Today's Lecture - Caches Reloaded

- Classic cache designs
- Cache misses and performance
- Writing to caches

Last Time: Fully Associative Cache

Building a cache with one comparator

Example: A Direct Mapped Cache

The limits of direct-mapped caches...
Recall: Performance Equation

\[
\text{Seconds} = \frac{\text{Program Instructions}}{\text{Program CPI}} \times \text{Cycles per Instruction} \times \frac{\text{Seconds}}{\text{Cycle}}
\]

Earlier, computed from ...

- Machine CPI
- Assumes a constant memory access time.

True CPI depends on the Average Memory Access Time (AMAT) for Inst & Data

AMAT = Hit Time + (Miss Rate x Miss Penalty)

Goal: Reduce AMAT

Beware! Improving one term may hurt other terms, and increase AMAT!

One type of cache miss: Conflict Miss

N blocks of same color in use at once, but cache can only hold M < N of them

Solution: Increase M (Associativity)

- Miss rate improvement equivalent to doubling cache size.
- Fully-associative

Other Solutions

- Increase number of cache lines (# blocks in cache)
- Why does this help?
- Add a small “victim cache” that holds blocks recently removed from the cache.
- Why does this help?

AMAT = Hit Time + (Miss Rate x Miss Penalty)

What if both regions have same block color?

<table>
<thead>
<tr>
<th>Cache Tag (26 bits)</th>
<th>Index</th>
<th>Byte Select (4 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache Data</td>
<td>Valid</td>
<td></td>
</tr>
<tr>
<td>Cache Tags</td>
<td>Valid</td>
<td></td>
</tr>
<tr>
<td>Cache Block</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16 bytes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PowerPC 970: 525K, 2-way set associative L1 P-cache

The benefits of set-associativity ...


Administrivia - Lab 3, HW 3 ...

- Homework 3 due 10/20 (Wednesday), 283 Soda, in CS 152 box at 5 PM
- Lab 3 final demo on 10/22 (Friday)
- Lab 3 report due: Monday 10/25 11:59 PM
Other causes of cache misses ...

Capacity Misses
Cache cannot contain all blocks accessed by the program
Solution: Increase size of the cache

Compulsory Misses
First access of a block by a program
Mostly unavoidable
Solution: Prefetch blocks (via hardware, software)

Also “Coherency Misses”: other processes update memory

Thinking about cache miss types ...

What kind of misses happen in a fully associative cache of infinite size?
A. Compulsory misses. Must bring each block into cache.

In addition, what kind of misses happen in a finite-sized fully associative cache?
A. Capacity misses. Program may use more blocks than can fit in cache.

In addition, what kind of misses happen in a set-associative or direct-map cache?
A. Conflict misses.

(all questions assume the replacement policy used is considered “optimal”)

Cache Block Replacement Strategy

After a cache read miss, if there are no empty cache blocks, which block should be removed from the cache?
The Least Recently Used (LRU) block? Appealing, but hard to implement.
A randomly chosen block? Easy to implement, how well does it work?

Miss Rate for 2-way Set Associative Cache

<table>
<thead>
<tr>
<th>Size</th>
<th>Random</th>
<th>LRU</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 KB</td>
<td>5.7%</td>
<td>5.2%</td>
</tr>
<tr>
<td>64 KB</td>
<td>2.0%</td>
<td>1.9%</td>
</tr>
<tr>
<td>256 KB</td>
<td>1.17%</td>
<td>1.15%</td>
</tr>
</tbody>
</table>

Also, try other LRU approx.

Why Hit Time Matters: CPU Interface!

Hit time is directly tied to clock rate of CPU.
If left unchecked, it increases when cache size and associativity increases.
Note that XScale pipelines both instruction and data caches, adding stages to the CPU pipeline.

Write Policy: Write-Through vs Write-Back

<table>
<thead>
<tr>
<th>Policy</th>
<th>Write-Through</th>
<th>Write-Back</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data written to cache block also written to lower-level memory</td>
<td>Write data only to the cache</td>
<td>Update lower level when a block falls out of the cache</td>
</tr>
</tbody>
</table>

Debug
Easy  Hard
Do read misses produce writes?
No  Yes
Do repeated writes make it to lower level?
Yes  No

Additional option -- let writes to an un-cached address allocate a new cache line (“write-allocate”).

Writes and Caches
Write Buffers for Write-Through Caches

Holds data awaiting write-through to lower level memory

Q. Why a write buffer?  
A. So CPU doesn’t stall

Q. Why a buffer, why not just one register?  
A. Bursts of writes are common.

Q. Are Read After Write (RAW) hazards an issue for write buffer?  
A. Yes! Drain buffer before next read, or check write buffers.

Lab 4: Why some groups have problems

Example bug: When does a write go to the cache?

A1. When address is already in the cache.  
Issue: Must check tag before writing, or else may overwrite the wrong address!  
Options: Stall and do tag check, or pipeline check.

A2. Always: we do allocate on write.

Cache Design: Datapath + Control

Most design errors come from incorrect specification of state machine behavior!  
Common bugs: Stalls, Block replacement, Write buffer

Conclusions

The cache design spectrum: from direct mapped to fully associative.

AMAT (Ave. Memory Access Time) = Hit Time + (Miss Rate x Miss Penalty)

Cache misses: conflict, capacity, compulsory, and coherency.

Lab 4 and project bugs usually from cache specification errors.