CACHE OPTIMIZATION

Mahdi Nazm Bojnordi
Assistant Professor
School of Computing
University of Utah
Overview

- Announcement
  - Homework 4 will be released on Oct. 30th

- This lecture
  - Cache replacement policies
  - Cache write policies

- Reducing miss penalty
Cache Replacement Policies

- Which block to replace on a miss?
  - Only one candidate in direct-mapped cache
  - Multiple candidates in set/fully associative cache
Cache Replacement Policies

- Which block to replace on a miss?
  - Only one candidate in direct-mapped cache
  - Multiple candidates in set/fully associative cache
- Ideal replacement (Belady's algorithm)

<table>
<thead>
<tr>
<th>Cache Set</th>
<th>Requested Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td>A</td>
<td>--</td>
</tr>
</tbody>
</table>
Cache Replacement Policies

- Which block to replace on a miss?
  - Only one candidate in direct-mapped cache
  - Multiple candidates in set/fully associative cache

- Ideal replacement (Belady’s algorithm)
  - Replace the block accessed farthest in the future

<table>
<thead>
<tr>
<th>Cache Set</th>
<th>Requested Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>A</td>
</tr>
</tbody>
</table>
Cache Replacement Policies

- Which block to replace on a miss?
  - Only one candidate in direct-mapped cache
  - Multiple candidates in set/fully associative cache

- Ideal replacement (Belady’s algorithm)
  - Replace the block accessed farthest in the future

- Least recently used (LRU)

<table>
<thead>
<tr>
<th>Cache Set</th>
<th>Requested Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>A</td>
<td>B B B C A</td>
</tr>
</tbody>
</table>
Cache Replacement Policies

- Which block to replace on a miss?
  - Only one candidate in direct-mapped cache
  - Multiple candidates in set/fully associative cache

- Ideal replacement (Belady’s algorithm)
  - Replace the block accessed farthest in the future

- Least recently used (LRU)
  - Replace the block accessed farthest in the past
Cache Replacement Policies

- Which block to replace on a miss?
  - Only one candidate in direct-mapped cache
  - Multiple candidates in set/fully associative cache
- Ideal replacement (Belady’s algorithm)
  - Replace the block accessed farthest in the future
- Least recently used (LRU)
  - Replace the block accessed farthest in the past
- Most recently used (MRU)

<table>
<thead>
<tr>
<th>Cache Set</th>
<th>Requested Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>--</td>
<td>A B C B B B C A</td>
</tr>
<tr>
<td>--</td>
<td></td>
</tr>
</tbody>
</table>
Cache Replacement Policies

- Which block to replace on a miss?
  - Only one candidate in direct-mapped cache
  - Multiple candidates in set/fully associative cache

- Ideal replacement (Belady’s algorithm)
  - Replace the block accessed farthest in the future

- Least recently used (LRU)
  - Replace the block accessed farthest in the past

- Most recently used (MRU)
  - Replace the block accessed nearest in the past
Cache Replacement Policies

- Which block to replace on a miss?
  - Only one candidate in direct-mapped cache
  - Multiple candidates in set/fully associative cache

- Ideal replacement (Belady’s algorithm)
  - Replace the block accessed farthest in the future

- Least recently used (LRU)
  - Replace the block accessed farthest in the past

- Most recently used (MRU)
  - Replace the block accessed nearest in the past

- Random replacement
  - Hardware randomly selects a cache block to replace
Random vs. LRU

- **2-Random**: choose two cache elements at random, and evict the least-recently-used one.

Source: [https://danluu.com/2choices-eviction/](https://danluu.com/2choices-eviction/)
Example Problem

Blocks A, B, and C are mapped to a single set with only two block storages; find the miss rates for LRU and MRU policies.


Example Problem

- Blocks A, B, and C are mapped to a single set with only two block storages; find the miss rates for LRU and MRU policies.

   - LRU: 100%
   - MRU: 66%

   - LRU: 66%
   - MRU: 44%
Cache Write Policies

- Write vs. read

- Diagram showing tag, index, and byte arrangement.
Cache Write Policies

- Write vs. read
  - Data and tag are accessed for both read and write
  - Only for write, data array needs to be updated

- Cache write policies
Cache Write Policies

- Write vs. read
  - Data and tag are accessed for both read and write
  - Only for write, data array needs to be updated
- Cache write policies
Cache Write Policies

- **Write vs. read**
  - Data and tag are accessed for both read and write
  - Only for write, data array needs to be updated

- **Cache write policies**
Write (No-)Allocate

- **Write allocate**
  - allocate a cache line for the new data, and replace old line
  - just like a read miss

- **Write no allocate**
  - do not allocate space in the cache for the data
  - only really makes sense in systems with write buffers
Write (No-)Allocate

- **Write allocate**
  - allocate a cache line for the new data, and replace old line
  - just like a read miss

- **Write no allocate**
  - do not allocate space in the cache for the data
  - only really makes sense in systems with write buffers

- **How to handle read miss after write miss?**
Cache Write Policies

- Write vs. read
  - Data and tag are accessed for both read and write
  - Only for write, data array needs to be updated

- Cache write policies
Write back

- On a write access, write to cache only
  - write cache block to memory only when replaced from cache
  - dramatically decreases bus bandwidth usage
  - keep a bit (called the *dirty* bit) per cache block
Write through

- Write to both cache and memory (or next level)
  - Improved miss penalty
  - More reliable because of maintaining two copies
Write through

- Write to both cache and memory (or next level)
  - Improved miss penalty
  - More reliable because of maintaining two copies

- Use write buffer alongside cache
  - works fine if
    - rate of stores < 1 / DRAM write cycle
  - otherwise
    - write buffer fills up
    - stall processor to allow memory to catch up
Reducing Miss Penalty

- Some cache misses are inevitable
  - when they do happen, want to service as quickly as possible

- Other miss penalty reduction techniques
  - Multilevel caches
  - Giving read misses priority over writes
  - Sub-block placement
  - Critical word first
Reducing Miss Penalty

- Some cache misses are inevitable
  - when they do happen, want to service as quickly as possible

- Other miss penalty reduction techniques
  - Multilevel caches
  - Giving read misses priority over writes
  - Sub-block placement
  - Critical word first
Reducing Miss Penalty

- Some cache misses are inevitable
  - when they do happen, want to service as quickly as possible

- Other miss penalty reduction techniques
  - Multilevel caches
  - Giving read misses priority over writes
  - Sub-block placement
  - Critical word first
Reducing Miss Penalty

- Some cache misses are inevitable
  - when they do happen, want to service as quickly as possible

- Other miss penalty reduction techniques
  - Multilevel caches
  - Giving read misses priority over writes
  - Sub-block placement
  - Critical word first
Victim Cache

- How to reduce conflict misses
  - Larger cache capacity
  - More associativity

- Associativity is expensive
  - More hardware; longer hit time
  - More energy consumption

- Observation
  - Conflict misses do not occur in all sets
  - Can we increase associativity on the fly for sets?
Victim Cache

- Small fully associative cache
- On eviction, move the victim block to victim cache
Victim Cache

- Small fully associative cache
  - On eviction, move the victim block to victim cache
Cache Inclusion

- How to reduce the number of accesses that miss in all cache levels?
  - Should a block be allocated in all levels?
    - Yes: inclusive cache
    - No: non-inclusive or exclusive
  - Non-inclusive: only allocated in L1

- Modern processors
  - L3: inclusive of L1 and L2
  - L2: non-inclusive of L1 (large victim cache)