inst.eecs.berkeley.edu/~cs61c

CS61C : Machine Structures

Lecture 35
Caches IV / VM I

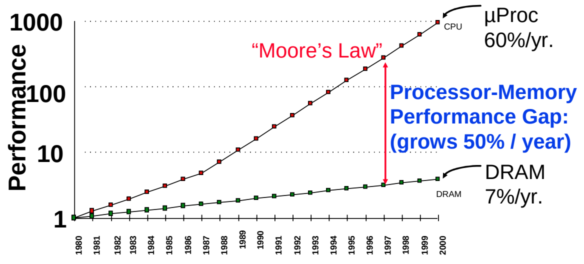
2004-11-19

Andy Carle

inst.eecs.berkeley.edu/~cs61c-ta

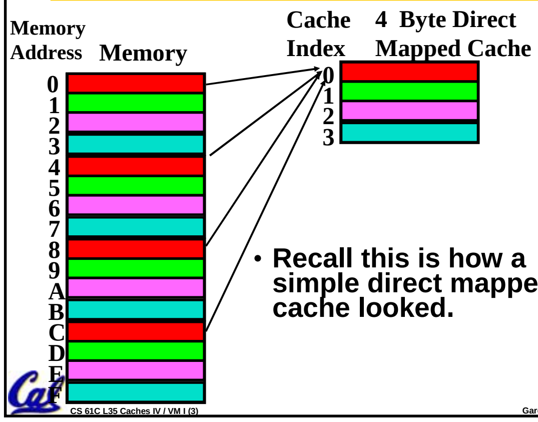
Google strikes back against recent encroachments into the Search world with the launch of two new services: Keyhole & Scholar.

Review: Why We Use Caches



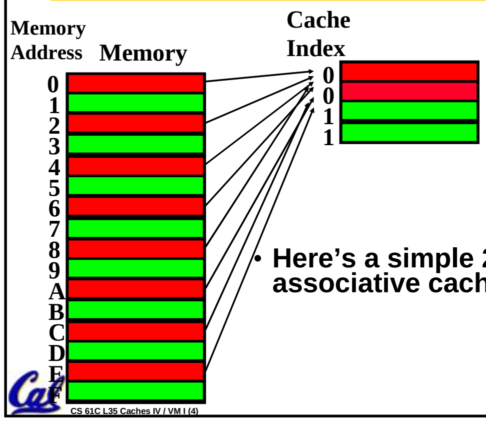
- 1989 first Intel CPU with cache on chip
- 1998 Pentium III has two levels of cache on chip

Review: Direct-Mapped Cache Example



- Recall this is how a simple direct mapped cache looked.

Review: Associative Cache Example



- Here's a simple 2 way set associative cache.

Block Replacement Policy (1/2)

- Direct-Mapped Cache: index completely specifies position which position a block can go in on a miss
- N-Way Set Assoc: index specifies a set, but block can occupy any position within the set on a miss
- Fully Associative: block can be written into any position
- Question: if we have the choice, where should we write an incoming block?

Block Replacement Policy (2/2)

- If there are any locations with valid bit off (empty), then usually write the new block into the first one.
- If all possible locations already have a valid block, we must pick a replacement policy: rule by which we determine which block gets "cached out" on a miss.

Block Replacement Policy: LRU

- LRU (Least Recently Used)
  - Idea: cache out block which has been accessed (read or write) least recently
  - Pro: temporal locality ⇒ recent past use implies likely future use: in fact, this is a very effective policy
  - Con: with 2-way set assoc, easy to keep track (one LRU bit); with 4-way or greater, requires complicated hardware and much time to keep track of this

Block Replacement Example

- We have a 2-way set associative cache with a four word *total* capacity and one word blocks. We perform the following word accesses (ignore bytes for this problem):

  0, 2, 0, 1, 4, 0, 2, 3, 5, 4

  How many hits and how many misses will there be for the LRU block replacement policy?

Block Replacement Example: LRU

- Addresses 0, 2, 0, 1, 4, 0, ...

