Practical Caches
COMP 252 - Lecture 3

Antoniu Pop

antoniu.pop@manchester.ac.uk

7 February 2018
Previous Lecture: How/why caches work

- **Locality**
  - Temporal & Spatial
  - What parameters have an impact on locality
  - Spatial: cache line size (how much data is stored in each entry)
  - Temporal: Replacement policy

- **Associativity**
  - Fully associative: good for temporal locality / expensive
  - Direct mapped: bad for temporal locality / cheap
  - Set associative: compromise

- **Cache Replacement Policy**
  - LRU, cyclic, random

- **Read vs. Write behaviour**
  - write through / back
  - write allocate / around
Today’s Lecture – Learning Objectives

To understand

- Additional Control Bits in Cache Lines
- Cache Line Size Tradeoffs
- Separate I&D caches
- Multiple Level Caches
Cache Control Bits

- We have, so far, ignored detail of cache initialisation.

- At some time it must start empty. We need a valid bit for each entry to indicate meaningful data.

- We also need a “dirty” bit if we are using “Write Back” rather than “Write Through.”
Exploiting Spatial Locality

▶ Storing and comparing the address or tag (part of address) is expensive

▶ So far we have assumed that each address relates to a single data item (byte or word)

▶ We can use a wider cache “line” and store more data per address/tag

▶ Spatial locality suggests we will make use of it (e.g. series of instructions)
Direct Mapped Cache – 2 words per line

- Tag
- Index
- Address
- Tag RAM
- Compare
- Multiplexer
- Word 0
- Word 1
- hit / miss
- data
Multiple Word Line Size

- Now bottom bits of address are used to select which word
- Can be used with fully or set associative as well
- Typical line size 16, 32 or 64 bytes (most common)
  - 4, 8 or 16 32-bit words
  - today often 64 bytes (8 64-bit values) in 64 bit architectures
- Transfer from RAM in “blocks”, usually equal to line size
  - use burst mode memory access
The Effect of Line Size

- **Spatial locality**: if we access data, then data close by is likely to be accessed as well

- So a larger line size means we get that nearby data in the cache and avoid misses

- But if line size is too big
  - Data may not be used
  - Displaces other possibly useful data
  - Larger RAM accesses take longer
Impact of Line Size – typical characteristic

![Graph showing the impact of line size on miss rate. The x-axis represents line size in words, and the y-axis represents miss rate in percent. The graph shows a trough in miss rate around a line size of 8 words.]
Separate Instruction & Data (I&D) Caches

- Instruction fetch every instruction
- Data fetch every 3 instructions
- Usually working in separate address areas
- Access patterns different
  - Instructions accessed in serial sections
  - Can use lower associativity
- Better utilization - use separate caches
- Called “Harvard” architecture
Split Level 1 (L1) Caches
Bigger caches have lower miss rates

As chips get bigger we could build bigger caches to perform better

But bigger caches always run slower

L1 cache needs to run at processor speed

Instead put another cache (Level 2) between L1 and RAM
Multiple Level Caches (2)

- CPU
- Registers
- L1 Inst Cache
- L1 Data Cache
- L2
- RAM Memory

Fetch
Data
On-chip
Multiple Level Caches (3)

- L2 cache is typically $16 \times$ bigger than L1
- L2 cache is typically $4 \times$ slower than L1
  - But still $10 \times$ faster than RAM!
- If only 1 in 50 accesses miss in L1 and similar in L2
  - Only have to cover very small number of RAM accesses
- Not quite that easy but works well
Multiple Level Caches (4)

- Vital to performance of modern processors
- L2 is usually shared by L1I and L1D
- Replacement strategy and write policy obviously gets more complex
E.g. Xeon E3-1280

- Available 2013Q2
- 4-core, 8-thread
- Core private caches
  - 32Kb L1 I-cache
  - 32Kb L1 D-cache
  - 256Kb L2 cache (I+D)
- 8Mb L3 cache (shared I+D by all cores)
- 2 channel DDR3
Cache Address Splitting

- 8 MB cache

Tag RAM

Compare

hit / miss

Tag Index

Index

Word ID Align

Word 0 Word 1

Word 7

Multiplexer

data

Address (32 bits)
Cache Example

- Assume CPU with simple L1 cache only
  - L1 cache 98% hit rate
  - L1 access time = 1 CPU cycle
  - RAM access = 50 cycles
  - Suggestion: consider 100 accesses

- What is effective overall memory access time?

- Assume CPU makes a RAM access (fetch) every cycle
Cache Example (solution)

- For every 100 accesses
  - 98 hit in cache : 98 * 1 cycle = 98 cycles
  - 2 miss in cache, go to RAM : 2 * (1+50) cycles = 102 cycles
  - Total : 98 + 102 = 200 cycles

- Average access time = 200/100 = 2 cycles

- CPU on average will only run at ½ speed
Now assume L2 cache between L1 & RAM

- Access time = 4 cycles
- Hit rate = 90%

L2: every 100 accesses take

- \((90 \times 4) + 10 \times (4 + 50) = 900\)
- Average access = 9 cycles
Two Level Cache Example

- Assume CPU with L1 and L2 cache
  - L1: 98% hit, 1 CPU cycle
  - L2 + RAM: average 9 CPU cycles
  - Suggestion: consider 100 accesses

- What is effective overall memory access time?

- Assume CPU makes a RAM access (fetch) every cycle
Two Level Cache Example (solution)

- Back to L1
- For every 100 accesses time taken

\[(1 \times 98) + 2 \times (1 + 9) = 118\text{cycles}\]

- Average access = 1.18 cycles

- Now approx 85% of potential full speed
Alternatively

- 1000 accesses
  - 980 will hit in L1 (98%)
  - 18 will hit in L2 (90% of 20)
  - 2 will go to main memory

- So access time is

  \[ 980 + 18 \times (1 + 4) + 2 \times (1 + 4 + 50) = 1180 \]

- Average = 1180/1000 = 1.18