2025 ECE 153A/253, CS 153A - Homework 1

Reading: Read the Vivado Getting Started and MicroBlaze References, and the papers by Lee, Tennenhouse and (grads) Lavagno.

Problems

1. Caches are very commonly used to improve the average performance of processors by providing a small, fast memory for reused portions of program or data. The issue with a cache is that when the needed program fragment is not present in the cache, the processor waits while the data is fetched. Finally, for simplicity and memory performance, the replacement often is a "cache-line" of 32 to 1024 words. In this problem, you are to make measurements augmenting a program to model the timing of processor memory accesses. Note: You need a C compiler for this problem. If you do not have one on your local machine, you can make use of the remote machines. Please refer to the 'Compiling C on Linux' note on Canvas for instructions on compiling your code using the Linux command line.

   - An access to main memory takes a fixed setup of 8 cycles plus 1 cycle for each sequential word, reads to random addresses must again pay the setup overhead.
   - The cache access is 1 cycle if the data is in the cache, and if not, it reads an 8-word cache line from the main memory (taking same time as a processor access to main memory), then an additional 3 cycles to update and respond to the read. As part of the update, some line in the cache is overwritten.
   - When the cache is disabled, a single buffer of 4 sequential words stores the result of the previous memory read.
   - The model assumes 16-bit address for words and a cache size of 256 lines, each line consisting of 8 words.

   A program for the model (in C) is provided, you should use this model and write C code that interfaces to this, providing addresses and gathering statistics. Due to the vagaries of random number generation, we have provided a random generator that you can use: `rand_int(int b)` which generates numbers in the range $[0, b-1]$. Technically, this is a pseudo-random generator – so will always give the same sequence. You can avoid this by calling `rand_jump()` which skips the sequence a `calls). This effectively puts you in a new pristine sequence (as long as you limit your tests to` $2^3 2$ `calls). Your job is to call the given functions repeatedly and gather statistics for the problems below.`

   (a) `Determine the expected access time with cache enabled and disabled, given random reads of the 16-bit memory space.`

(b) Assuming that:
s=0.6 is the probability that the next address follows the current one,
p=0.35 is the probability that the next address is not sequential but is within 40 words (either direction) of the current one,
thus 0.05 is the probability of a random far address,
determine the expected values for access time with cache enabled and disabled.

(c) What is the worst case access time for the memory?

(d) For the data gathered in parts (a) and (b) above, plot Cumulative Distribution Functions (CDFs). Based on the CDFs, what are the confidence limits on your data? Estimate the fraction of time for which the memory access time is less than the worst case access time calculated in part (c). Why is this hard to estimate in a real system?
Hint: The distributions you will find above hardly meet Gaussian (normal) statistics. A classic method to find the expectation of accuracy of a measuremen that does not require normal statistics is re-sampling. The idea is to do the experiment, say for 1000 address trials, and determine the expectation. Then redo the experiment a few dozen times - each time you should get a new expectation that isn't too different { but indicates the uncertainty of the measurement. In practice (bootstrapping), you can do this by randomly sampling a portion of the original measurement (when data is expensive to generate). Tools to perform bootstrap resampling are built in to both Mathematica™ and Python statistics and science packages.

2. A hand mixer is to be run by a grand loop controller on a small micro-controller. The following tasks must be run, each taking a certain amount of time:

Task	Description	Time to Run	Deadline
Ctrl	Control Calculations	9 ms	24 ms
Torq	Torque Check	3 ms	18 ms
Temp	Temperature Check	3 ms	30 ms
UI	User Interface	5 ms	30 ms

(a) What is the specification utility of the system?

In practice, the grand loop time is fixed by the fact that we can only switch power at the zero crossing of the mains voltage. Given 60 Hz mains and 2 zero crossings per cycle, this gives a loop time of 8 ms, i.e., reads and writes can occur every 8 ms. The deadline for each task is timed between consecutive writes of that task. Every write should be preceded by a read and complete execution of the task. Task execution can also be broken into any number of non-contiguous segments.

(b) Show a valid schedule with an 8 ms loop time. What is the period of execution for each task?

(c) What is the processor utility under this schedule?

Assuming the mixer is successful in the US market, it is desired to introduce it to the European market. Given 50 Hz mains, the loop cycle time will be 10 ms:

(d) Show that it is no longer possible to find a schedule under the above assumptions

(e) Given this, the torque check task is re-written so as to raise the deadline to 20 ms while still maintaining adequate safety. A schedule should now be possible. Find a schedule that works.

(f) What is the utility of that schedule? Is it higher or lower than the 60 Hz / 8 ms schedule?