Advanced Computer Architecture Chapter 1.4

Caches: a quick review of introductory *memory system* architecture

Objective: bring everyone up to speed, and also establish some key ideas that will come up later in the course in more complicated contexts

October 2022 Paul H J Kelly

These lecture notes are partly based on the course text, Hennessy and Patterson's Computer Architecture, a quantitative approach (6th ed), and on the lecture slides of David Patterson's Berkeley course (CS252)

Course materials online on https://scientia.doc.ic.ac.uk/2223/modules/60001/materials and https://www.doc.ic.ac.uk/~phjk/AdvancedCompArchitecture/aca20/ Intel Skylake quad-core die photo



Intel Skylake quad-core die photo



Intel Skylake quad-core die photo



https://en.wikichip.org/wiki/File:skylake (quad-core) (annotated).png

We finished the last lecture by asking how fast a pipelined processor can go?

- A simple 5-stage pipeline can run at 5-9GHz
- Limited by critical path through slowest pipeline stage logic
- Tradeoff: do more per cycle? Or increase clock rate?

Or do more per cycle, in parallel...

- At 3GHz, clock period is 330 picoseconds.
 - The time light takes to go about four inches
 - About 10 gate delays
 - for example, the Cell BE is designed for 11 FO4 ("fan-out=4") gates per cycle:

www.fe.infn.it/~belletti/articles/ISSCC2005-cell.pdf

Pipeline latches etc account for 3-5 FO4 delays leaving only 5-8 for actual work

How can we build a RAM that can implement our MEM stage in 5-8 FO4 delays?

Life used to be so easy Processor-DRAM Memory Gap (latency)



Time

In 1980 a large RAM's access time was close to the CPU cycle time. 1980s machines had little or no need for cache. Life is no longer quite so simple.

Levels of the Memory Hierarchy



~ Exponential increase in access latency, block size, capacity

- The Principle of Locality:
 - Programs access a relatively small portion of the address space at any instant of time.
- Two Different Types of Locality:
 - <u>Temporal Locality</u> (Locality in Time): If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
 - <u>Spatial Locality</u> (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straightline code, array access)
- Most modern architectures are heavily reliant (totally reliant?) on locality for speed

1 KB "Direct Mapped" Cache, 32B blocks

• For a 2^{N} byte cache:

- The uppermost (32 - N) bits are always the Cache Tag

- The lowest M bits are the Byte Select (Block Size = 2^{M})



Data: the cached data itself, arranged in cache lines/blocks

Direct-mapped cache - storage

1 KB "Direct Mapped" Cache, 32B blocks

• For a 2^{N} byte cache:

- The uppermost (32 - N) bits are always the Cache Tag

- The lowest M bits are the Byte Select (Block Size = 2^{M})



1 KB Direct Mapped Cache, 32B blocks



Associativity conflicts in a direct-mapped cache



Associativity conflicts in a direct-mapped cache



Direct-mapped Cache - structure

- Capacity: C bytes (eg 1KB)
- Blocksize: B bytes (eg 32)
- Byte select bits: 0..log(B)-1 (eg 0..4)
- Number of blocks: C/B (eg 32)
- Address size: A (eg 32 bits)
- Cache index size: I=log(C/B) (eg log(32)=5)
- Tag size: A-I-log(B) (eg 32-5-5=22)



Two-way Set Associative Cache

- N-way set associative: N entries for each Cache Index
 N direct mapped caches operated in parallel (N typically 2 to 4)
- Example: Two-way set associative cache
 - Cache Index selects a "set" from the cache
 - The two tags in the set are compared in parallel
 - Data is selected based on the tag result



Disadvantage of Set Associative Cache

- N-way Set Associative Cache v. Direct Mapped Cache:
 - N comparators vs. 1
 - Extra MUX delay for the data
 - Data comes AFTER Hit/Miss
- In a direct mapped cache, Cache Block is available BEFORE Hit/Miss:
 - Possible to assume a hit and continue. Recover later if miss.



Example: Intel Pentium 4 Level-1 cache (pre-Prescott)



- Block: 64 bytes (so there are 8K/64=128 blocks in the cache)
- Ways: 4 (addresses with same index bits can be placed in one of 4 ways)
- Sets: 32 (=128/4, that is each RAM array holds 32 blocks)
- Index: 5 bits (since 2⁵=32 and we need index to select one of the 32 ways)
- Tag: 21 bits (=32 minus 5 for index, minus 6 to address byte within block)
- Access time: 2 cycles, (.6ns at 3GHz; pipelined, dual-ported [load+store])



4 Questions for Memory Hierarchy

 Q1: Where can a block be placed in the upper level?

- Block placement

- Q2: How is a block found if it is in the upper level?
 - Block identification
- Q3: Which block should be replaced on a miss?
 Block replacement
- Q4: What happens on a write?
 - Write strategy

Q1: Where can a block be placed in the upper level?



