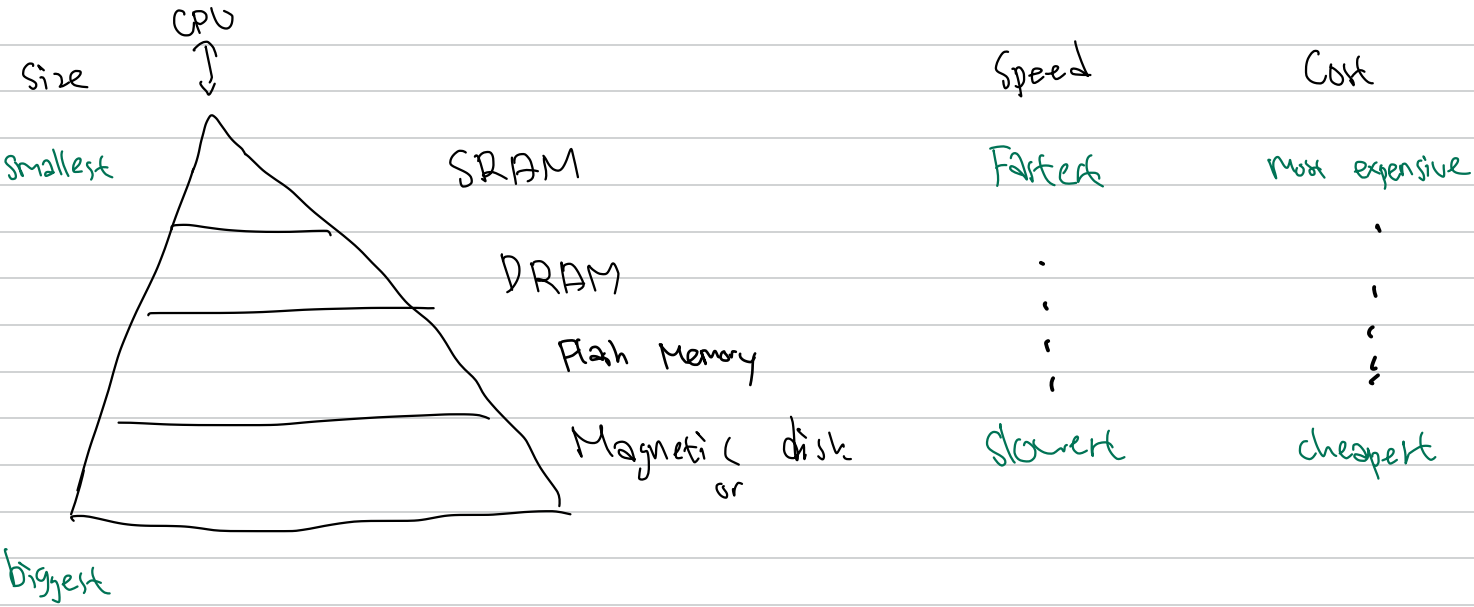
Name: 洪 理川
ID : B113040056

CH05 : Large and fast : Exploiting Memory Hierarchy

Memory Hierarchy 5.1 ~ 5.5



SRAM - Static Random Access Memory

- Used 6~8 transistor → 1 bit storage, these bit packed in rows & column creating SRAM
- Doesn't need to refresh
- used in cache
- Access time : 0.5 ~ 2.5 ns
- Price per GiB : $500 ~ $1000

DRAM - Dynamic Random Access Memory

- Used 1 transistor → 1 bit storage → stored capacitor charge
- Used in Main Memory
- Need to be refresh → that's why called dynamic
- Access time : 50 ~ 70 ns
- Price per GiB : $3 ~ $6

Flash Memory

↳ a type of electrically erasable programmable read-only memory

Disadvantages : can wear out after too much solution
Solution : remapping blocks

SDRAM - Synchronous DRAM

↳ DRAM + clocks → eliminates the time memory and processor sync

Disk Memory

↳ A lot of Rotating Magnetic Hard Disk, the disk is coated with magnetised metal grains. Grains grouped to form a bit.

It uses movable arm with small electromagnetic coil to read/write data.

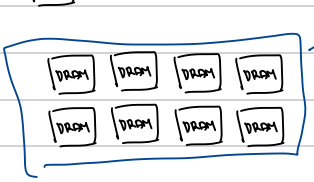
DDR SDRAM

↳ Double Data Rate
transfering data on both rising and falling edge of clock

Ex: DDR4 - 3600 DRAM

- DDR 4 generation
- = 3600 million transfer / second
- = 1800 - MHz clock

let say [DRAM] is a DRAM



→ Create A DIMM
Dual inline memory modules

Cache

↳ Initially it means the memory between processor and main memory. Nevertheless, It now means any storage that takes advantages of locality

① Assigning Cache with Direct-Mapped Cache (DMC)

DMC → We map the cache with the address of the word in memory. Because cache slot is limited we use below formula

(word address) mod (# cache slot)

Ex: cache with 4 slots

Memory Address	Saved in ? cache slot
1 mod 4	1
2 mod 4	2
3 mod 4	3
4 mod 4	4
5 mod 4	1

↑ will collide with slot 1.

