CprE 381 Homework 6

[Note: This homework's purpose is to increase your comfort at calculating and analyzing the performance of processors. By the end, you should be able to accurately estimate how a change to software or hardware will likely impact the overall execution time of an application.]

1. Amdahl's Law

   Your company builds hardware for doing machine learning inference for image classification. 'Inference' is predicting the class of an image based on the model you learned using sophisticated machine learning algorithms. While everyone is busy trying to develop the next Neuromorphic or Quantum computing chip, your supervisor assigns you the tasks of optimizing the execution of the time of the only application that actually generates the $$$ for your company (i.e., image classification of cats). For image inference: **Convolution** is the most fundamental operations which consists of convolving or "sliding" a filter (sometimes called the learned kernel 2D array) of size k*k across the input image of size n*n which generates an output of size (n-k+1)*(n-k+1). If you want more information about this sort of convolution, you can view the following video: https://www.youtube.com/watch?v=XuD4C8vJzEQ&list=PLkDaE6sCZn6Gl29AoE31iwdVwSG-KnDzF&index=2.

   The attached C and MIPS code implement a vertical edge detector that is an important portion of the cat image classification algorithm. As you can see, if there is a hardware floating point multiplier in the system the program will use * that will compile to mul.s MIPS instructions, otherwise it will use library call that performs software floating point multiplication. Your specific task is to determine whether or not you should add a hardware multiplier to your MIPS processor.

   a. Based on your understanding of the MIPS and the C code *estimate* the total number of clocks cycles the application will require to execute on your single-cycle processor. Also estimate the fraction of these cycles that the software multiplication function requires. *[Note: this is a somewhat open-ended problem and you can make any reasonable assumption. Please state all the calculations and assumptions you make. Note that you can actually run the MIPS code on MARS with and without the hardware multiplier to get instruction counts.]*

One method to estimate the fraction of cycles taken by the software multiplication routine is to simulate the whole application in MARS and use its Instruction Counter tool:

STEPS to find Instruction Statistics in Mars**:**

1. Download and open the Mars Framework: http://courses.missouristate.edu/KenVollmar/mars/

2. Click on the file tab and open the required Mips assembly file.

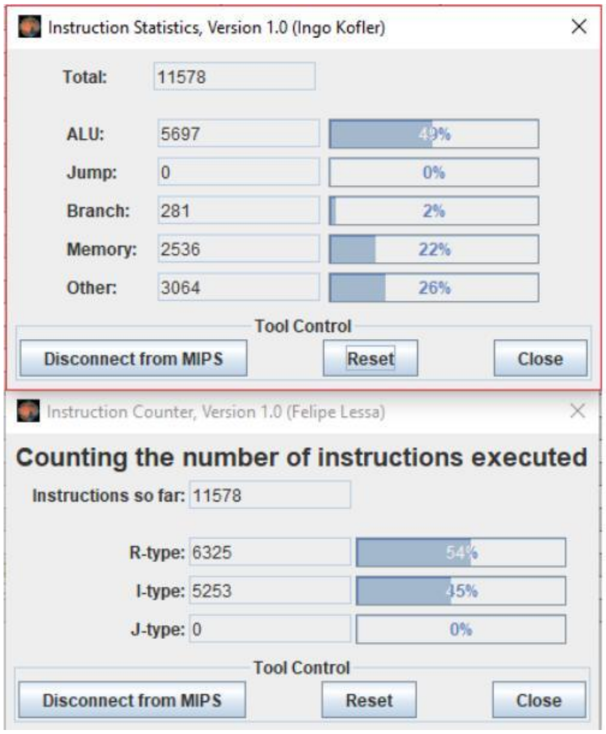
3. Click on the run tab and Assemble the file.

4. Once successfully assembled click on the tools tab and then choose 'Instruction Statistics'. Also click on connect to Mips.

5. Click on the run tab and choose 'Go' to execute the program. You will get the total count of the instruction executed and the types of instructions (ALU, Jump, Branch, and Memory & Others). You can follow a similar process and choose 'Instruction Counter' option in tools to display various types of instructions (R, I, J) executed.

