Caches: PA5 quickstart, metrics, cache friendly code

Yipeng Huang

Rutgers University

April 13, 2021
Table of contents

Announcements

PA5: Simulating a cache and optimizing programs for caches

Cache design parameters
   Cache placement policy (how to find data at address for read and write hit)
   Cache replacement policy (how to find space for read and write miss)
      Direct-mapped caches need no cache replacement policy
      Associative caches need a cache replacement policy (e.g., FIFO, LRU)
   Policies for writes from CPU to memory
   Multilevel cache hierarchies

Cache performance metrics: hits, misses, evictions
   Cache hits
   Cache misses

Cache-friendly code
   Loop interchange
   Cache blocking
Looking ahead

Class plan

3. PA5 now out. Due Monday, 4/26.
Table of contents

Announcements

PA5: Simulating a cache and optimizing programs for caches

Cache design parameters
  Cache placement policy (how to find data at address for read and write hit)
  Cache replacement policy (how to find space for read and write miss)
    Direct-mapped caches need no cache replacement policy
    Associative caches need a cache replacement policy (e.g., FIFO, LRU)
  Policies for writes from CPU to memory
  Multilevel cache hierarchies

Cache performance metrics: hits, misses, evictions
  Cache hits
  Cache misses

Cache-friendly code
  Loop interchange
  Cache blocking
PA5: Simulating a cache and optimizing programs for caches

Write a cache simulator
1. fully Associative
2. direct Mapped
3. set Associative

Optimize some code for better cache performance
1. cache Blocking
2. cache Oblivious
PA5: Simulating a cache and optimizing programs for caches

A tour of files in the package

▶ trace files
▶ csim-ref
Table of contents

Announcements

PA5: Simulating a cache and optimizing programs for caches

Cache design parameters
   Cache placement policy (how to find data at address for read and write hit)
   Cache replacement policy (how to find space for read and write miss)
      Direct-mapped caches need no cache replacement policy
      Associative caches need a cache replacement policy (e.g., FIFO, LRU)
   Policies for writes from CPU to memory
   Multilevel cache hierarchies

Cache performance metrics: hits, misses, evictions
   Cache hits
   Cache misses

Cache-friendly code
   Loop interchange
   Cache blocking
Cache placement policy (how to find data at address for read and write hit)

Several designs for caches

- Fully associative cache
- Direct-mapped cache
- E-way set-associative cache

Cache design options use $m$-bit memory addresses differently

- $t$-bit tag
- $s$-bit set index
- $b$-bit block offset

Figure: Memory addresses. Image credit CS:APP
Direct-mapped cache

No need for replacement policy

- The number of sets in cache is $S = 2^s = 2^2 = 4$.
- A hash function that limits exactly where a block can go.
- In direct-mapped cache, blocks can go into only one of $E = 1$ way.
- No cache replacement policy is needed.

Figure: Direct-mapped cache. Image credit CS:APP
Associative caches

Figure: Fully associative cache. Image credit CS:APP

Needs replacement policy

- Blocks can go into any of E ways
- Here, $E = 3$
- Good for capturing temporal locality.
- If all ways/lines/blocks are occupied, and a cache miss happens, which way/line/block will be the victim and get evicted for replacement?
Cache replacement policies for associative caches

FIFO: First-in, first-out

- Evict the cache line that was placed the longest ago.
- Each cache set essentially becomes limited-capacity queue.

LRU: Least Recently Used

- Evict the cache line that was last accessed longest ago.
- Needs a counter on each cache line, and/or a global counter (e.g., program counter).
Policies for writes from CPU to memory

How to deal with write-hit?

▶ **Write-through.** Simple. Writes update both cache and memory. Costly memory bus traffic.

▶ **Write-back.** Complex. Writes update only cache and set a dirty bit; memory updated only upon eviction. Reduces memory bus traffic. (For multi-core CPUs, motivates complex cache coherence protocols)

How to deal with write-miss?

