Name:_____ Email:_____@ncsu.edu

ECE 461/561, Spring 2022: Quiz 2 Solution
Analyzing and "Optimizing" Code for Speed

This quiz is closed-computer, closed-notes. You may use one 8.5" x 11" sheet of paper with anything you want written or printed on its two sides.

Assume the code is built using MDK-ARM (armcc compiler, armlink linker, all settings for maximum optimization for time) for the Kinetis KL25Z128 MCU used on the FRDM-KL25Z evaluation board and the core clock frequency is fixed at 48 MHz. Students in ECE 561 must answer all questions.

Analysis with a Profile

Consider the following execution time profile for a program.

Total Samples	1000	Sample count for entire program execution
Update_Screen	400	User-written function
__aeabi_fmul	310	floating point multiply library function
sinf	180	floating point sine library function
analyze_data	80	User-written function
calc_statistics	30	User-written function

1. When trying to optimize the program, which function would you start with, and why?

   Update_Screen, since it takes the most time.

2. How can you reduce the time taken by **sinf** without optimizing or replacing it?
   Reduce the number of times it's called. (Either improve the code (e.g. lazy execution) or cache (reuse) previous result(s).
   OK: --fpmode=fast. We're just linking to sinf, not compiling it.
   Leverage periodicity (with example)

   Incorrect: changing to single precision (is already single-precision), changing to fixed-point (not allowed to replace it) optimizing argument to sinf,
   Single precision has only one decimal point

3. You want to replace **sinf** by replacing it with a faster function. Explain two different ways to calculate the value of sine faster than **sinf.**

   A lookup table stores pre-calculated values of sine in an array. At run-time the correct entry is identified and read.

   A polynomial approximation uses an equation (e.g. based on a Taylor Series expansion) to calculate an approximate value of sine.

   Incorrect: Call cosine instead. Define it as a macro.

4. If you could optimize only the function **__aeabi_fmul**, what is the minimum possible value of Total Samples?

   That function takes 310 samples. If we optimized it down to 0 samples, the minimum possible value of Total Samples would be 1000 – 310 = 690 samples.

   OK: 691

Partial: providing value which is not the limit, but with context/justification.

Incorrect/Extra information: Changing position of __aeabi_fmul in profile table. Changing from double to single precision (is already single precision). Can't optimize since it's a library function.

Analysis without a Profile

Consider the following function code. It processes an array **d** of **n** integer elements to calculate **result**.

```
1  int Evaluate(int * d, int n, int mode) {
2      // d points to an array of integers
3      int t, result = 0;
4      do {
5          switch (mode) {
6          case 1: // count 1 bits in element
7              t = *d;
8              while (t > 0) {
9                  result += t & 1;
10                 t >>= 1;
11             }
12             break;
13         case 2:
14             result *= 1/(*d);
15             break;
16         case 3:
17             result -= *d;
18             break;
19         default:
20             break;
21         }
22         d++;
23         n--;
24     } while (n>0);
25     return result;
26 }
```

5. Which line of code is most likely to dominate execution time if **mode** == 1? Explain why.

There are two possible answers:

Line 9, because it needs to perform two instructions: an AND and then an ADD.

Line 8, because it performs a comparison and then conditional branch (two instructions). It executes once more than lines 9 or 10 (the last time is when t reaches 0).

6. **561 Only:** In some situations, your previous answer's code won't dominate execution time. Explain why.

If t is ≤ 0, then the loop doesn't execute at all. The number of loop iterations depends on where the most-significant 1 bit is in the word. The farther it is to the right, the fewer shifts it will take to zero out t and end the loop.

7. Will the function complete faster with **mode** == 2 or **mode** == 3? Explain why.

Mode 3 will be faster, since line 17 (subtract) is faster than line 14 (divide, requires a library routine call).

*Also OK: Mode 2 will be faster because compiler may see 1/(*d) can either be 1, 0 or -1, since *d is an integer, so result is either zeroed, negated or retained.*

Partial Credit: "Yes, modes 2 and 3 will be faster than mode 1." Mode 2 will be slower (division and multiplication) but mode 3 will be faster (subtraction).

8. You want to optimize the code's run-time for all values of **mode**. Describe how would you change the code, and why it would result in a speed-up.

*Exchange the nesting of the **switch** statement and the **do/while** loop. Edit each switch case to contain a loop. The switch test and jumps would only be executed once, not **n** times.*

Wrong: replacing floating point with a faster version (single-precision, or fixed point). There are no floating point data or operations in the function.

Passing entire array d (vs. just the pointer).

9. **561 Only:** Would replacing the variable **t** with ***d** be likely to speed up the code? Why or why not?

No, it wouldn't speed up the code.

The code uses variable t as a temporary variable, so the compiler has probably already optimized it by promoting it to a register, eliminating extra memory traffic. In fact, it might significantly slow down the code for mode 1 depending on optimizations. Line 10 might zero out every element in d in memory, one shift at a time.

For the other modes there would be no improvement, since t is not used.