② Accessing a Cache

Accessing cache will be a problem when the memory references is larger than the blocks of cache.

If 2 Memory reference in the cache slots, the first references will miss. This behaviours uses temporal locality → recently references word will replace less references word

11010_{two} ← Data address in Memory

- tag → to identify the memory address
- cache index → to let cache select the cache block.

In 32-bit address with Direct-mapped cache

Size tag = $32 - (n + m + 2)$

bits in DMC = $2^n \times (\text{block size} + \text{tag size} + \text{valid field size})$

n = cache size in 2^n

m = Block size in 2^m words = 2^{m+2} bytes

Handling Cache Misses

Handle by processor control unit + controller initiates the memory access.

↓

Stall processor, freezing processor temporary and user-registers

↓

Wait for the Memory / allow some instructions executed.

Handles Writes

normal writes

↓

inconsistent data → data in memory differ from data in cache

Write-through (write into cache & memory in the same time)

↓

Consistent data → data in memory same as data in cache

But very slow

↓

Solution1: Write buffer

by creating buffer to let be written to it. After writting to it, the processor can continue execution.

- Buffer free when write into main memory complete.
- When Buffer full, the processor stall

Solution 2: Write-back

In the data-write hit, only update the block in cache.

- More complex the write-through

Cache Performance

2 technique : Reducing the miss rate by reduce probability 2 same cache location be used.
Reducing the time to copy data from main memory to cache / miss penalty by adding another cache layer.

Before that, we have know how cache performance is calculated:

Only calculate miss time

Cost of cache related to CPU clock cycles times

(Total CPU cycles + Total Memory cycles) . Clock cycle time

Primaly come from cache miss

Total Memory cycles / memory-stall clock = Read-stall cycles + Write-stall cycles

$$\text{Read-stall cycles} = \frac{\text{Reads}}{\text{program}} \cdot \text{Read miss rate} \cdot \text{Read miss penalty}$$

$$\text{Write-stall cycles} = \left(\frac{\text{Writes}}{\text{program}} \cdot \text{Write miss rate} \cdot \text{Write miss penalty} \right) + \text{Write buffer stalls}$$

↳ ignored because too small

Alternatively usually we use AMAT to calculate hit time → time for data sent from cache to processor.

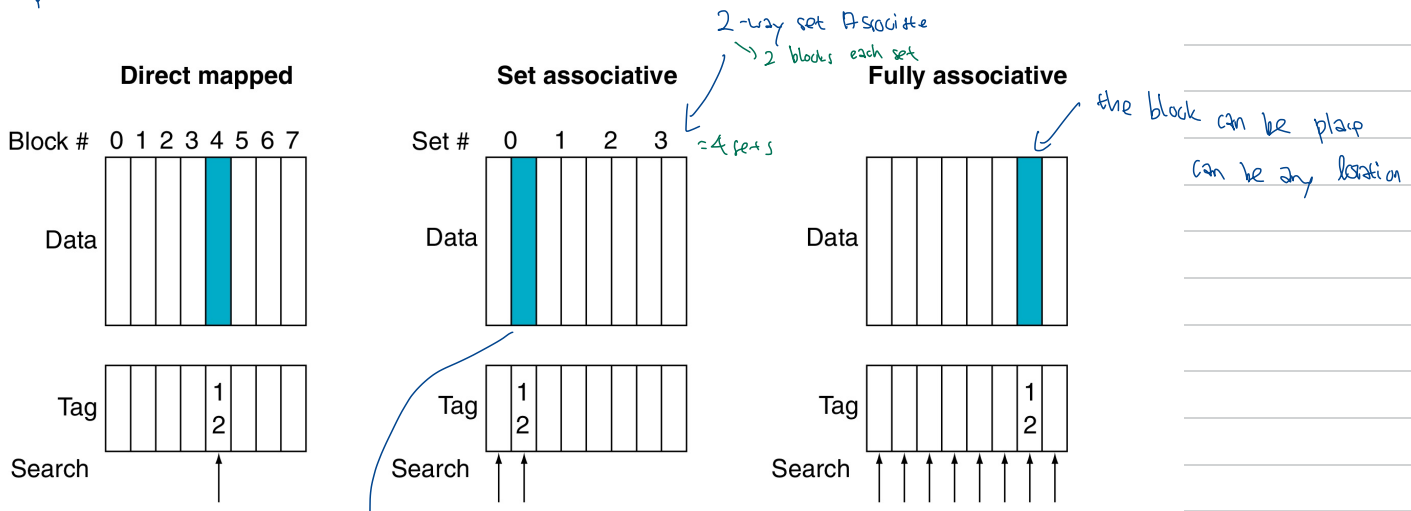
AMAT (Average memory access time)

↳ Time for a hit + Miss rate . Miss penalty.