  0: miss, bring into set 0 (loc 0)

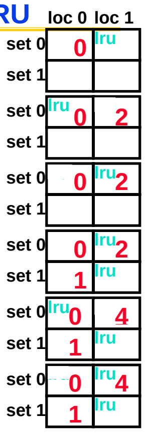
  2: miss, bring into set 0 (loc 1)

  0: hit

  1: miss, bring into set 1 (loc 0)

  4: miss, bring into set 0 (loc 1, replace 2)

  0: hit



Big Idea

- How to choose between associativity, block size, replacement policy?
- Design against a performance model
  - Minimize: *Average Memory Access Time*

    = Hit Time
      + Miss Penalty x Miss Rate
  - influenced by technology & program behavior
  - Note: Hit Time encompasses Hit Rate!!!
- Create the illusion of a memory that is large, cheap, and fast - on average

Example

- Assume
  - Hit Time = 1 cycle
  - Miss rate = 5%
  - Miss penalty = 20 cycles
  - Calculate AMAT…
- Avg mem access time

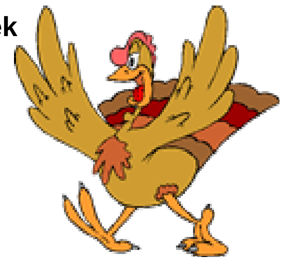
  = 1 + 0.05 x 20

  = 1 + 1 cycles

  = 2 cycles

Administrivia

- Do your reading! VM is hard!
- Project 3 Due Next Wednesday
- No Labs Next Week

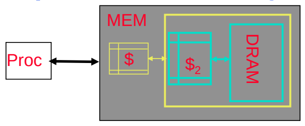


Ways to reduce miss rate

- Larger cache
  - limited by cost and technology
  - hit time of first level cache < cycle time
- More places in the cache to put each block of memory – associativity
  - fully-associative
    - any block any line
  - k-way set associated
    - k places for each block
    - direct map: k=1

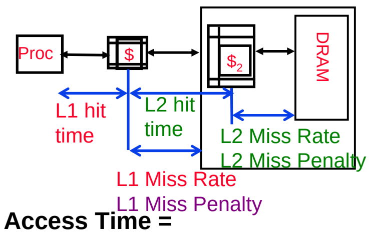
Improving Miss Penalty

- When caches first became popular, Miss Penalty ~ 10 processor clock cycles
- Today 2400 MHz Processor (0.4 ns per clock cycle) and 80 ns to go to DRAM ⇒ 200 processor clock cycles!



Solution: another cache between memory and the processor cache: Second Level (L2) Cache

Analyzing Multi-level cache hierarchy



Avg Mem Access Time =
L1 Hit Time + L1 Miss Rate * L1 Miss Penalty

L1 Miss Penalty =
L2 Hit Time + L2 Miss Rate * L2 Miss Penalty

Avg Mem Access Time =
L1 Hit Time + L1 Miss Rate *
(L2 Hit Time + L2 Miss Rate * L2 Miss Penalty)

Typical Scale

- L1
  - size: tens of KB
  - hit time: complete in one clock cycle
  - miss rates: 1-5%
- L2:
  - size: hundreds of KB
  - hit time: few clock cycles
  - miss rates: 10-20%
- L2 miss rate is fraction of L1 misses that also miss in L2
  - why so high?

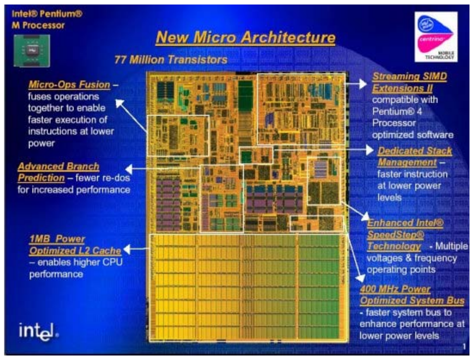
Example: with L2 cache

- Assume
  - L1 Hit Time = 1 cycle
  - L1 Miss rate = 5%
  - L2 Hit Time = 5 cycles
  - L2 Miss rate = 15% (% L1 misses that miss)
  - L2 Miss Penalty = 200 cycles
- L1 miss penalty = 5 + 0.15 * 200 = 35
- Avg mem access time = 1 + 0.05 x 35
  = 2.75 cycles

