ECE/CS 250 – Summer 2017 – Prof. Bletsch
Recitation #7
Caches and Memory

Objective: In this recitation, you will gain a greater understanding of the importance of caches/memory and how they work.

Complete as much of this as you can during recitation. If you run out of time, please complete the rest at home.

1. Caches and Memory in the Real World

1) How much cache is in your laptop? You may need to google this to find out. And to google it, you'll need to know exactly which processor you have. On a Windows machine, open up the Start menu (bottom right – Windows logo), then right click on the "Computer" option. On a Linux machine (like the teer machines), you can find this info by typing "more /proc/cpuinfo" at the command line. I recommend looking at both Windows and Linux, if you can. On Windows, you can also check using a program called CPU-Z.
2) How much physical memory do you have? On a Windows machine, you can hit ctrl-alt-delete and then from there open up the Task Manager; in the Task Manager, click on the "Performance" tab. On a Linux machine, you can type "top" at the command line and see much the same information. Once again, I recommend looking at both Windows and Linux.
3) How much disk space do you have? Can you see how much is being used for "swap" (i.e., to hold pages that don't currently fit in physical memory? This is an aspect of "virtual memory", which we'll cover next.

2. Benchmarking memory and cache

1) Download the program "memdance.c" from the course site and compile it with:
   g++ -O3 -o memdance memdance.c
   This program is a simple benchmark to measure the rate of random memory access to a block of memory of a given size for a given duration of a time. Run it without arguments for help.
2) Use this program on the machine of your choice to measure the memory throughput for the following buffer sizes: 1MB, 2MB, 4MB, 8MB, 16MB, 32MB, 64MB, 128MB. A test duration of around 3 seconds per run should be sufficient to get good data.
3) Compare the results to the cache size found in Task 1, especially the lowest level cache (L3 on most modern systems). What do you observe? What's the percentage difference between the fastest and slowest rate? What does this tell you about the importance of cache to software performance?

3. Cache Examples

You have a 64-bit machine with a cache that is 128KB and 2-way set-associative. Blocks are 64B.

1) Sketch this cache. How many frames does it have? How many sets does it have?
2) Divide up the 64-bit address into its three fields, for purposes of accessing this cache. How many block offset bits are there? How many set index bits are there? How many tag bits are there?
3) For a given sequence of requests, can you explain which ones hit and which ones miss? And for the misses, can you classify them as cold, capacity, or conflict misses? For example, consider the address sequence (assume these are all caused by load-byte instructions): 67, 125, 18, 10, 64K+67, 128K+67.
4) What are the first three blocks that map to set 2? What are the first three blocks that map to set 7? Think about this in two ways:
    a. You know that set 0 hold the blocks [0:63], [64K:64K+63], [128K:128K+63], etc.
    b. You know that all blocks that map to a set have the same set index bits. Write out the set index bits and set the block offset bits to zero (to get the address of the 0^{th} byte in the block). If you set the tag bits to zero, you get the 0^{th} block that maps to this set. If you set the tag bits to 0000…..0001, you get the 1^{st} block that maps to this set. Etc.

Hint: for Task 3, recall this slide summarizing the cache arithmetic: