Review: Why We Use Caches

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

- Mechanism for transparent movement of data among levels of a storage hierarchy
  - set of address/value bindings
  - address => index to set of candidates
  - compare desired address with tag
  - service hit or miss
    - load new block and binding on miss
Block Size Tradeoff (1/3)

• Benefits of Larger Block Size

  • **Spatial Locality**: if we access a given word, we’re likely to access other nearby words soon

  • Very applicable with Stored-Program Concept: if we execute a given instruction, it’s likely that we’ll execute the next few as well

  • Works nicely in sequential array accesses too
Block Size Tradeoff (2/3)

• Drawbacks of Larger Block Size
  • Larger block size means larger miss penalty
    - on a miss, takes longer time to load a new block from next level
  • If block size is too big relative to cache size, then there are too few blocks
    - Result: miss rate goes up

• In general, minimize Average Memory Access Time (AMAT)
  = Hit Time
  + Miss Penalty x Miss Rate
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
Block Size Tradeoff Conclusions

- **Miss Penalty**
  - Block Size
  - Graph shows linear increase.

- **Miss Rate**
  - Block Size
  - Graph shows decrease, then increases after a point.

- **Exploits Spatial Locality**
  - Fewer blocks: compromises temporal locality

- **Average Access Time**
  - Block Size
  - Graph shows decrease, then increases after a point.

- **Increased Miss Penalty & Miss Rate**
  - Block Size
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
    - 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-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
Fully Associative Cache (2/3)

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

![Diagram of Fully Associative Cache]

- Cache Tag (27 bits long)
- Byte Offset
- Cache Data
- Valid

B 31  ·  ·  B 1  B 0

B 31  ·  ·  B 1  B 0

B 31  ·  ·  B 1  B 0

B 31  ·  ·  B 1  B 0

B 31  ·  ·  B 1  B 0

B 31  ·  ·  B 1  B 0

B 31  ·  ·  B 1  B 0
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
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 Associative caches.
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 w/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, hit!, otherwise a miss.
  • Finally, use the offset field as usual to find the desired data within the 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
Associative Cache Example

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

• This is also a **1-way set-associative cache**!
Associative Cache Example

- Here’s a simple 2 way set associative cache.
Pre-Exam Exercise #3

We are now going to stop for ~5 minutes. During this time, your goal is to (by yourself) come up with a potential exam exercise covering the topics of CPU Design & Pipelining. Make it as much like a real exam question as possible.

After this five minutes, you will explain your question to a small group and work through how you would go about solving it. I’ll call on some random samples for the full class.
Adminstrivia

• Midterm 2:
  • Friday, 11:00am – 2:00pm
  • 306 Soda (HP Auditorium)
  • Conflicts, DSP, &&|| terrified about the drop deadline: Contact Andy ASAP

• HW7 Out Now, Due Sunday
• Project 3 Out Later This Week
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 more 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, ...**

<table>
<thead>
<tr>
<th></th>
<th>loc 0</th>
<th>loc 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>set 0</td>
<td>0</td>
<td>lru</td>
</tr>
<tr>
<td>set 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th></th>
<th>loc 0</th>
<th>loc 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>set 0</td>
<td>lru</td>
<td>0</td>
</tr>
<tr>
<td>set 1</td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

0: **hit**

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

<table>
<thead>
<tr>
<th></th>
<th>loc 0</th>
<th>loc 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>set 0</td>
<td>0</td>
<td>lru</td>
</tr>
<tr>
<td>set 1</td>
<td>1</td>
<td>lru</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>loc 0</th>
<th>loc 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>set 0</td>
<td>lru</td>
<td>0</td>
</tr>
<tr>
<td>set 1</td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>

0: **hit**

<table>
<thead>
<tr>
<th></th>
<th>loc 0</th>
<th>loc 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>set 0</td>
<td>0</td>
<td>lru</td>
</tr>
<tr>
<td>set 1</td>
<td>1</td>
<td>lru</td>
</tr>
</tbody>
</table>
Big Idea

• How to choose between associativity, block size, replacement policy?

• Design against a performance model
  • Minimize: \( \text{Average Memory Access Time} \)
    \[ = \text{Hit Time} + \text{Miss Penalty} \times \text{Miss Rate} \]
  • influenced by technology & program behavior
  • Note: \( \text{Hit Time encompasses Hit Rate}!!! \)

• 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
  \[= 1 + 0.05 \times 20\]
  \[= 1 + 1\] cycles
  \[= 2\] cycles
Ways to reduce miss rate

• Larger cache
  • limited by cost and technology
  • hit time of first level cache < cycle time

• 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
Improving Miss Penalty

• When caches first became popular, Miss Penalty ~ 10 processor clock cycles

• Slightly more modern: 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 = \( L1 \text{ Hit Time} + L1 \text{ Miss Rate} \times L1 \text{ Miss Penalty} \)

\( L1 \text{ Miss Penalty} = \frac{L2 \text{ Hit Time} + L2 \text{ Miss Rate} \times L2 \text{ Miss Penalty}}{L2 \text{ Miss Rate}} \)

Avg Mem Access Time = \( L1 \text{ Hit Time} + L1 \text{ Miss Rate} \times (L2 \text{ Hit Time} + L2 \text{ Miss Rate} \times L2 \text{ Miss Penalty}) \)
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 * 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 \times 200
  = 11 \text{ cycles}

• 4x faster with L2 cache! (2.75 vs. 11)
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?**
An Actual CPU – Pentium M

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

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

1MB Power Optimized L2 Cache – enables higher CPU performance

400 MHz Power Optimized System Bus - faster system bus to enhance performance at lower power levels
Peer Instructions

1. In the last 10 years, the gap between the access time of DRAMs & the cycle time of processors has decreased. (i.e., is closing)

2. A 2-way set-associative cache can be outperformed by a direct-mapped cache.

3. Larger block size $\Rightarrow$ lower miss rate
Peer Instructions Answer

1. That was was one of the motivation for caches in the first place -- that the memory gap is big and widening.

2. Sure, consider the caches from the previous slides with the following workload: 0, 2, 0, 4, 2
   
   2-way: 0m, 2m, 0h, 4m, 2m; DM: 0m, 2m, 0h, 4m, 2h

3. Larger block size ⇒ lower miss rate, true until a certain point, and then the ping-pong effect takes over

1. In the last 10 years, the gap between the access time of DRAMs & the cycle time of processors has decreased. (i.e., is closing)

2. A 2-way set-associative cache can be outperformed by a direct-mapped cache.

3. Larger block size ⇒ lower miss rate
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
  • 2\textsuperscript{nd} level cache?
  • 3\textsuperscript{rd} level cache?
  • Write through v. write back?

• Use performance model to pick between choices, depending on programs, technology, budget, ...