Storage 2: From HW Caching to SW caching

• Learning Objectives
  • Translate what we learned about hardware caches to software.
  • Evaluate the efficacy of a cache.
  • Define:
    • Cache block
    • Cache slot
    • Cache hit/miss rate
    • Replacement policy
The Memory Hierarchy

- Registers
- L1 Cache
- L2 Cache
- L3 Cache
- L4 Cache
- Main Memory

Bigger (More Expensive)
The Memory Hierarchy

- Registers
- L1 Cache
- L2 Cache
- L3 Cache
- L4 Cache
- Main Memory
- Flash Drive

Bigger, Faster (More Expensive)
The Memory Hierarchy

- Registers
- L1 Cache
- L2 Cache
- L3 Cache
- L4 Cache
- Main Memory
- Flash Drive
- Disk Drive

Bigger (Faster (More Expensive))
The Memory Hierarchy -- Speed

- Registers: .3 ns
- L1 Cache: 1.1 ns
- L2 Cache: 3.3 ns
- L3 Cache: 12.8 ns
- L4 Cache: 42.4 ns
- Main Memory: 62.9 ns
- Solid State (flash) Drive: ~.1 ms
- Disk Drive: ~3 ms

Bigger: 2-5 TB, ~200 GB – 1 TB, 32 GB, 128 MB, 8 MB, 1 MB (256KB/core), 256 KB (64 KB/core), ~1 KB (~100 b/core)

Faster: ~.1 ms, ~3 ms, 62.9 ns, 42.4 ns, 12.8 ns, 3.3 ns, 1.1 ns, .3 ns
Screen Capture

- w01_sync
- w02_syscall
- w03_stdio
Where oh were are the SW caches?

Let’s put a cache here to hide how slow persistent storage is!
Where oh were are the SW caches?

- Registers
- L1 Cache
- L2 Cache
- L3 Cache
- Main Memory
- Flash Drive
- Disk Drive

In fact, let’s put TWO caches there!

OS cache (shared among all processes)

Per-process caches
An Abstract Cache

Application

Read data

Write data

Cache

Data Source
Reading from the cache: HIT
Reading from the cache: MISS

1. Please read foo
2. Do I have that item?

No.

Data Source

Cache

Application

6. Here you go.
Decisions: Servicing misses (blocksize)

Application

1. Please read foo
2. Do I have that item?
3. No!
4. May I have foo?

Cache

5. Here you go.
6. Here you go.

Data Source

How much data should I return?
How much data should I return?

• Most storage devices have a native size for data access and/or transmission, e.g., disk block (4 KB).
• Recall: HW caches also have a unit they use to transfer data to/from the cache:
  • Cache lines: typically 64 or 128 bytes.
• **Block size**: the unit in which data is stored in a cache.
  • HW caches: 64 bytes
  • File system caches: 4+ KB
  • Object caches: size of the object
Decisions: A full cache (eviction)

1. Please read foo
2. Do I have that item?
3. No!
4. May I have foo?
5. Here you go.
6. Here you go.

What do I do when I run out of space?
What do I do when I fill up?

• A cache has a limited capacity.
• At some point, the application will fill the cache and request another item.
• Caching the new item requires **evicting** some other item.
• What item do I evict?
  • We need an **eviction policy**
  • The available decisions here vary between hardware and software.
Eviction in Hardware

- A cache is comprised of some number of slots (locations in the cache, each of which can hold a cache line or cache block).
- The hardware often limits the number of possible slots in which an item can be placed.
- Call the number of slots in which a particular item can be placed \( A \). Let \( N \) be the total number of slots in the cache.
  - \( A = 1 \): Direct mapped: each object can live in exactly one slot in the cache, so you have no choice but to evict the item in that slot.
  - \( A > 1 \), \( A \ll N \): A-way set associative: an object can live in one of \( A \) slots; \( A \) is typically 2, 4, or 8. On eviction, choose randomly from among the \( A \) slots.
  - \( A = N \): Fully associative: an object can live in any slot.
Eviction in Software

- In software, we almost always have a fully associative cache.
- In a perfect world, we’d like to evict the item that is least valuable.
- In the real world, we don’t know what that item is.
- Practically all software caches try to approximate this ideal.
  - LRU: Least-recently-used – find the item that has been unused the longest and get rid of that.
  - FIFO: First-in-first-out – find the item that has been in the cache the longest.
  - LFU: Least-frequently-used – find the item that has been used less frequently and get rid of that.
  - Clock: Used in virtual memory systems to approximate LRU, take CS161 for details.
  - Something tuned to known access patterns.
Evaluating a Cache: Hit Ratio

- Hits are much better than misses!
- We measure the efficiency of a cache in terms of its cache hit rate:
  - \( \frac{\text{# cache hits}}{\text{# cache accesses}} \)
  - \( \frac{\text{# cache hits}}{\text{# cache hits} + \text{# cache misses}} \)
- Example:
  - I access my cache 1000 times and have 400 hits.
    - My cache hit rate is \( \frac{400}{1000} = 40\% \)
- Good performance requires high cache hit rates.
More than one way to get a hit …

• If you touch the same item more than once, you get a hit, but there is another way to get a hit.

• Think about the fact that your cache is organized in blocks…

• Consider this:
  • Let’s say you are accessing an array of 4-byte integers.
  • A cache line is 64 bytes.

• Here is the question:
  • Let’s say that you have 160 items in the array and you’ve never accessed it before, how many cache misses will you take?
Fun With Eviction

• Consider the following set of references to cache blocks:

1 2 3 1 1 2 4 5 2 1 4

• Live People Demo!