Lecture: Cache Hierarchies

- Topics: example problems on cache access, cache innovations

Upcoming Schedule:

- HW 6 due: Oct 28 (Wed)
- HW 7 due: Nov 11 (Wed)
- HW 8 due: Nov 18 (Wed)
- HW 9 due: Nov 30 (Mon)

Dec 2\textsuperscript{nd} – last day of class (review session)
Dec 7\textsuperscript{th} – final exam
Problem 2

• Assume a direct-mapped cache with just 4 sets. Assume that block A maps to set 0, B to 1, C to 2, D to 3, E to 0, and so on. For the following access pattern, estimate the hits and misses:

A B B E C C A D B F A E G C G A
Problem 2

• Assume a direct-mapped cache with just 4 sets. Assume that block A maps to set 0, B to 1, C to 2, D to 3, E to 0, and so on. For the following access pattern, estimate the hits and misses:

A B B E C C A D B F A E G C G A
M MH MM H MM HM HMM M M M
Problem 3

• Assume a 2-way set-associative cache with just 2 sets. Assume that block A maps to set 0, B to 1, C to 0, D to 1, E to 0, and so on. For the following access pattern, estimate the hits and misses:

A B B E C C A D B F A E G C G A
Problem 3

- Assume a 2-way set-associative cache with just 2 sets. Assume that block A maps to set 0, B to 1, C to 0, D to 1, E to 0, and so on. For the following access pattern, estimate the hits and misses:

A B B E C C A D B F A E G C G A
M MH M MH MM HM HMM M H M
Problem 5

• 8 KB fully-associative data cache array with 64 byte line sizes, assume a 40-bit address

• How many sets? How many ways?

• How many index bits, offset bits, tag bits?

• How large is the tag array?
Problem 5

- 8 KB fully-associative data cache array with 64 byte line sizes, assume a 40-bit address
- How many sets (1) ? How many ways (128) ?
- How many index bits (0), offset bits (6), tag bits (34) ?
- How large is the tag array (544 bytes) ?

Equations:
Data array size (cache size) = \#sets \times \#ways \times \text{blocksize}
Tag array size = \#sets \times \#ways \times \text{tagsize}
Index bits = \log_2 (\#sets)
Offset bits = \log_2 (\text{blocksize})
Tag bits + index bits + offset bits = address width
Types of Cache Misses

• Compulsory misses: happens the first time a memory word is accessed – the misses for an infinite cache

• Capacity misses: happens because the program touched many other words before re-touching the same word – the misses for a fully-associative cache

• Conflict misses: happens because two words map to the same location in the cache – the misses generated while moving from a fully-associative to a direct-mapped cache

• Sidenote: can a fully-associative cache have more misses than a direct-mapped cache of the same size?
More Cache Basics

- L1 caches are split as instruction and data; L2 and L3 are unified

- The L1/L2 hierarchy can be inclusive, exclusive, or non-inclusive

- On a write, you can do write-allocate or write-no-allocate

- On a write, you can do writeback or write-through; write-back reduces traffic, write-through simplifies coherence

- Reads get higher priority; writes are usually buffered

- L1 does parallel tag/data access; L2/L3 does serial tag/data
Tolerating Miss Penalty

• Out of order execution: can do other useful work while waiting for the miss – can have multiple cache misses -- cache controller has to keep track of multiple outstanding misses (non-blocking cache)

• Hardware and software prefetching into prefetch buffers – aggressive prefetching can increase contention for buses
Reducing Miss Rate

- Large block size – reduces compulsory misses, reduces miss penalty in case of spatial locality – increases traffic between different levels, space waste, and conflict misses

- Large cache – reduces capacity/conflict misses – access time penalty

- High associativity – reduces conflict misses – rule of thumb: 2-way cache of capacity N/2 has the same miss rate as 1-way cache of capacity N – more energy
Techniques to Reduce Cache Misses

- Victim caches
- Better replacement policies – pseudo-LRU, NRU, DRRIP
  -- insertion, promotion, victim selection
- Prefetching, cache compression
Victim Caches

• A direct-mapped cache suffers from misses because multiple pieces of data map to the same location

• The processor often tries to access data that it recently discarded – all discards are placed in a small victim cache (4 or 8 entries) – the victim cache is checked before going to L2

• Can be viewed as additional associativity for a few sets that tend to have the most conflicts
Replacement Policies

- Pseudo-LRU: maintain a tree and keep track of which side of the tree was touched more recently; simple bit ops

- NRU: every block in a set has a bit; the bit is made zero when the block is touched; if all are zero, make all one; a block with bit set to 1 is evicted
Prefetching

- Hardware prefetching can be employed for any of the cache levels

- It can introduce cache pollution – prefetched data is often placed in a separate prefetch buffer to avoid pollution – this buffer must be looked up in parallel with the cache access

- Aggressive prefetching increases “coverage”, but leads to a reduction in “accuracy” → wasted memory bandwidth

- Prefetches must be timely: they must be issued sufficiently in advance to hide the latency, but not too early (to avoid pollution and eviction before use)
Stream Buffers

• Simplest form of prefetch: on every miss, bring in multiple cache lines

• When you read the top of the queue, bring in the next line
Stride-Based Prefetching

- For each load, keep track of the last address accessed by the load and a possibly consistent stride

- FSM detects consistent stride and issues prefetches