use gcov to get the per-line execution counts from the -O3 version (but use the -g version to gather the gcov profile). After running the gcov version, execute the gcov program to generate a profile of the appropriate file (eg., run “gcov -o OBJ -b main.c” to profile the file main.c). Running gcov will create main.c.gcov (for main.c). NOTE: if you run the gcov program multiple times it will add to the counts in main.c.gcov; you have to remove the .gcda and .gcno files in OBJ/ to start counting from zero.

Q17 (1 mark) Based only on the gcov results (ie., don’t think too much about what the code says) list the functions in the order that you would focus on optimizing them for the provided lab1 inputs and why. Identify each location by its line numbers in the original source file.

2.9 Get familiar with GCC man page

Use **man gcc** shell command or view it online at https://man7.org/linux/man-pages/man1/gcc.1.html

Q18 (1 bonus mark) Name the shortest GCC compiler flag (i.e, -xxxxxxxxxxx) where the compiler optimization it enables requires memory alignment. How many bytes does the data needs to be aligned?

3 Submission

Please record your answer in lab1.txt file in **/cad2/ece454f/hw1/lab1.txt**. Make sure you fill out your personal detail before submitting using the following command.

submitece454f 1 lab1.txt

Changing the format or failure to submit using the template file will result in a mark of zero. If you wish to change your report, you may overwrite your submitted file by executing the above script again. To view your submission, enter

submitece454f -l 1