A...B...Caches*

Vishal Soni

Caches: Their place in the scheme of things

- Memory Hierarchy: Caches come in between the CPU and lower order memory (like RAM). Hard Disks are the the bottom of this hierarchy. What is at the top?
- Cost vs. access speed: As you move up the hierarchy, cost increases and access time (a.k.a access latency) decreases.

Types of caches

Direct Mapped If each block has only one place it can appear in the cache, the cache is said to be direct mapped.

To give you an idea, the mapping is usually (not necessarily always)

(Block address) mod (number of blocks in cache)

Fully associative If a block can be placed anywhere in cache, it is said to be fully associative.

Set associative If a block can be placed in a restricted set of places in the cache, the cache is set associative. A set is a group of blocks in the cache. A block is first mapped onto a set, and then the block can be placed anywhere within that set.

To give you an idea, the set is usually (not necessarily always) chosen

(Block address) mod (number of sets in cache)

If there are n blocks in a set, the cache is said to be n-way set associative.

Finding a block in a cache

An address in a (set associative) cache has three portions in its address

- Index: Used to find the set in the cache.
- Tag: Compared against the tag field in the cache object for a hit.
- Block offset: Used to find the data within the block.

Cache objects have a *valid bit* to indicate whether the cache entry contains valid information.
Size of index vs. size of tag field

*stolen without abandon from various sources

Replacement/Eviction in the event of a miss

When a cache miss occurs, a **block** must be selected to be replaced with the desired data. A common technique for deciding which block to replace is LRU, i.e. pick the block that has been unused for the longest time. LRU exploits a concept known as **temporal locality**[1]; recently used blocks are more likely to be used again.

Dealing with writes

Two basic options for writing to cache:

- Write Through: In the event of a write, the data is written to both the cache block and the block in the memory
- Write Back: In the event of a write, the data is written only to cache block. The modified cache block is written to memory only when it is replaced.
- What is a *dirty bit*?
- Advantages / disadv. of these schemes? Write back is usually faster than write through.

  In write through, read misses never result in writes to memory (what does this mean?). Write through is also easier to implement.

Useful stuff, mmmm'kay?

The size of the index depends on the cache size, block size, and the set associativity.

$$2^{index} = \frac{cache\ size}{block\ size \times set\ associativity}$$

It helps to express things as powers of 2...

$$\begin{aligned} 2^{index} &= \frac{cache\ size}{block\ size \times set\ associativity} \\ &\equiv \frac{2^i}{2^j \times 2^k} \\ &= 2^{i-(j+k)} \\ \Rightarrow index &= i-(j+k) \end{aligned}$$

Performance

One way to compare performance of **memory hierarchies** is to compare their *average memory access times*

$$\text{Average memory access time} = \text{Hit time} + (\text{Miss rate} \times \text{Miss penalty})$$

[1] can be thought of as *nearness in time*