WHEN FIBER OPTICS IS TOO SLOW

An 800-foot microwave tower in a Belgian cow pasture transmitted messages for
Microwave use has more recently moved to Europe, where trading firms are

Block Size Tradeoff
- Benefits of Larger Block Size
  - Spatial Locality: if we access a given word, we’re
  - Very applicable with Stored-Program Concept
  - Works well for sequential array accesses
- Drawbacks of Larger Block Size
  - Larger block size means larger miss penalty
  - If block size is too big relative to cache size, then
  - Result: miss rate goes up

Types of Cache Misses (1/2)
- “Three Cs” Model of Misses
  - 1st C: Compulsory Misses
    - occur when a program is first started
    - cache does not contain any of that program’s
data yet, so misses are bound to occur
    - can’t be avoided easily, so won’t focus on these in
this course

Types of Cache Misses (2/2)
- 2nd C: Conflict Misses
  - miss that occurs because two distinct memory
addresses map to the same cache location
  - two blocks (which happen to map to the same
location) can keep overwriting each other
  - big problem in direct-mapped caches
  - how do we lessen the effect of these?
  - Dealing with Conflict Misses
    - Solution 1: Make the cache size bigger
    - Solution 2: Multiple distinct blocks can fit in the
same cache index?

Fully Associative Cache (1/3)
- Memory address fields:
  - Tag: same as before
  - Offset: same as before
  - Index: non-existant
- What does this mean?
  - no “rows”: any block can go anywhere in the
cache
  - must compare with all tags in entire cache to see
if data is there
In fact, for a cache with \( M \) blocks,
- each set contains multiple blocks
- once we’ve found correct set, must compare with all tags in that set to find our data
- Size of $S$ is $\#$of sets x $N$ blocks x block size

N-Way Set Associative Cache (2/3)

- What’s so great about this?
  - even a 2-way set assoc cache avoids a lot of conflict misses
  - hardware cost isn’t that bad: only need $N$ comparators
- In fact, for a cache with $M$ blocks,
  - it’s Direct-Mapped if it’s 1-way set assoc
  - it’s Fully Assoc if it’s $M$-way set assoc
- so these two are just special cases of the more general set associative design

Fully Associative Cache (3/3)

- Benefit of Fully Assoc Cache
  - No Conflict Misses (since data can go anywhere)
- Drawbacks of Fully Assoc Cache
  - Need hardware comparator for every single entry:
    - if we have a 64KB of data in cache with 4B entries, we need 16K comparators: infeasible

Final Type of Cache Miss

- 3rd C: Capacity Misses
  - miss that occurs because the cache has a limited size
  - miss that would not occur if we increase the size of the cache
  - sketchy definition, so just get the general idea
- This is the primary type of miss for Fully Associative caches.

Block Replacement Policy

- 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?
  - if there’s a valid bit off, write new block into first invalid.
  - if all are valid, pick a replacement policy
    - rule for 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

Big Idea

- How to choose between associativity, block size, replacement & write policy?
- Design against a performance model
  - Minimize: Average Memory Access Time
    - Hit Time
      + Miss Penalty x Miss Rate
    - influenced by technology & program behavior
  - Create the illusion of a memory that is large, cheap, and fast - on average
  - How can we improve miss penalty?

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!

And in Conclusion...

- We’ve discussed memory caching in detail. Caching in general shows up over and over in computer systems
  - Filesystem cache, Web page cache, Game 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.
  - Cache design choices:
    - Size of cache: speed vs capacity
    - Block size (i.e., cache aspect ratio)
    - Write Policy (Write through v. write back)
    - Associativity choice of N (direct-mapped v. set v. fully associative)
    - Block replacement policy
    - 2nd level cache?
    - 3rd level cache?
  - Use performance model to pick between choices, depending on programs, technology, budget, ...

Bonus slides

- These are extra slides that used to be included in lecture notes, but have been moved to this, the “bonus” area to serve as a supplement.
  - The slides will appear in the order they would have in the normal presentation

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

Peer Instruction

1. A 2-way set-associative cache can be outperformed by a direct-mapped cache.
2. Larger block size ⇒ lower miss rate
3. a) FF, b) FT, c) TF, d) TT
Example

- Assume
  - Hit Time = 1 cycle
  - Miss rate = 5%
  - Miss penalty = 20 cycles
  - Calculate AMAT...
- Avg mem access time
  - $1 + 0.05 	imes 20$
  - $1 + 1$ cycles
  - $2$ cycles

Ways to reduce miss rate

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

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 = $5 + 0.15 	imes 200 = 35$
  - Avg mem access time = $1 + 0.05 	imes 35$
    - $2.75$ cycles

Example: without L2 cache

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

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?

An actual CPU – Early PowerPC

- Cache
  - 32 KB Instructions and 32 KB Data L1 caches
  - External L2 Cache interface with integrated controller and cache tags, supports up to 1 MByte external L2 cache
  - Dual Memory Management Units (MMU) with Translation Lookaside Buffers (TLB)
- Pipelining
  - Superscalar (3 inst/cycle)
  - 6 execution units (2 integer and 1 double precision IEEE floating point)