Lecture 7: Caches

Hossam A. H. Fahmy

Cairo University, Faculty of Engineering

Overview

Introduction to caches

- Why do we have memories in computers?
- What is the hierarchy of storage elements (latency, bandwidth, capacity, and price)?
- Why is there a difference in speed?

Imagine yourself in a large library with many books. You want to read a number of sections from a few books. What are you going to do?

Why do we have caches?

On the average, our goal is to give the processor the illusion of a *large* memory system with a *short* effective access time.

```
for(i=0;i<n;i++)
    sumsq  = sumsq + x[i]*x[i] + y[i]*y[i];
```

The basic principles of locality:

1. Spatial locality.
    - Sequential access.
2. Temporal locality.

Some definitions

When the processor fetches a piece of information it might be an instruction or a data value. Hence,

- we may use a unified (integrated) cache for both or
- we may use a split I$ and D$.

We may also have multiple levels of caches. The processor tries first to find the information in the nearest (highest) level of the hierarchy.

- The information request may *hit* in the cache and the needed word reaches the processor after the *hit time* or
- it may *miss* in the cache and is retrieved from the lower level of the hierarchy after an additional *miss penalty*. (*miss time* = hit time + miss penalty)

How much time are we loosing on misses?

- Each instruction accesses the memory for its own fetch.
- It may also access the memory for data.

$$\Rightarrow \text{memory accesses per instruction} \geq 1.$$

Each instruction may hit or miss. If we profile our applications we get the *hit rate* and *miss rate* and calculate

$$\begin{aligned} AMAT &= \text{hit time} \times \text{hit rate} + \text{miss time} \times \text{miss rate} \\ &= \text{hit time} + \text{miss rate} \times \text{miss penalty}. \end{aligned}$$

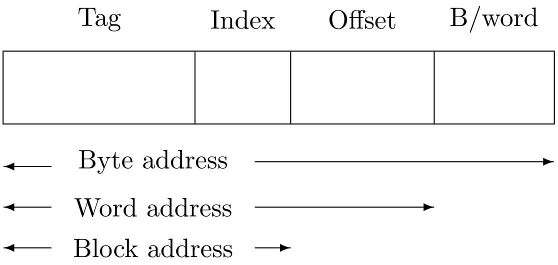
Implementation

To understand how caches work, let us ask a few fundamental questions. Here are the first two.

1. Where is the block placed in the cache?
   - Simplest is *Index = (Block address)mod (# blocks in cache)*. This is called direct mapping.
2. Is the block available (hit) in the cache?
   - Each block in the cache is associated with a tag (and a valid bit). If the requested block has the same index but a different tag it is a miss.

Let us think about a cache with 8 blocks, each one word, that is initially empty and the references: 22, 26, 22, 26, 16, 4, 16, and 18 to words in the memory.

Address



Let us try to find the size of a cache with 9 bits index in a machine having 32 bits for its addresses assuming that the word is four bytes and the block is eight words. *Why eight words in a block?*

Back to fundamentals

Here is another fundamental question

3. In a miss and a need to replace a block, which one shall I choose?
    - Trivial for direct mapping.
    - Random, Least recently used, or FIFO for other mapping techniques that we will study later. *Why do we need other mapping techniques?*

Write policies

The last fundamental question is

4. what happens on write?

Write through: Write to the lower level as well. May slow things down. ⇒ use a write buffer.

Write back: (or Copy back) write only when the block is replaced. ⇒ minimize the traffic by indicating if the block is *dirty*.

What about a miss at the time of writing (remember the case of multiple words per block)?

Write through	Copy back
Write allocate	$\boxed{\text{Write allocate}}$
$\boxed{\text{No-write allocate}}$	No-write allocate

Now what

A designer seeks to reduce

- the miss penalty,
- the miss rate, and
- the hit time.

We must balance that with the rest of the hierarchy as well.