In a direct-mapped cache, block 12 can only be placed in one cache location, determined by its low-order address bits -

 $(12 \mod 8) = 4$



In a two-way set-associative cache, the set is determined by its low-order address bits –

 $(12 \mod 4) = 0$

Block 12 can be placed in either of the two cache locations in set 0



In a fully-associative cache, block 12 can be placed in any location in the cache

More associativity:

- More comparators larger, more energy
- Better hit rate (diminishing returns)
- Reduced storage layout sensitivity more predictable

Q2: How is a block found if it is in the upper level?



- Tag on each block
 - No need to check index or block offset

Block Address	Block	
Тад	Index	Offset

Increasing associativity shrinks index, expands tag

Q3: Which block should be replaced on a miss?

- With Direct Mapped there is no choice
- With Set Associative or Fully Associative we want to choose
 - Ideal: least-soon re-used
 - LRU (Least Recently Used) is a popular approximation
 - Random is remarkably good in large caches

Assoc:	2-way		4-way		8-way	
Size	LRU	Ran	LRU	Ran	LRU	Ran
16 KB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 KB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

Benchmark studies show that LRU beats random only with small caches

LRU can be pathologically bad......

Q4: What happens on a write?

- <u>Write through</u>—The information is written to both the block in the cache and to the block in the lower-level memory
- <u>Write back</u>—The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.
 - is block clean or dirty?
- Pros and Cons of each?
 - WT: read misses cannot result in writes
 - WB: absorbs repeated writes to same location
- WT always combined with write buffers so that we don't wait for lower level memory

Caches are a big topic

- Cache coherency
 - If your data can be in more than one cache, how do you keep the copies consistent?
- Victim caches
 - Stash recently-evicted blocks in a small fully-associative cache (a "competitive strategy")
- Prefetching
 - Use a predictor to guess which block to fetch next before the processor requests it
- And much much more......

What's at the bottom of the memory hierarchy?

- StorageTek STK 9310 ("Powderhorn")
 - 2,000, 3,000, 4,000, 5,000, or 6,000
 cartridge slots per library storage module (LSM)
 - Up to 24 LSMs per library (144,000 cartridges)
 - 120 TB (1 LSM) to 28,800 TB capacity (24 LSM)
 - Each cartridge holds 300GB, readable up to 40 MB/sec
- Up to 28.8 petabytes
- Ave 4s to load tape
- 2017 product: Oracle SL8500
- Up to 1.2 Exabyte per unit
- Combine up to 32 units into single robot tape drive system
- http://www.oracle.com/us/products/ser vers-storage/storage/tapestorage/034341.pdf



https://www.itnews.com.au/gallery/inside-suns-multi-storey-colorado-data-centre-135385/page1

IBM System Storage Tape Library ts3500 ts4500 From https://www.youtube.com/watch?v=CVN93H6EuAU&list =PLp5rLKqrfZu_EvvnFM1HDptl_n0k5Th_q



StorageTek Powderhorn before disassembly, CERN 2007 http://www.flickriver.com/photos/naezmi/2074280052/#large



Can we live without cache?

- Interesting exception: Cray/Tera MTA, first delivered June 1999:
 - <u>www.cray.com/products/systems/mta/</u>
- Each CPU switches every cycle between 128 threads
- Each thread can have up to 8 outstanding memory accesses
- 3D toroidal mesh interconnect
- Memory accesses hashed to spread load across banks
- MTA-1 fabricated using Gallium Arsenide, not silicon
- "nearly un-manufacturable" (wikipedia)
- Third-generation Cray XMT:
 - <u>http://www.cray.com/Products/XMT.aspx</u>
 - YarcData's uRiKA (<u>http://www.yarcdata.com/products.html</u>)

Summary:

- Without caches we are in trouble
 - DRAM access times are commonly >100 cycles
- Without locality caches won't help
- Spatial vs temporal locality
- Direct-mapped
- Set-associative
- Associativity conflicts
- Policy questions:
 - Write-through
 - Write-back

memory parallelism to overcome this – especially in GPUs

We will look at various

techniques to exploit

We will see similar structures, and issues, in branch predictors, prefetching etc

We will see similar choices in cache coherency protocols for multicore

Many more – see next chapter!



Discussion exercise – the "Turing Tax"

- Then dynamic scheduling
- Then a deeper dive into caches and the memory hierarchy

In response to a student question:

- There is a tag for each 32-byte cache block (and in the 1KB cache, there would, as you say, be 32 blocks, since 1024=32x32).
- Two adjacent cache blocks could (normally will) hold 32-byte blocks from different parts of the memory.
- In a fully-associative cache we would have a tag and a tag comparator for every 32-byte block.
- In a direct-mapped cache, we have a tag for every block, but only one tag comparator.
- This is cheaper, faster and lower-power. But in order to make it work, we use some of the low-order address bits to index the cache - to select just one cache block. If its tag matches, we have a hit. If not, we don't. Similarly, when data is allocated into the cache. the same index bits are used to select the cache block that will be used (perhaps displacing whatever was there before).
- This means that different addresses that happen to have the same index bits map to the same cache block. So only one of them can be in the cache at the same time.