Since we are assuming a single-cycle processor design, the number of cycles the application takes is simply the number of instructions executed:

Hardware Multiplier:


Instruction Statistics, Version 1.0 (Ingo Kofler)
Tool Control
Disconnect from MIPS
Reset
Close
Instruction Counter, Version 1.0 (Felipe Lessa)
Tool Control
Disconnect from MIPS
Reset
Close

Total: 11578

ALU: 5697 49%

Jump: 0 0%

Branch: 281 2%

Memory: 2536 22%

Other: 3064 26%

Counting the number of instructions executed

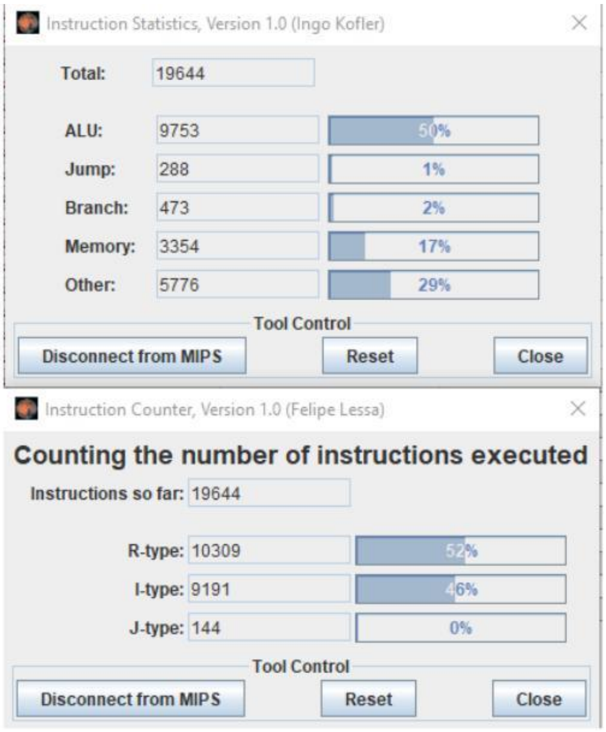
Instructions so far: 11578

R-type: 6325 54%

I-type: 5253 45%

J-type: 0 0%

Software Multiplier:


Instruction Statistics, Version 1.0 (Ingo Kofler)
Tool Control
Disconnect from MIPS
Reset
Close
Instruction Counter, Version 1.0 (Felipe Lessa)
Tool Control
Disconnect from MIPS
Reset
Close

Total: 19644

ALU: 9753 50%

Jump: 288 1%

Branch: 473 2%

Memory: 3354 17%

Other: 5776 29%

Counting the number of instructions executed

Instructions so far: 19644

R-type: 10309 52%

I-type: 9191 46%

J-type: 144 0%

Therefore, assuming a single-cycle processor (including single-cycle hardwre multiply), the application takes **11578** clock cycles with the hardware multiplier and **19644** clock cycles with the software multiplier. A single convolution of a 6*6 array with a 3*3 kernel will result in a total of 4*4*(9(point wise multiplication per convolution)) = 144 multiples. Since the hardware multiple assembly code only takes one mul.s instruction vs all of the function call instructions (both setup, call, and function body instructions), there are **144** hardware multiply cycles. Now we can calculate the fraction of cycles executed by the software multiply: # instructions executed by software multiply version - #instructions executed with hardware multiplier - # hardware multiply instructions executed = 19644-11578-144 = **7922**. [Assuming the clock cycle duration remains same during both the software and hardware multiply]. Fraction is 7922/19644 ~= **40.3%**

b. Based on the above question identify which part of the application will benefit from the hardware multiplier and why? What is maximum speed up possible? *[Again, you will have to make assumptions regarding the multiplier unit.]*

All of the software multiplication portion (40.3%) of original application will benefit using the hardware multiplier because software multiplier takes 7922 cycles while hardware multiplier will take a total of 144 cycles to do the same operation. Of course, this assumes a single-cycle processor and no impact of hardware multiplier on clock frequency.

For the maximum speed up possible there may be different answers depending upon what assumptions are taken as discussed below:

1. Assuming the improvement factor goes to infinity which means the hardware multiplier takes 0 cycles and so according to Amdahl's law the maximum speedup is 1/(1-f) where f is the fraction of the code that could be enhanced. Since f is 0.40 here, the maximum speed up is (1/ (1-0.40)) ~= **1.67**. Clearly this is a big assumption considering the new multiplier is "magic" and takes no time at all, including no increase in frequency.

