CS152 Quiz 3 Answer Guide

Problem Q3.1: Page Size and TLBs

Problem Q3.1.A

Address translation cycles =
*3 arrays * 256 pages per array * (100 + 100 +100) (latency for L1, L2 and L3 PTE)*

*Data access cycles = 3 arrays * 256 pages per array * 4K bytes per page * 100 latency (there is no data cache, memory access is byte-wise)*

[grading scheme: accepted answers that included TLB hit time and multiple cycle access for one PTE]

If all data pages are 1MB, compute the ratio of cycles for address translation to cycles for data access.

*Address translation cycles = 3 arrays * 1 page per array * (100 + 100) (for L1, L2 PTE)*

*Data access cycles = 3 arrays * 1 page per array * 1M bytes per page * 100 latency (there is no data cache, this assumes that memory access is byte-wise)*

Problem Q3.1.B

*Address translation cycles = (256*3 + 3 + 1) * 100*
*(Note that the arrays are contiguous and reuse L1/L2 PTE entries. 256 L3 PTEs per array * 3 arrays, 1 L2 PTE per array * 3 arrays, 1 L1 PTE.)*

*Data access cycles = 3M*100*

[grading scheme: accepted answers that included TLB hit time and partial credits for noting the number of L3, L2 and L1 PTEs needed]

Problem Q3.1.C

No. For the sample program given, all L3 PTEs are used only once.

Problem Q3.1.D

4. (1 for L1 and 3 for L2)

Problem Q3.2: 64-bit Address Spaces 25 Points

Alyssa P. Hacker is unhappy with the large hole in the virtual address space given by Ben's scheme. She decides that a hashed page table is the way to go. Again, the machine has a 64-bit virtual address and 4KB pages. The hardware paging system has only one page table with 64 slots, each containing 8 PTEs. Alyssa decides to use ***X mod 64*** as the hash function to select a slot, where ***X*** is the VPN. The page table resides in memory and Alyssa's design has no TLB, so each PTE read requires one memory access. During a page table lookup, all PTEs in each slot are searched sequentially. If there is a miss in the page table, a trap is raised and a software handler will refill the page table, with each refill requiring 10 memory accesses on average.

Problem Q3.2.A

Alyssa is happy with her new modification and runs a very simple benchmark that repeatedly loops over an array of 2^{21} bytes, reading one byte at a time in sequential address order. On average in the steady state, how many memory accesses are performed for each byte read by the user program? Ignore the memory traffic for instruction fetch, assume that the array starts on a page boundary, and there are no other memory accesses in the user code apart from the single byte memory accesses.

The array of 2^{21} bytes will occupy 2^{9} pages, which is exactly the size of the page table. Therefore, after the initial cold misses, there will not be more page table misses. In the steady state, all page table pages are accessed an equal number of times. Since all pages in a slot are searched sequentially, the average number of memory accesses required to find the address in the page table page is

$(1 + 2 + ... + 8) / 8 = 4.5$

Including the one memory access required for the data, total average latency is 5.5 memory accesses.

Problem Q3.2.B

Alyssa now decides to add a one-entry TLB to the hashed paging system. What should the replacement policy for the TLB be? Circle the most appropriate of the following choices and explain:

a) FIFO
b) LRU
c) Random
d) Doesn't matter, all of the above give the same performance
There is only one TLB entry, so it will always be replaced

Problem Q3.2.C

Given your answer to the previous question, what is now the average number of memory accesses per user byte read for Alyssa's benchmark?

With the TLB, only the first byte read in each page will result in a miss. The average access per TLB miss is still 5.5, and the latency is only 1 for data access. Thus, the average number of memory accesses is

$(1/2^{12}) * 5.5 + (1 - 1/2^{12}) * 1 = 1 + 9 / 2^{13}$

Problem Q3.3: Virtual Memory and Caches 24 Points

Problem Q3.3.A

Alyssa's suggestion solves the homonym problem. If we add a PID as a part of the cache tag, we can ensure that two same virtual addresses from different processes can be differentiated between in the cache, because their PIDs will be different.

Problem Q3.3.B

Putting a PID in the tag of a cache does not solve the synonym problem. This is because the synonym problem already deals with different virtual addresses, which presumably would have different tags in the cache. In fact, those two virtual addresses would usually belong to the different processes, which would have different PIDs.

Problem Q3.3.C

Ben is wrong in thinking that changing the cache to be direct mapped helps in any way. The homonym problem still happens, because same virtual addresses still receive the same tags. The synonym problem still happens because two different virtual addresses still receive different tags.

One way to solve both these problems is to make the cache physically tagged, as described in Lecture 10.