Technique 1: Changing the block Placement Scheme

Until now, we only know DMC (Direct-mapped cache). There are 2 other schemes : Set associative & Fully associative.

Example : cache with 8 blocks



2-way set Associate → 2 blocks each set

=4 sets

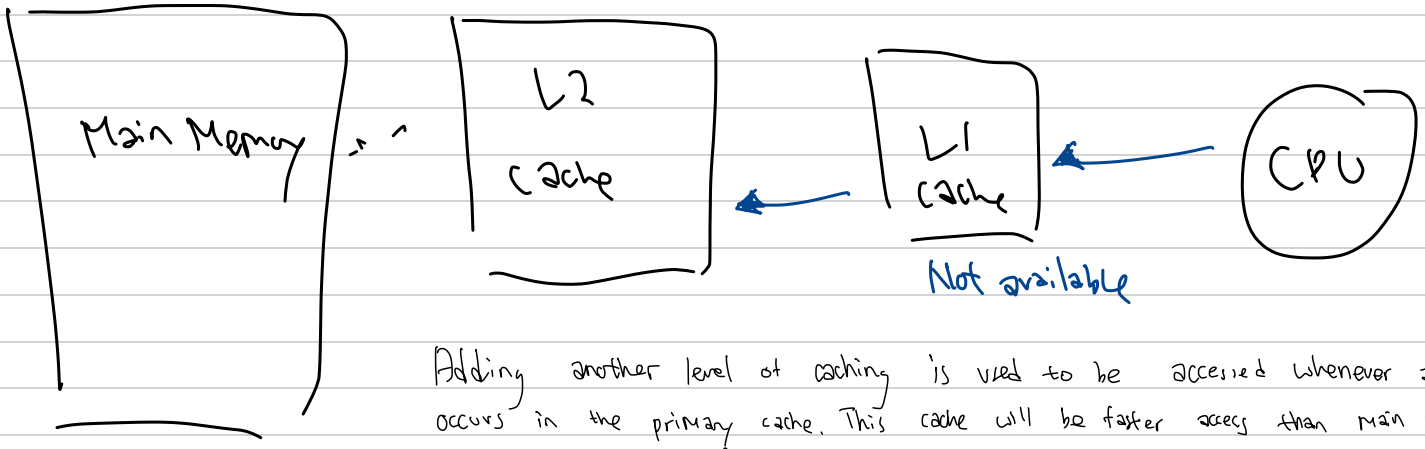
the block can be place can be any location

↳ can be place in any block but usually we use least used for replacement

Advantages: Reduce miss rate → As if DMC be access more than once, the cache will be missed, but in set Associative / fully associative we can accept more than 1 until n-way of the same cache location.

Disadvantage : Extra in hardware and hit time because need to search in that set

Techniques 2: Adding Another Layer of cache



Adding another level of caching is used to be accessed whenever a miss occurs in the primary cache. This cache will be faster access than main memory. To calculate the performance of this technique, we need to have the access time of the cache & memory.

Calculate the miss penalty ⟶ calculate the Total CPI ⟶ then compare.

Dependable of Memory

↳ ↑ Dependable = ↑ Availability = ↓ failure.

Availability can be said to be a measure of the service is normal/ready in respect of failed

↳ can be calculated by $\frac{MTTF}{(MTTF + MTTR)}$

MTTF - Mean time to failure
MTTR - Mean time to repair

How to Improve Availability

Note: MTBF (Mean time between failures) usually in hard disk is calculated by MTTF + MTTR

1. Shrink MTTR - tools for fault detection, diagnosis and repair

2. Improve MTTF - 1. Prevent fault occurence by construction.
   2. Fault Tolerance by using redundancy
   3. Fault prediction to replace before it fails

Availability Standard

Availability	Downtime / Year	Downtime / Month	Downtime / Week	Downtime / Day
99.999%	5.256 Minutes	0.438 Minutes	0.101 Minutes	0.014 Minutes
99.995%	26.28 Minutes	2.19 Minutes	0.505 Minutes	0.072 Minutes
99.990%	52.56 Minutes	4.38 Minutes	1.011 Minutes	0.144 Minutes
99.950%	4.38 Hours	21.9 Minutes	5.054 Minutes	0.72 Minutes
99.900%	8.76 Hours	43.8 Minutes	10.108 Minutes	1.44 Minutes
99.500%	43.8 Hours	3.65 Hours	50.538 Minutes	7.2 Minutes
99.250%	65.7 Hours	5.475 Hours	75.808 Minutes	10.8 Minutes
99.000%	87.6 Hours	7.3 Hours	101.077 Minutes	14.4 Minutes

It's called "nines of availability".

source: AWS

Hamming Code

↳ error detection code that can fixed the errors when data is transmited, When using this hamming code, it will increase the redudant bits to help it self fixed the errors bits.