2. Assuming that each of the hardware multiplication takes one clock cycle and so in total 144 cycles for the convolution and so according to Amdahl's law the maximum speed up is (again, there is the assumption of no impact of the hardware multiplier on the clock frequency):

   1/ (1-fraction enhanced + improvement factor)
   Improvement factor =
   Proportion of program in software multiplier/ Speed up by the hardware multiplier
   Now speed up by the hardware multiplier = 7922/144 = ~ 55
   .403 / (7922/144) = **0.007325**
   Maximum Speed up possible = 1/ (1-0.403+0.007325) ~= **1.65** (not too far off from the 0-cycle case)

   More realistic assumption would be considering the impact of hardware multiplier on the clock frequency because each multiplication might take multiple cycles as it needs to execute on floating point co-processor in MIPS and for a single cycle processor it will make the overall clock frequency slower and slower clock will penalize all instructions. Now assuming each multiplication takes c cycles and the processor overall frequency is slowed by s.

   Maximum speed up possible = 1/ (1-fraction enhanced + improvement factor)

   Improvement factor =
   Proportion of program in software multiplier/ Speed up by the hardware multiplier
   Now Speed up by the hardware multiplier = 7922/144*c = ~ 55/c
   .403 / (7922/144*c) = 0.007325c
   Maximum Speed up possible **=** 1/ (1-0.403+0.007325c)*(1+s)

c. What would the maximum speedup be if your hardware multiplier slowed the clock frequency by 25%? *[You can still solve this by using Amdahl's law—you just have to appropriately adjust the equation from b.]*

Maximum Speed up possible = 1/ ((1-0.403+0.007325)*(1/(1-0.25))) ~= **1.2375** since the clock frequency is slowed by 25% and so each instructions take 1/0.75 = 1.33 extra time than the original.

2. Pipelining Cycle Time

a. Assuming the following worst-case latencies for components, what is the cycle time for the pipelined processor in Figure 4.7.1 (COD Figure 4.33) from your textbook? You must quantitatively justify your answer (e.g., specify what set's the cycle time and why it set's the cycle time).

I-Mem	Adder	MUX	ALU	Reg Read	D-Mem	Sign-Extend	Shift-Left-2	Control	ALU Control	AND gate
230ps	80ps	25ps	130ps	120ps	270ps	20ps	5ps	50ps	20ps	10ps

The cycle time of the pipeline is set by the longest propagation delay of any stage.

Stage	Critical Path	Critical Path Time
Fetch	I-Mem	230ps
Decode	Reg Read	120ps
Execute	MUX + ALU	155ps
Memory	D-Mem	270ps
Writeback	MemtoReg Mux	25ps

Since the Memory stage has the longest critical path at 270ps, the pipeline's cycle time is 270ps (i.e., ~3.70 GHz :o).

b. Compared to a single-cycle processor Figure 4.4.5 (COD Figure 4.17) with the above component latencies, what would the pipeline design's CPI need to be in order to benefit performance?

As we've seen in HW5, the memory access path in a single-cycle processor (Figure 4.17 COD) causes the critical path. Calculating the cycle time for it:

Critical path: I-Mem ➔ Reg Read ➔ ALU ➔ D-Mem ➔ MUX (MemReg)
Latency: 230ps + 120ps + 130ps + 270ps + 25ps = 775ps

For the pipelined processor to benefit performance:

$\text{CPU time}_{\text{pipe}} < \text{CPU time}_{\text{single-cycle}}$

$\text{CPI}_{\text{pipe}} \text{ x Instruction-count x Cycle time}_{\text{pipe}} < \text{CPI}_{\text{single-cycle}} \text{ x Instruction-count x Cycle time}_{\text{single-cycle}}$

For a given program P, the number of instructions executed is the same.

$\text{CPI}_{\text{pipe}} \text{ x } \cancel{\text{Instruction-count}} \text{ x Cycle time}_{\text{pipe}} < \text{CPI}_{\text{single-cycle}} \text{ x } \cancel{\text{Instruction-count}} \text{ x Cycle time}_{\text{single-cycle}}$

$\text{CPI}_{\text{pipe}} < 1 \text{ x } 775\text{ps} / 270\text{ps}$

$\text{CPI}_{\text{pipe}} < 2.87$