250P: Computer Systems Architecture

Lecture 10: Caches

Anton Burtsev
November, 2019
The Cache Hierarchy
Accessing the Cache

Direct-mapped cache: each address maps to a unique address

8-byte words

Sets

Data array

Byte address

Offset

101000

8 words: 3 index bits
The Tag Array

Direct-mapped cache: each address maps to a unique address
Increasing Line Size

A large cache line size → smaller tag array, fewer misses because of spatial locality

Byte address

Tag

Offset

10100000

Tag array

Data array

32-byte cache line size or block size
Associativity

Set associativity → fewer conflicts; wasted power because multiple data and tags are read

1. Tag array
2. Compare
3. Tag
4. Byte address
5. 10100000

Way-1

Way-2

Data array
Example

• 32 KB 4-way set-associative data cache array with 32 byte line sizes

• How many sets?

• How many index bits, offset bits, tag bits?

• How large is the tag array?
Types of Cache Misses

• Compulsory misses: happens the first time a memory word is accessed – the misses for an infinite cache

• Capacity misses: happens because the program touched many other words before re-touching the same word – the misses for a fully-associative cache

• Conflict misses: happens because two words map to the same location in the cache – the misses generated while moving from a fully-associative to a direct-mapped cache

• Sidenote: can a fully-associative cache have more misses than a direct-mapped cache of the same size?
Reducing Miss Rate

• Large block size – reduces compulsory misses, reduces miss penalty in case of spatial locality – increases traffic between different levels, space waste, and conflict misses

• Large cache – reduces capacity/conflict misses – access time penalty

• High associativity – reduces conflict misses – rule of thumb: 2-way cache of capacity N/2 has the same miss rate as 1-way cache of capacity N – more energy
More Cache Basics

• L1 caches are split as instruction and data; L2 and L3 are unified

• The L1/L2 hierarchy can be inclusive, exclusive, or non-inclusive

• On a write, you can do write-allocate or write-no-allocate

• On a write, you can do writeback or write-through; write-back reduces traffic, write-through simplifies coherence

• Reads get higher priority; writes are usually buffered

• L1 does parallel tag/data access; L2/L3 does serial tag/data
Problem 1

- Memory access time: Assume a program that has cache access times of 1-cyc (L1), 10-cyc (L2), 30-cyc (L3), and 300-cyc (memory), and MPKIs of 20 (L1), 10 (L2), and 5 (L3). Should you get rid of the L3?
Problem 1

- Memory access time: Assume a program that has cache access times of 1-cyc (L1), 10-cyc (L2), 30-cyc (L3), and 300-cyc (memory), and MPKIs of 20 (L1), 10 (L2), and 5 (L3). Should you get rid of the L3?

With L3: $1000 + 10 \times 20 + 30 \times 10 + 300 \times 5 = 3000$
Without L3: $1000 + 10 \times 20 + 10 \times 300 = 4200$
Techniques to Reduce Cache Misses

- Victim caches
- Better replacement policies – pseudo-LRU, NRU, DRRIP
- Cache compression
Victim Caches

• A direct-mapped cache suffers from misses because multiple pieces of data map to the same location

• The processor often tries to access data that it recently discarded – all discards are placed in a small victim cache (4 or 8 entries) – the victim cache is checked before going to L2

• Can be viewed as additional associativity for a few sets that tend to have the most conflicts
Thank you!