CS 61C: Great Ideas in Computer Architecture  
(Machine Structures)  
Caches Part 2  
Instructors:  
Krste Asanović & Randy H. Katz  
http://inst.eecs.berkeley.edu/~cs61c/

Outline
- Cache Organization and Principles
- Write Back vs. Write Through
- Cache Performance
- Cache Design Tradeoffs
- And in Conclusion ...

Typical Memory Hierarchy
- Principle of locality + memory hierarchy presents programmer with ≈ as much memory as is available in the cheapest technology at the ≈ speed offered by the fastest technology

Key Cache Concepts
- Principle of Locality
  - Temporal Locality and Spatial Locality
- Hierarchy of Memories (speed/size/cost per bit) to exploit locality
- Cache — copy of data in lower level of memory hierarchy
- Direct Mapped to find block in cache using Tag field and Valid bit for Hit
- Cache Design Organization Choices:  
  - Fully Associative, Set-Associative, Direct-Mapped
Cache Organizations

- "Fully Associative": Block placed anywhere in cache
  - First design last lecture
  - Note: No Index field, but one comparator/block
- "Direct Mapped": Block goes only one place in cache
  - Note: Only one comparator
  - Number of sets = number blocks
- "N-way Set Associative": N places for block in cache
  - Number of sets = Number of Blocks / N
  - N comparators
  - Fully Associative: N = number of blocks
  - Direct Mapped: N = 1

Note:

- Only Index field, but one comparator/block

First design last lecture

What Limits Number of Sets?

- For a given total number of blocks, we save comparators if have more than two sets
- Limit: As Many Sets as Cache Blocks => only one block per set – only needs one comparator!
- Called "Direct-Mapped" Design

Memory Block vs. Word Addressing

- **Block Offset**: Byte address within block
- **Set Index**: Selects which set
- **Tag**: Remaining portion of processor address

Processor Address Fields Used by Cache Controller

- **Tag**: Remaining portion of processor address
  - Size of Index = log2(number of sets)
  - Size of Tag = Address size – Size of Index
  - log2(number of bytes/block)

Direct Mapped Cache Example: Mapping a 6-bit Memory Address

- In example, block size is 4 bytes/1 word
- Memory and cache blocks always the same size, unit of transfer between memory and cache
- # Memory blocks >> # Cache blocks
  - 16 Memory blocks = 16 words = 64 bytes >> 6 bits to address all bytes
  - 4 Cache blocks, 4 bytes (1 word) per block
  - 4 Memory blocks map to each cache block
- Memory block to cache block, aka index: middle two bits
- Which memory block is in a given cache block, aka tag: top two bits
One More Detail: Valid Bit

- When start a new program, cache does not have valid information for this program
- Need an indicator whether this tag entry is valid for this program
- Add a “valid bit” to the cache tag entry
  - 0 => cache miss, even if by chance, address = tag
  - 1 => cache hit, if processor address = tag

Example: Alternatives in an 8 Block Cache

- Direct Mapped: 8 blocks, 1 way, 1 tag comparator, 8 sets
- Fully Associative: 8 blocks, 8 ways, 8 tag comparators, 1 set
- 2 Way Set Associative: 8 blocks, 2 ways, 2 tag comparators, 4 sets
- 4 Way Set Associative: 8 blocks, 4 ways, 4 tag comparators, 2 sets

Peer Instruction

- For a cache with constant total capacity, if we increase the number of ways by a factor of two, which statement is false:
  - A: The number of sets could be doubled
  - B: The tag width could decrease
  - C: The block size could stay the same
  - D: The block size could be halved
Outline
- Cache Organization and Principles
- Write Back vs. Write Through
- Cache Performance
- Cache Design Tradeoffs
- And in Conclusion …

Handling Stores with Write-Through
- Store instructions write to memory, changing values
- Need to make sure cache and memory have same values on writes: two policies
  1) Write-Through Policy: write cache and write through the cache to memory
     - Every write eventually gets to memory
     - Too slow, so include Write Buffer to allow processor to continue once data in Buffer
     - Buffer updates memory in parallel to processor

Write-Through Cache
- Write both values in cache and in memory
- Write buffer stops CPU from stalling if memory cannot keep up
- Write buffer may have multiple entries to absorb bursts of writes
- What if store misses in cache?

