EECS470 Homework 5 Answers

1. a) Will only evict A if either B or C go to the same set.
   P(A≠B)*P(A≠C) = 56.25%
   or
   1 – P(A=B)*P(A≠C) – P(A≠B)*P(A=C) – P(A=B)*P(A=C) = 56.25%
   b) 100% because the cache will never kick A out since it never fills up.
   c) Same as part a except the probability P(A=X) = 1/8 now. 76.5625%
   d) Will evict A if both B and C go to the same set.
   1 – P(A=B)*P(A=C) = 75%
2. a) (2) + 0.05*(10) + 0.05*0.5*(200) = 7.5
   or
   0.95*(2) + 0.05*0.5*(2 + 10) + 0.05*0.5*(2 + 10 + 200) = 7.5
   b) 1 + (1-x)*(2) + (1-x)*0.05*(10) + (1-x)*0.05*0.5*(200) = 7.5
   x = 13.33%
   The L0miss * L1miss is still 5% because the L1 is inclusive of the L0 because every access that missed in the L1 before the L0 was added will still miss in the new setup.
   c) The bandwidth to the L1 cache is doubled because every access is sent to the L1. In the previous setup accesses that missed the L0 cache were the only ones sent to the L1 cache.
3. a) addresses are 16 tag bits, 8 set index bits, 8 block-offset bits. There are 256 sets, * (16 tag bits + 1 valid bit) = 4352 bits. 4096 bits is also ok if you didn't include a valid bit as part of the "tag".
   b) (16 tag bits + 8 valid bits)* 256 sets= 6144 bits. 9 valid bits is acceptable, too. c) i)
   Design A: 1MB / 256B per miss = 4096 misses
   Design B: 1MB / 32B per miss = 32768 misses
   ii) Both designs transfer 1MB
   d) i) Both designs incur 4096 misses (every access is a miss in both designs) ii)
   Design A: 4096 misses * 256B = 1MB
   Design B: 4096 misses * 32B = 128KB
4. a) 2 + 12 * .05 + 250 * .05 * .4 = 7.6
   b) Target access time = 0.85 * 7.6 = 6.46
   2 + 14 * .05 + .05 * x * 250 = 6.46
   x = 30.08%
5. Multiple reasonable answers for these. Here are ours:
   a. **Vector machine** – Matrix multiplication has high DLP. We don't need dynamic scheduling, superscalar pipelines, multithreading capabilities, or cache coherency (which are all potentially big and power hungry) to get good performance.
   b. **Multicore machine** – Many independent packets suggests high TLP. A multithreaded machine might have the problem of threads knocking each other out of the L1 cache. We can share the L2 cache so all processors can access the virus signatures efficiently. (However, if the number of threads is high enough or L2 access time is low enough, fine grained multithreaded machine might do ok)

c. Superscalar machine – Compiling a single file likely does not have much TLP or DLP (for instance, splitting the file in half and assigning to different threads would likely be inefficient, since syntax elements in the second half likely depend on the first). We'll need to rely on traditional ILP techniques to speed this up.

d. Vector machine – Superscalar machines, SMT cores, multicores, and multithreaded machines don't really provide more or less FP throughput by themselves, those all differ in how work is scheduled to those units. Vector machines, on the other hand, can provide more throughput with minimal hardware overhead.

e. SMT machine – We want to dynamically partition all the floating point units to the ray tracing code. The SMT machine is the only one that enables tat.

6. We need to ensure that only the page-offset bits are used for the cache index bits, otherwise we'll get homonym/synonym problems. Number of page offset bits $=\log(4K)=12$, number of block offset bits for the cache $=\log(64)=6$. So, that leaves $12-6=6$ bits for the cache index, or $2^6=64$ sets.

   a/b) For 4-way, this gives a total size of

   64 sets * 4 ways/set * 64 bytes/way = 16 KB

   c) For direct-mapped:

   64sets * 1 way/set * 64 bytes/way = 4 KB

   Note, if you included tag and other overheads as part of the total cache size (normally we don't), then you will need to specify and address size, whether it's write-back or write through, etc.

   d) Hardware:
   - use PIPT caches
   - Ensure cache size less than or equal to page size times associativity

   Software:
   - Don't allow OS to map one physical address to multiple virtual addresses • Restrict page placement such that the index of the virtual address is always equivalent to the index of the physical address

   e) Unless we enforce alignment (which would defeat much of the point of supporting smaller page sizes), then we will need to translate part of the page offset bits as well. Page allocation will also be more complicated.

7. See lecture slides