▶ **No-write-allocate.** Simple. Write-misses do not load block into cache. But write-misses are not mitigated via cache support.

▶ **Write-allocate.** Complex. Write-misses will load block into cache.

Typical designs:

▶ **Simple:** write-through + no-write-allocate.
▶ **Complex:** write-back + write-allocate.
Multilevel cache hierarchies

Small fast caches nested inside large slow caches

- L1 data and instruction cache: 32KB, 64 set, 8-way associative, 64B block, 4 cycle latency.
- L2 cache: 256KB, 512 set, 8-way associative, 64B block, 10 cycle latency.
- L3 cache: 8MB, 8192 set, 16-way associative, 64B block, 40-75 cycle latency.

Notice how latency cost increases as $E$-way associativity increases.

Figure: Intel Core i7 cache hierarchy. Image credit CS:APP

Figure: Intel 2020 Gulftown die shot. Image credit AnandTech
Table of contents

Announcements

PA5: Simulating a cache and optimizing programs for caches

Cache design parameters
  Cache placement policy (how to find data at address for read and write hit)
  Cache replacement policy (how to find space for read and write miss)
    Direct-mapped caches need no cache replacement policy
    Associative caches need a cache replacement policy (e.g., FIFO, LRU)
  Policies for writes from CPU to memory
  Multilevel cache hierarchies

Cache performance metrics: hits, misses, evictions
  Cache hits
  Cache misses

Cache-friendly code
  Loop interchange
  Cache blocking
Cache hits

Memory access is serviced from cache

- Hit rate = \( \frac{\text{Number of hits}}{\text{Number of memory accesses}} \)
- Hit time: latency to access cache (4 cycles for L1, 10 cycles for L2)
Cache misses: metrics

Memory access cannot be serviced from cache

- Miss rate = \( \frac{{\text{Number of misses}}}{{\text{Number of memory accesses}}} \)
- Miss penalty (miss time): latency to access next level cache or memory (up to 200 cycles for memory).
- Average memory access time = hit time + miss rate \( \times \) miss penalty
Cache misses: Classification

Compulsory misses
- First access to a block of memory will miss because cache is cold.

Conflict misses
- Multiple blocks map (hash) to the same cache set.
- Fully associative caches have no such conflict misses.

Capacity misses
- Occurs when the set of active cache blocks (working set) is larger than the cache.
- Direct mapped caches have no such capacity misses.
Table of contents

Announcements

PA5: Simulating a cache and optimizing programs for caches

Cache design parameters
  - Cache placement policy (how to find data at address for read and write hit)
  - Cache replacement policy (how to find space for read and write miss)
    - Direct-mapped caches need no cache replacement policy
    - Associative caches need a cache replacement policy (e.g., FIFO, LRU)
  - Policies for writes from CPU to memory
  - Multilevel cache hierarchies

Cache performance metrics: hits, misses, evictions
  - Cache hits
  - Cache misses

Cache-friendly code
  - Loop interchange
  - Cache blocking
Cache-friendly code

Algorithms can be written so that they work well with caches

▶ Maximize hit rate
▶ Minimize miss rate
▶ Minimize eviction counts

Advanced optimizing compilers can automatically make such optimizations

▶ GCC optimizations
▶ -floop-interchange
▶ -floop-block
Loop interchange

Refer to textbook slides on "Rearranging loops to improve spatial locality"

▶ In PA5, fourth part "cacheBlocking" you can explore the impact of this on matrix multiplication.
▶ In practice, programmers do not have to worry about this optimization.
▶ Optimized automatically in GCC by compiler flag `-floop-interchange` and `-O3`
Cache blocking

Refer to textbook slides on "Using blocking to improve temporal locality"

- In PA5, fourth part "cacheBlocking" you can explore the impact of this on matrix multiplication.
- In practice, programmers do not have to worry about this optimization.
- Optimized automatically in GCC by compiler flag `-floop-block`. But it is not part of default optimizations such as `-O3` so you have to remember to set it.