Review

"We would like to have the capacity of disk at the speed of the processor: unfortunately this is not feasible.

"So we create a memory hierarchy:

- each successively lower level contains "most used" data from next higher level
- exploits temporal locality
- do the common case fast, worry less about the exceptions (design principle of MIPS)

"Locality of reference is a Big Idea

Outline

- Block Size Tradeoff
- Types of Cache Misses
- Fully Associative Cache
- Course Advice
- N-Way Associative Cache
- Block Replacement Policy
- Multilevel Caches (if time)
- Cache write policy (if time)
Block Size Tradeoff (3/3)

° Hit Time = time to find and retrieve data from current level cache
° Miss Penalty = average time to retrieve data on a current level miss (includes the possibility of misses on successive levels of memory hierarchy)
° Hit Rate = % of requests that are found in current level cache
° Miss Rate = 1 - Hit Rate

Extreme Example: One Big Block

- Cache Size = 4 bytes
- Block Size = 4 bytes
- Only ONE entry in the cache!
- If item accessed, likely accessed again soon
- But unlikely will be accessed again immediately!
- The next access will likely to be a miss again
- Continually loading data into the cache but discard data (force out) before use it again
- Nightmare for cache designer: Ping Pong Effect

Block Size Tradeoff Conclusions

Types of Cache Misses (1/2)

° 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)

° 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
  - fails at some point
° 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-existent

- 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

Fully Associative Cache (2/3)

- Fully Associative Cache (e.g., 32 B block)
  - Compare tags in parallel

- 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

Third Type of Cache Miss

- 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 Associate caches.

Administrivia: General Course Philosophy

- Take variety of undergrad courses now to get introduction to areas
  - Can learn advanced material on own later once know vocabulary
- Who knows what you will work on over a 40 year career?

Administrivia: Courses for Telebears

- Take courses from great teachers!
- HKN ratings; >= 6 very good, < 5 not good
- www.hkn.eecs.uc.edu/student/coursesurveys.shtml

- Top Faculty / CS Course (may teach soon)
  - CS 70 Discrete Math Papadami. 6.3 S00
  - CS 150 Logic design Katz (DTA) 6.3 F92
  - CS 152 Computer Kubiatowicz 6.7 F99
  - CS 160 User Interface Rowe 6.0 F99
  - CS 164 Compilers Aiken 6.1 S00
  - CS 169 SW engin. Brewer 6.3 F99
  - CS 174 Combinatoric Sinclair 6.0 F97
  - CS 184 Graphics Sequin 6.1 S99
  - CS 188 Artific. Intel. Russell 6.0 F97
Administrivia: Courses for Telebears
° General Philosophy
  • Take courses from great teachers!
° Top Faculty / EE Course (may teach soon)
  • EE 105 Micro. Devices Howe 6.2 S98
  • EE 120 Signal, System Kahn 6.0 F99
  • EE 121 Noise Analysis Tse 6.8 S00
  • EE 130 I.C. Devices Hu (DTA) 6.6 F99
  • EE 140 Linear I.C.s Brodersen 6.2 F98
  • EE 141 Digital I.C.s Rabaey 6.4 F98
  • EE 142 I.C. for Comm. Meyer 6.2 F98
  • EE 143 Process I.C.s Cheung 6.0 S00
  • EE 192 Mechatronics Fearing 6.1 S00

If many good teachers: My recommendations
° CS169 Software Engineering
  • Everyone writes programs, even HW designers
  • Often programs are written in groups
    ⇒ learn skill now in school (before it counts)
° CS162 Operating Systems
  • All special-purpose HW will run a layer of SW
    that uses processes and concurrent programming; CS162 is the closest thing
° EE122 Introduction to Communication Networks
  • World is getting connected;
    communications must play major role

If many good teachers: Courses to consider
° E190 Technical Communication
  • Talent in writing and speaking critical for success
  • Now required for EECS majors
° CS 150 Lab Hardware Design
  • Hands on HW design
° CS 152 Design a Computer
° CS 186 Understand databases
  • Information more important now than computation?

Administrivia: Courses for Telebears
° Remember:
  ° Teacher quality more important to learning experience than official course content
  ° Take courses from great teachers!
° Distinguished Teaching Award:
  www.uga.berkeley.edu/sled/dta-dept.html
° HKN Evaluations:
  www-hkn.eecs.berkeley.edu/student/coursesurveys.shtml

N-Way Set Associative Cache (1/4)
° Memory address fields:
  • Tag: same as before
  • Offset: same as before
  • Index: points us to the correct “row” (called a set in this case)
° So what’s the difference?
  • each set contains multiple blocks
  • once we’ve found correct set, must compare with all tags in that set to find our data

N-Way Set Associative Cache (2/4)
° Summary:
  • cache is direct-mapped with respect to sets
  • each set is fully associative
  • basically N direct-mapped caches working in parallel: each has its own valid bit and data
**N-Way Set Associative Cache (3/4)**

- Given memory address:
  - Find correct set using Index value.
  - Compare Tag with all Tag values in the determined set.
  - If a match occurs, it’s a hit, otherwise a miss.
  - Finally, use the offset field as usual to find the desired data within the desired block.

**N-Way Set Associative Cache (4/4)**

- 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

**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 (N > 1): 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)**

- Solution:
  - 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 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

### 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

### Big Idea

- How chose between options of associativity, block size, replacement policy?
- Design against a performance model
  - Minimize: \( \text{Average Access Time} = \text{Hit Time} + \text{Miss Penalty} \times \text{Miss Rate} \)
  - influenced by technology and program behavior
- 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
- **Avg mem access time** = \( 1 + 0.05 \times 20 = 2 \text{ cycle} \)

### Improving Miss Penalty

- When caches first became popular, Miss Penalty ~ 10 processor clock cycles
- Today 1000 MHz Processor (1 ns per clock cycle) and 100 ns to go to DRAM ⇒ 100 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** = \( \text{L1 Hit Time} + \text{L1 Miss Rate} \times \text{L1 Miss Penalty} \)

**L1** Hit Time = \( \text{L1 Miss Rate} \times \text{L1 Miss Penalty} \)

**Avg Mem Access Time** = \( \text{L1 Hit Time} + \text{L1 Miss Rate} \times (\text{L2 Hit Time} + \text{L2 Miss Rate} \times \text{L2 Miss Penalty}) \)
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 (cont)

° 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 = 100 cycles

° L1 miss penalty = 5 + 0.15 * 100 = 20
° Avg mem access time = 1 + 0.05 x 20 = 2 cycle

Example: without L2 cache

° Assume
  • L1 Hit Time = 1 cycle
  • L1 Miss rate = 5%
  • L1 Miss Penalty = 100 cycles

° Avg mem access time = 1 + 0.05 x 100 = 6 cycles

° 3x faster with L2 cache

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 line indicating that memory needs to be updated when block is replaced
  • OS flushes cache before I/O !!!

° Performance trade-offs?

Things to Remember (1/2)

° Caches are NOT mandatory:
  • Processor performs arithmetic
  • Memory stores data
  • Caches simply make data transfers go faster

° Each level of memory hierarchy is just a subset of next higher level

° Caches speed up due to temporal locality: store data used recently

° Block size > 1 word speeds up due to spatial locality: store words adjacent to the ones used recently

Things to Remember (2/2)

° 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, ...