Handling Stores with Write-Back
  2) Write-Back Policy: write only to cache and then write cache block back to memory when evict block from cache
     - Writes collected in cache, only single write to memory per block
     - Include bit to see if wrote to block or not, and then only write back if bit is set
     - Called “Dirty” bit (writing makes it “dirty”)

Write-Back Cache
- Store/cache hit, write data in cache only and set dirty bit
  - Memory has stale value
- Store/cache miss, read data from memory, then update and set dirty bit
  - “Write-allocate” policy
- Load/cache hit, use value from cache
- On any miss, write back evicted block, only if dirty. Update cache with new block and clear dirty bit

Write-Through vs. Write-Back
- Write-Through:
  - Simpler control logic
  - More predictable timing simplifies processor control logic
  - Easier to make reliable, since memory always has copy of data (big idea: Redundancy!)
- Write-Back
  - More complex control logic
  - More variable timing [0,1,2 memory accesses per cache access]
  - Usually reduces write traffic
  - Harder to make reliable, sometimes cache has only copy of data
Administrivia

- Midterm #2 2 weeks away! October 31!
  - In class 9-9:30 AM
  - Synchronous digital design and Project 2 (processor design) included
  - Pipelines and Caches
  - ONE Double sided Crib sheet
  - Review Session: Saturday, Oct 28 (Location TBA)
- 5-10 open drop-in seats for these tutoring sessions:
  - M 3-4 Soda 611
  - Th 3-4 Soda 380
  - F 5-6 Soda 651
- Guerrilla Session tonight 7-9 pm in Cory 293
- Project 2-1 Party tomorrow 7-9 pm Cory 293
- If you would like to change your partnership for Project 2, email your lab TA
  - We will send out a Google form to track all Project 2 partnerships

Outline

- Cache Organization and Principles
- Write Back vs. Write Through
- Cache Performance
- Cache Design Tradeoffs
- And in Conclusion …

Cache (Performance) Terms

- Hit rate: fraction of accesses that hit in the cache
- Miss rate: 1 – Hit rate
- Miss penalty: time to replace a block from lower level in memory hierarchy to cache
- Hit time: time to access cache memory (including tag comparison)
- Abbreviation: “$“ = cache (a Berkeley innovation!)

Average Memory Access Time (AMAT)

- Average Memory Access Time (AMAT) is the average time to access memory considering both hits and misses in the cache
  \[ \text{AMAT} = \text{Time for a hit} + \text{Miss rate} \times \text{Miss penalty} \]

Important Equation!

Peer Instruction

AMAT = Time for a hit + Miss rate \times Miss penalty
- Given a 200 psec clock, a miss penalty of 50 clock cycles, a miss rate of 0.02 misses per instruction and a cache hit time of 1 clock cycle, what is AMAT?
  - A: ≤200 psec
  - B: 400 psec
  - C: 600 psec
  - D: ≥800 psec

Ping Pong Cache Example: Direct-Mapped Cache

w/4 Single-Word Blocks, Worst-Case Reference String
- Consider the main memory address reference string of word numbers:
  \[ 0 \ 4 \ 0 \ 4 \ 0 \ 4 \ 0 \ 4 \]
  - Start with an empty cache; all blocks initially marked as not valid
  - Use a 2-bit tag for each block

<table>
<thead>
<tr>
<th>Tag</th>
<th>0</th>
<th>4</th>
<th>0</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Consider the main memory address reference string of word numbers: 0 4 0 4 0 4
Start with an empty cache - all blocks initially marked as not valid.

• 8 requests, 0 misses

Ping-pong effect due to conflict misses - two memory locations that map into the same cache block.

Example: 2-Way Set Associative $\$ (4 words = 2 sets x 2 ways per set)

Cache
Way
Set
0
1
Tag
0
1
Data
0
1
Q: Is it there?

Compare all the cache tags in the set to the high order 3 memory address bits to tell if the memory block is in the cache.

