Lecture 13: Cache Hierarchies

• Today: cache access basics and innovations (Sections 5.1-5.2)
The Cache Hierarchy

Core  →  L1  →  L2  →  L3  →  Off-chip memory
Accessing the Cache

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

8 words: 3 index bits

Byte address

Offset

Data array

Sets

8-byte words
The Tag Array

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

- Byte address
- Tag
- Compare
- Tag array
- Data array
- 8-byte words
- 101000

Tag array

Data array
Increasing Line Size

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

32-byte cache line size or block size

Tag array

Data array

10100000

Tag

Offset

Byte address
Associativity

Set associativity $\rightarrow$ fewer conflicts; wasted power because multiple data and tags are read.

Tag array

Tag

Byte address

10100000

Compare

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?
What Influences Cache Misses?

<table>
<thead>
<tr>
<th></th>
<th>Compulsory</th>
<th>Capacity</th>
<th>Conflict</th>
</tr>
</thead>
<tbody>
<tr>
<td>Increasing cache capacity</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Increasing number of sets</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Increasing block size</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Increasing associativity</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Reducing Miss Rate

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

• Large caches – 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

• Way prediction – by predicting the way, the access time is effectively like a direct-mapped cache – can also reduce power consumption
Cache Misses

• On a write miss, you may either choose to bring the block into the cache (write-allocate) or not (write-no-allocate)

• On a read miss, you always bring the block in (spatial and temporal locality) – but which block do you replace?
  ➢ no choice for a direct-mapped cache
  ➢ randomly pick one of the ways to replace
  ➢ replace the way that was least-recently used (LRU)
  ➢ FIFO replacement (round-robin)
Writes

• When you write into a block, do you also update the copy in L2?
  ➢ write-through: every write to L1 → write to L2
  ➢ write-back: mark the block as dirty, when the block gets replaced from L1, write it to L2

• Writeback coalesces multiple writes to an L1 block into one L2 write

• Writethrough simplifies coherency protocols in a multiprocessor system as the L2 always has a current copy of data
Reducing Cache Miss Penalty

- Multi-level caches
- Critical word first
- Priority for reads
- Victim caches
Multi-Level Caches

• The L2 and L3 have properties that are different from L1
  ➢ access time is not as critical for L2 as it is for L1 (every load/store/instruction accesses the L1)
  ➢ the L2 is much larger and can consume more power per access

• Hence, they can adopt alternative design choices
  ▪ serial tag and data access
  ▪ high associativity
Read/Write Priority

• For writeback/thru caches, writes to lower levels are placed in write buffers

• When we have a read miss, we must look up the write buffer before checking the lower level

• When we have a write miss, the write can merge with another entry in the write buffer or it creates a new entry

• Reads are more urgent than writes (probability of an instr waiting for the result of a read is 100%, while probability of an instr waiting for the result of a write is much smaller) – hence, reads get priority unless the write buffer is full
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
Tolerating Miss Penalty

• Out of order execution: can do other useful work while waiting for the miss – can have multiple cache misses -- cache controller has to keep track of multiple outstanding misses (non-blocking cache)

• Hardware and software prefetching into prefetch buffers – aggressive prefetching can increase contention for buses
Title

• Bullet