University of Pennsylvania
Department of Electrical and System Engineering
System-on-a-Chip Architecture

ESE532, Fall 2017 HW4: SIMD Wednesday, September 20

Due: Friday, September 29, 5:00PM

In this assignment, we will accelerate the streaming application from last homework using the ARM NEON vector processor. Note that there were a few modifications to the application. You can find the sources for this homework the course website.

Collaboration

In this assignment, you work with partners that we assigned. You can find the assignment on Canvas in the *Partners* map under the *Files* section. In the event that the partner assignment does not work out, contact the instructor or TA as soon as possible. Partners may share code and results and discuss analysis, but each writeup should be prepared independently. Outside the assigned groups, only sharing of tool knowledge is allowed. See the course policies on the course web page `http://www.seas.upenn.edu/~ese532` for full details of our policies for this course.

ARM NEON

Information about the NEON architecture and datatypes is available in the ARM assembler user guide. Another section in the same guide lists the instructions. Note that not all information may be applicable to the ARMv7 architecture of the Cortex A9 processor that we are using. You are encouraged to locate other sources as needed and to share them.

Homework Submission

1. **Teamwork**

   As the difficulty of homework is ramping up, we encourage you to spend a moment planning on how to tackle the homework as a team.

   (a) Describe which tasks of this homework you will perform, which tasks will be performed by your teammate(s), and which tasks you will perform together (e.g., pair programming, where you both sit together at the same terminal). Motivate your task distribution. (5 lines)

Optimization level	Latency (Mcycles)	Code size (bytes)
`-O0`		
`-O1`		
`-O2`		
`-O3`		
`-Os`		

Table 1: Latency and Code Size per Optimization Level

(b) Give an estimate of the duration of each of the tasks. (5 lines)

(c) Explain how you will make sure that the lessons and knowledge gained from the exercises are shared with everybody in the team. (3 lines)

2. Compiler Optimizations

Before we dive into the vector optimizations, we will investigate the effects of different levels of compiler optimizations.

(a) Measure the latency and size of the `Baseline` project at the different optimization levels. Put your measurements in a table like Table 1. You can change the optimization level as follows: Right-click on the project in the *Project Explorer*, and select *C/C++ Build Settings* from the popup menu. In the *Tool Settings* tab, go to *ARM v7 gcc compiler* → *Optimization*, and select one of the optimization levels under *Optimization Level*. You can see the code size by opening the *CDT Global Build Console*. The code size is in the column *text*.

(b) Include the assembly code of inner loop of `Filter_horizontal` at optimization level `-O0` in your report.

(c) Include the assembly code of inner loop of `Filter_horizontal` at optimization level `-O2` in your report.

(d) Based on the machine code of questions 2b and 2c, explain the most important difference between the `-O0` and `-O2` versions. (2 lines)

(e) Why would you want to use optimization level `-O0`? Hint: Compile the code with `-O3` and track the values of the variables `X`, `Y`, and `i` as you step through `Filter_horizontal`. (3 lines)

(f) Include the assembly code of inner loop of `Filter_horizontal` at optimization level `-O3` in your report.

(g) Explain the most important difference between the `-O2` and `-O3` versions. (1 line)

(h) What are two drawbacks of using a higher optimization level? (5 lines)

3. Automatic Vectorization

The easiest way to take advantage of vector instructions is by using the automatic vectorization feature of the GCC compiler, which automatically generates NEON instructions from loops. The archive that we provided contains a project (`Vectorized`)

that has the right compiler settings to enable autovectorization. Automatic vectorization in GCC is sparsely documented in the GCC documentation. Although we are not using the ARM compiler, the ARM compiler user guide may give some more insight.

(a) Report the latency of each stage of the baseline application at `-O3`. (5 lines)

(b) Which loops in the streaming stages of the application have sufficient data parallelism for vectorization? Motivate your answer. (5 lines)

(c) What speedup do you expect your application can achieve under ideal circumstances? (5 lines)

(d) If you compare the `Vectorized` and `Baseline` projects, you will notice that frames are now assumed to be aligned to a multiple of 32 bytes. Moreover, we allocated 1024 bytes for each row instead of the exact row length (e.g., 960 bytes for the input of `Scale`). Explain how this may increase performance. Illustrate your point with a picture that shows the data loaded by a few different iterations of the loop, and the cache lines. Note that would also have benefited the baseline. (5 lines)

(e) Explain in your own words why adding the `__restrict` keyword to some of the pointers may have led to a better performance. (5 lines)

(f) Report the speedup of the vectorized code with respect to the baseline. (1 line)

(g) Explain the discrepancy between your measure and ideal performance based on the optimization of `Filter_horizontal`. Hint: look at the size of the multiplications in the disassembly. (3 lines)

(h) Show how you can resolve the issue that you encountered. (1 line)

(i) Report the speedup with respect to the baseline after resolving the issue in both `Filter_horizontal` and `Filter_vertical`. (1 line)

4. **NEON Intrinsics**

We will accelerate the `Filter_vertical` function using intrinsics. An intrinsic behaves syntactically like a function, but the compiler translates it to a specific instruction that is inlined in the code. The Neon intrinsics are listed in the ARM documentation.

(a) How can we deal with a number of data elements that is not divisible by the number of vector lanes without losing significant performance? (3 lines)

(b) Explain at which granularity and in which order you should process the input data with vector instructions to achieve a good performance. Motivate your answer. Hint: Minimize the number of loads. (7 lines)

(c) Using NEON intrinsics, accelerate the `Filter_vertical` function. Include the accelerated function in your report. Make sure that you verify your optimized code properly.

(d) Report the latency of `Filter_vertical` and the application as a whole. (2 lines)

(e) Identify the critical path lower bound for `Filter_vertical` in terms of compute operations. Focus on the data path. Ignore control flow and offset computations. (5 lines)

(f) Report the resource capacity lower bound. There are many resources that may limit the performance. Choose one that makes sense for your application. (5 lines)

(g) Compare your performance with the lower bounds. (1 lines)

(h) Compare the performance of manual and automatic vectorization. (1 lines)

5. **Reflection**

Reflect on the cooperation in your team.

- What was the most useful thing you learned from or working with your teammate? (3 lines)
- What do you believe was the most useful thing that you could contribute to your team? (3 lines)