Example: 4-Word 2-Way SA $\$, Same Reference String

• Consider the main memory word reference string

Four-Way Set-Associative Cache

• $2^4 = 256$ sets each with four ways (each with one block)

Outline

• Cache Organization and Principles
• Write Back vs. Write Through
• Cache Performance
• Cache Design Tradeoffs
• And in Conclusion ...

 ping pong cache example: direct-mapped cache w/4 single-word blocks, worst-case reference string

• Consider the main memory address reference string of word numbers:
 0 4 0 4 0 4
Start with an empty cache - all blocks initially marked as not valid.

• 8 requests, 0 misses

Ping pong cache example due to conflict misses - two memory locations that map into the same cache block.

Example: 2-Way Set Associative $\$ (4 words = 2 sets x 2 ways per set)

Cache
Way
Set
0
1
Tag
0
1
Data
0
1
Q: Is it there?

Compare all the cache tags in the set to the high order 3 memory address bits to tell if the memory block is in the cache.

Example: 4-Word 2-Way SA $\$, Same Reference String

• Consider the main memory address reference string

Start with an empty cache - all blocks initially marked as not valid.

• 8 requests, 0 misses

Solves the ping pong effect in a direct-mapped cache due to conflict misses since now two memory locations that map into the same cache set can co-exist!
Break!

Range of Set-Associative Caches

• For a fixed-size cache and fixed block size, each increase by a factor of two in associativity doubles the number of blocks per set (i.e., the number of ways) and halves the number of sets—decreases the size of the index by 1 bit and increases the size of the tag by 1 bit.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Index</th>
<th>Word Offset</th>
<th>Byte Offset</th>
</tr>
</thead>
</table>

Decreasing associativity, lower way, more sets
Increasing associativity, higher way, less sets

Direct mapped (only one way)
Singly tagged, only a single comparator

Total Cache Capacity = Associativity × # of sets × block_size

Bytes = blocks/set × sets × Bytes/block

C = N × S × B

address_size = tag_size + index_size + offset_size
= tag_size + log_2(S) + log_2(B)

Your Turn

• For a cache of 64 blocks, each block four bytes in size:
  1. The capacity of the cache is __bytes.
  2. Given a 2-way Set Associative organization, there are ___ sets, each of __ blocks, and ___ places a block from memory could be placed.
  3. Given a 4-way Set Associative organization, there are ___ sets each of __ blocks and ___ places a block from memory could be placed.
  4. Given an 8-way Set Associative organization, there are ___ sets each of __ blocks and ___ places a block from memory could be placed.
Costs of Set-Associative Caches

- N-way set-associative cache costs
  - N comparators (delay and area)
  - MUX delay (set selection) before data is available
  - Data available after set selection (and Hit/Miss decision).
  - DM S-block is available before the Hit/Miss decision
  - In Set-Associative, not possible to just assume a hit and continue and recover later if it was a miss
  - When miss occurs, which way’s block selected for replacement?
    - Least Recently Used (LRU): one that has been unused the longest (principle of temporal locality)
    - Must track when each way’s block was used relative to other blocks in the set
      - For 2-way SA S, one bit per set \( \rightarrow \) set to 1 when a block is referenced; reset the other way’s bit (i.e., “last used”)

Cache Replacement Policies

- Random Replacement
  - Hardware randomly selects a cache entry
- Least-Recently-Used
  - Hardware keeps track of access history
  - Replace the entry that has not been used for the longest time
  - For 2-way set-associative cache, need one bit for LRU replacement
- Example of a Simple "Pseudo" LRU Implementation
  - Assume 64 Fully Associative entries
  - Hardware replacement pointer points to one cache entry
  - Whomever access is made to the entry the pointer points to:
    - Move the pointer to the next entry
    - Otherwise: do not move the pointer
  - (example of “not-most-recently-used” replacement policy)

Benefits of Set-Associative Caches

- Choice of DM S versus SA S depends on the cost of a miss versus the cost of implementation
  - Largest gains are in going from direct mapped to 2-way (20%+ reduction in miss rate)

Outline

- Cache Organization and Principles
- Write Back vs. Write Through
- Cache Performance
- Cache Design Tradeoffs
- And in Conclusion ...
And in Conclusion ...

- Name of the Game: Reduce AMAT
  - Reduce Hit Time
  - Reduce Miss Rate
  - Reduce Miss Penalty
- Balance cache parameters (Capacity, associativity, block size)