Example: without L2 cache

- Assume
  - L1 Hit Time = 1 cycle
  - L1 Miss rate = 5%
  - L1 Miss Penalty = 200 cycles
- Avg mem access time = 1 + 0.05 x 200
  = 11 cycles
- 4x faster with L2 cache! (2.75 vs. 11)

An Actual CPU – Pentium M


Intel® Pentium® M Processor
New Micro Architecture
77 Million Transistors
Micro-Ops Fusion – fuses operations together to enable faster execution of instructions at lower power
Advanced Branch Prediction – fewer re-dos for increased performance
1MB Power Optimized L2 Cache – enables higher CPU performance
Streaming SIMD Extensions II compatible with Pentium® 4 Processor optimized software
Dedicated Stack Management – faster instruction at lower power levels
Enhanced Intel® SpeedStep® Technology - Multiple voltages & frequency operating points
400 MHz Power Optimized System Bus - faster system bus to enhance performance at lower power levels
intel.

What to do on a write hit?

- **Write-through**
  - **update the word in cache block and corresponding word in memory**
- **Write-back**
  - **update word in cache block**
  - **allow memory word to be "stale"**

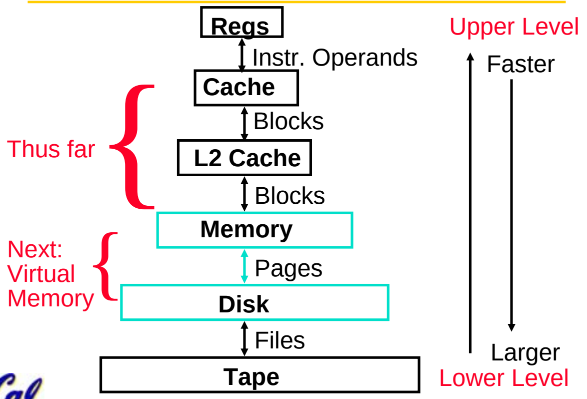
  ⇒ add 'dirty' bit to each block indicating that memory needs to be updated when block is replaced

  ⇒ OS flushes cache before I/O…
- **Performance trade-offs?**

Generalized Caching

- **We've discussed memory caching in detail. Caching in general shows up over and over in computer systems**
  - **Filesystem cache**
  - **Web page cache**
  - **Game Theory databases / tablebases**
  - **Software memoization**
  - **Others?**
- **Big idea: if something is expensive but we want to do it repeatedly, do it once and cache the result.**

Another View of the Memory Hierarchy



Memory Hierarchy Requirements

- **If Principle of Locality allows caches to offer (close to) speed of cache memory with size of DRAM memory, then recursively why not use at next level to give speed of DRAM memory, size of Disk memory?**
- **While we're at it, what other things do we need from our memory system?**

Memory Hierarchy Requirements

- **Share memory between multiple processes but still provide protection – don't let one program read/write memory from another**
- **Address space – give each program the illusion that it has its own private memory**
  - **Suppose code starts at address 0x40000000. But different processes have different code, both residing at the same address. So each program has a different view of memory.**

Virtual Memory

- Called "Virtual Memory"
- Also allows OS to share memory, protect programs from each other
- Today, more important for protection vs. just another level of memory hierarchy
- Historically, it predates caches

Peer Instruction

1. Increased associativity (1->2->4->8-way) ⇒ decreased or steady miss rate.
2. Increased associativity ⇒ increased cost & slower access time.
3. The ratio of costs of a "miss" vs. a "hit" are within an order of magnitude between VM & cache

	ABC
1:	FFF
2:	FFT
3:	FTF
4:	FTT
5:	TFF
6:	TFT
7:	TTF
8:	TTT

And in Conclusion…

- Cache design choices:
  - size of cache: speed v. capacity
  - direct-mapped v. associative
  - for N-way set assoc: choice of N
  - block replacement policy
  - 2nd level cache?
  - Write through v. write back?
- Use performance model to pick between choices, depending on programs, technology, budget, ...
- Virtual Memory
  - Predates caches; each process thinks it has all the memory to itself; protection!