Computer Architecture- Spring 2018- Lab #4

Performance Measurement and Optimization

Weighting: 4 Days

Due: Tuesday May 1, 2018 11:55pm

You may work in pairs for this assignment.

Purpose

To measure and optimize system performance for a specific application, and to experiment with compiler optimizations and profiling tools. Cycle count, instruction count, and the performance equation will also be directly measured. Amdahl's Law will also be directly measured.

Method

In this lab, you'll implement a function, matadd, in ARM assembly that computes the sum C of two matrices A and B, which are matrices with integer values. A declaration of the function in the C language is as follows:

```
void matadd (int **C, int **A, int **B, int height, int width)
```

The function needs to compute C = A + B, where height and width specify the dimensions of all three matrices.A, B, and C are arrays of arrays of integers. So for each element i,j in the arrays, C[i][j] = A[i][j] + B[i][j]. The elements of each row of a matrix are allocated sequentially, but the rows themselves may not be consecutive in memory.

A driver routine that allocates and initializes the matrices, calls matadd iteratively, and prints the result is provided. You'll use a compiler to test the effect of different optimization levels on runtime performance, and to demonstrate understanding of profiling tools applied to a real application.

Program Specifications

- Your function must obey the ARM register usage convention, and procedure calling convention.
- Only array C should be modified in your function. A and B should be left unmodified.
- Strive for the lowest dynamic instruction count and CPI. You should be able to implement this function such that the whole program executes less than 1 billion dynamic instructions.
- Your program must compute the correct output. A file with the correct results is provided.

What to Do

Read this writeup carefully. There are several steps, with many details that need to be followed precisely for you to get the correct information.

You will measure the performance of your program on the ARM architecture. You will measure runtime directly using the 'perf' command (Note that this only works on Raspberry Pi versions 2 and 3, not version 1). Perf will be accessing the performance counters available on the system.

You may assume that the matrices are correctly allocated and initialized, are of the same size, and don't contain non-integer values.

On the Pis, you will compile your program (with GCC) using four (4) different compiler optimization levels; -O0, -O1, -O2, -O3.

For each experiment, follow these steps:

1. In your matadd source directory, compile the code with one of the optimization levels. For example:

1. cd matadd
2. Edit the Makefile and change the optimization level in the CFLAGS parameter
3. make

2. Use the performance measurement tools

1. run perf on the executable:

1. perf stat ./matadd > my.out
2. Record the Execution time, Cycles, Instructions, and Branch Misses.

3. Repeat this procedure for your program for all 4 optimization levels.
4. For the O1 version of your program, profile the application:

1. Add the -pg flag to the CFLAGS parameter in your Makefile
2. Run the program. (This is needed to profile the executable, but you don't need to record the runtime) Do this the same way you did in 2. above.
3. gprof ./matadd | less
4. Record the fraction of time which is spent executing your matadd function.

Chart your Run times

You must create a graphical chart of the execution run times, one per machine, with all optimization levels, and present the chart and a table with your data in your report.

An example table:

matadd - Pi	-O0	-O1	-O2	-O3
Average CPI				
Instructions				
Branch-misses				

Runtime (Measured)

Runtime (Equation)

You should consider writing a script to generate all this data!

Loop Unrolling And Amdahl's Law

You should implement loop unrolling (as discussed in class), on the inner-most loop of your matadd function. Try three different levels of unrolling: two, four, and eight. Record your results for cycle count, instruction count, branch-misses, and execution time, and include them in your report.

Using the measured fraction of time your matadd function spends executing, compute the enhancement E to your matadd function using Amdahl's Law. You have measured F, and you can compute Speedup directly. Include the calculation in your report.

Performance Equation

Describe any differences between your measured runtimes and those predicted by the performance equations. Show your tabulated results in your report.

Web Resources

https://perf.wiki.kernel.org/index.php/Main_Page- Linux perf Documentation

http://gcc.gnu.org/- GCC Homepage

http://sourceware.org/binutils/docs/gprof/- gprof Documentation

What to Hand In

All of your source code, any Makefiles you use, and an electronic copy of your written report in PDF format. Please do not submit Microsoft Office documents. One report per group. Also submit a documented version of your script if you used one to generate the data for your report.

Report Requirements

Your report should contain team member names, all data collected in your experiments, for both machines for all 4 levels of compiler optimization. Describe the differences in instruction and cycle counts for the different machines. Also describe your observations about how the compiler is optimizing your code.

Formatting requirements:

- Use standard fonts (e.g. Arial or Times), with minimum size of 10pt and maximum size of 12pt
- Margins should be 2cm on all sides
- Tables and figures should be labeled and captioned
- Only PDF documents will be graded (no Word, openoffice, google docs, etc. allowed)

Submission Instructions

Submit the following files via Polylearn or GitHub(if this is your option upload link to your Github to polylearn):

All source files

Report in PDF format (no Docs!):

Any scripts you wrote/used.

Updated README with the usual content.