MS108: Computer System 1

Spring 2015

Homework #3

Due: Two Weeks from Assignment

TA: Ran Ye

Email: 叶冉 [happyinglife@sjtu.edu.cn]

Collaboration Policy

These homework sets will be extremely valuable as tools for learning the material and for doing well on the midterm and final. You are required to obey the following rules:
(a) Each student should write out their solution independently and in their own words.
(b) Same applies to programming assignments – you should do your own coding.
Above all, make sure that you understand the solution to these homework problems. They really are assigned to help you understand the material and be prepared for the types of problems on the midterm and final!

Q1. Simple Cache

a) Here is a string of address references given as word addresses: 1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17. Assume a direct mapped cache with 16 one-word blocks that is initially empty, label each reference in the list as a hit or miss and show the final contents of the cache.

b) Using the same reference string, show the hits and misses and final cache contents for a direct mapped cache with four-word blocks and a total size of 16 words.

c) Using the same reference string, show the hits and misses and final cache contents for a two-way set associative cache with one-word blocks and a total size of 16 words. Assume LRU replacement.

d) Using the same reference string, show the hits and misses and final cache contents for a fully associative cache with one-word blocks and a total size of 16 words. Assume LRU replacement.

Q2. Cache Addressing
i. What is the total number of bits (overhead **and** data) required for this particular cache configuration: 1 MB total data, 16-way set associative, 512 Byte blocks. Assume a "write-back" write strategy and a "FIFO" replacement strategy. Assume a 32-bit, byte-addressed architecture.

Q3. Cache Timing
EMAT = Time for a hit + (Miss rate x Miss penalty)
i. Find the EMAT for a machine with a 1-ns clock, a miss penalty of 40 clock cycles, a miss rate of 0.05 misses per instruction, and a cache access time (including hit detection) of 1 clock cycle. Assume that the read and write miss penalties are the same and ignore other write stalls.

ii. Suppose we can improve the miss rate to 0.03 misses per reference by doubling the cache size. This causes the cache access time to increase to 2 clock cycles. Using the EMAT as a metric, determine if this is a good trade-off. Please show your work.

iii. Generally speaking, the CPU cycle time is matched to the cache access time in a pipelined processor. Let us consider two machines that have identical instruction sets and pipeline structure. They differ only in the clock speeds of the processor and the cache structure.

Machine A:
CPU clock cycle time = 1 ns
Cache access time = 1 CPU cycle
Cache miss rate = 5%
Cache miss penalty = 60 CPU cycles

Machine B:
CPU Clock cycle time = 2 ns
Cache access time = 1 CPU cycle
Cache miss rate = 3%
Cache miss penalty = 30 CPU cycles

Both machines have a CPI of 3 without accounting for memory stalls. Both incur 1.45 memory references on an average per instruction.

(a) Which processor has a better EMAT?

(b) Is the EMAT sufficient to declare one machine to be better than the other? Why not?

(c) Which machine is actually better?

Q4. Cache Policy
i. Suppose a fully associative cache has 8 frames, each frame being able to hold one block, and applies the FIFO replacement policy. Present a memory access sequence using block numbers (e.g., 2, 3, 2, 22, 7, 9) in which the miss rate would INCREASE if the size of the cache is increased to 9 frames.

ii. Would the same anomaly occur with a cache using the LRU replacement policy? Please explain the reason.

Q5. SimpleScalar Assignment
This problem gives you a feel for how quantitative evaluations and resource constraints can guide implementation choices in computer systems. In this problem, we will use SimpleScalar's "sim-cache" simulator that allows the user to model a cache memory system. Use the same benchmarks and script as for HW1 – choose 3 of the benchmarks. Hopefully you will only need to make very minor changes to the script to get things running with sim-cache. If you'd like, you can also use "sim-cheetah" which performs multiple cache simulations in parallel.

The quality of our cache organization will be determined by the average access time for that cache:

Average Access Time = Hit Time + Miss Rate * Miss Penalty
Where Miss Penalty = ((Block size in words)/2 + 10) * Hit Time

We will use a hit time of 1ns for a direct-mapped cache and 1.5ns for a set-associative cache.

You are given 80 4Kx8bit static RAMs for the cache, as well as a selection of miscellaneous logic parts. From this "inventory" you must be able to build up the entire cache, including the tags, data, and three control bits per cache line. The address supplied to the cache is 32-bits. Your goal is to find the best cache organization for each of the benchmarks given. You should explore several different block sizes and several different degrees of associativity. (Reasonable degrees of associativity are from direct-mapped through 8-way set associative. Don't explore higher degrees than 8-way). Also, you can leave the TLB parameters unchanged – only look at L1 Dcache and L1 Icache. Assume the caches are write-through and ignore all write stalls.

What to hand in:
You should create a summary of benchmark behavior on each cache configuration. The summary should include:
1) A cache description. This includes the size of the cache, the block size, the associativity. (The “cache size” is its true data capacity, NOT the amount of RAM that was used to build it including tags, etc).
2) A RAM accounting breakdown: This includes the amount of RAM used for the tags, data, and control bits of this organization.
3) The total miss rate, and the read and write miss rates for the benchmarks.
4) The average access time (as given by the formula above) per reference.
Also include an overall summary that gives your conclusions about the applications and cache behavior. Here, you should combine your observations about the different applications to make a recommendation about which cache structure would be best to build, given that the system obviously can’t have a different cache for each application.