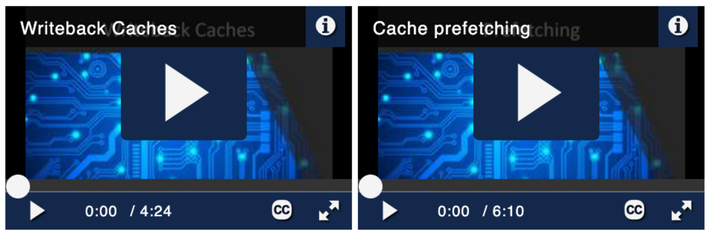
PRE20.1. 20 Text and Videos

Videos



Video credits: Geoffrey Herman, Text credits: Geoffrey Herman

The Big Picture

Cache Writing

Ideally, we don't just want to read from our caches/memories, but we want to also write the results of our calculations back to caches and also later to the main memory. This poses a problem if we take either of 2 naive solutions: 1) if we write back to both the cache and memory, then every store instruction becomes a cache miss, degrading performance or 2) if we write back only to the cache to get cache hits, then our cache will get out of sync with main memory. To solve this problem, we generally use "write-back" caches that add one more piece of meta-data - a dirty bit - that keep track of when a cache block gets out of sync with main memory.

When a cache block is brought into the cache, it is set to be valid (valid bit = 1) and clean (dirty bit = 0). The dirty bit will stay 0 until a store is performed in the cache block at which point the dirty bit is set to 1. The dirty bit will stay 1 until the cache block is replaced.

If the dirty bit is 0 when the cache block is replaced, no data is copied back to main memory.

If the dirty bit is 1 when the cache block is replaced, then we copy the contents of the cache block back to main memory.

Pre-fetching

One reason that caches are effective is that we can reasonably predict that if an address in memory is accessed that it will either be 1) accessed again soon (i.e., temporal locality) or 2) nearby addresses will be accessed as well soon (i.e., spatial locality). Pre-fetching takes the second, spatial-locality-based, prediction a step further by predicting that cache blocks that are a bit further away will also like be accessed soon. For example, if we are traversing a very long array, we will be able to detect that pattern in our code (either the programmer or the hardware can do make detect this pattern). We (or the hardware) can use pre-fetching to indicate to our cache/main memory that we expect that we will use data from future addresses based on a predictable pattern in the code and start getting that data copied to the cache before we actually access it with our program.

Hardware can pretty much only pre-fetch addresses that are part of an array traversal or some other very predictable data structure.

Humans can help hardware pre-fetching by enabling pre-fetching of unpredictable addresses such as those in linked lists or other linked data structures.

Tips for solving code to hits/misses problems

- Determine the dimensions of the cache
- Trace the C code to identify the order of addresses it accesses
- Map the sequence of accesses to their, tag, set index, and block offset (note: once you determine the set index and block offset of an element of an array, A[0] for example, it will always have the same index and offset for each iteration)
- Keep track of the number of hits/misses for each iteration of a loop
- If there are multiple traversals over the same data structure, pay attention to what data is left in the cache after a traversal
- After tracing several iterations, find a pattern and extrapolate

Sorry we don't have more text for this section ready yet.

Mark as read

(a) I've read this!

Select all possible options that apply.

Save & Grade *Unlimited attempts* Save only

PRE 20

Assessment overview

Total points: 60/60

Score: 100%

Question PRE20.1

Value: 1

Total points: — /1

Auto-graded question

Previous question Next question

Personal Notes

No attached notes

Attach a file

Add text note