UNIVERSITÀ DI PISA
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Cache Memory

Phd: De Vitis Gabriele Antonio

gabrieleantonio.devitis@ing.unipi.it

@CompArch2019

Cache Memory – Replacement Policy

- Direct mapped: no choice
- Set associative
  - Prefer non-valid entry, if there is one
  - Otherwise, choose among entries in the set
- Least-recently used (LRU)
  - Choose the one unused for the longest time
    - Simple for 2-way, manageable for 4-way, too hard beyond that
- Random
  - Gives approximately the same performance as LRU for high associativity

Cache Memory – **Write-through**

- On data-write hit, could just update the block in cache
  - But then cache and memory would be inconsistent
- Write through: also updates memory
- But makes writes take longer
  - e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
    - Effective CPI = 1 + 0.1×100 = 11
- Solution: write buffer
  - Holds data waiting to be written to memory
  - CPU continues immediately
    - Only stalls on write if write buffer is already full

Cache Memory – **Write-Back**

- Alternative: On data-write hit, just update the block in cache
  - Keep track of whether each block is dirty
- When a dirty block is replaced
  - Write it back to memory
  - Can use a write buffer to allow replacing block to be read first

Cache Memory – **Write Allocation**

- What should happen on a write miss?
- Alternatives for write-through
  - Allocate on miss: fetch the block
  - Write around: don't fetch the block
    - Since programs often write a whole block before reading it (e.g., initialization)
- For write-back
  - Usually fetch the block

Cache Memory – **Loading Policies**

How should a block be loaded into cache?

- Blocking
  - the requested word is only sent to the processor after the whole block has been loaded into cache
    - Simpler to implement
    - According to spatial locality, the next access will be to the same block
- Non Blocking
    - Greater impact in caches where the block loading implies several memory accesses.
  - Early Restart
    - fetch the words in normal order, but as soon as the requested word of the block arrives, send it to the processor and let the processor continue execution;
  - Critical Word First
    - request the missed word first from memory and send it to the processor as soon as it arrives; let the processor continue execution while filling the rest of the words in the block.

Cache Memory – **Measuring Performances (1)**

- Components of CPU time
  - Program execution cycles
    - Includes cache hit time
  - Memory stall cycles
    - Mainly from cache misses

- Assuming cache hit costs are included as part of the normal CPU execution cycle, then:

$$\text{CPU time} = IC \times CPI \times CC$$

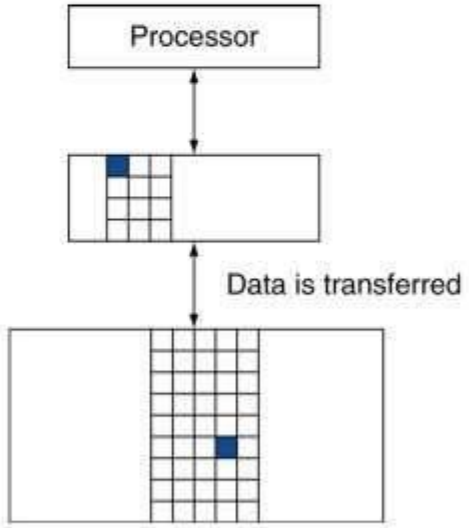
$$= IC \times \underbrace{(CPI_{ideal} + \text{Memory-stall cycles})}_{CPI_{stall}} \times CC$$

- For write-through caches:

$$\text{Memory-stall cycles} = \text{accesses/program} \times \text{miss rate} \times \text{miss penalty}$$

Cache Memory – **Measuring Performances (2)**

- Given
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles
  - Base CPI (ideal cache) = 2
  - Load & stores are 36% of instructions



- Miss cycles per instruction
  - I-cache: 0.02 × 100 = 2
  - D-cache: 0.36 × 0.04 × 100 = 1.44

- Actual CPI = 2 + 2 + 1.44 = 5.44
  - Ideal CPU is 5.44/2 =2.72 times faster

Cache Memory – **Measuring Performances (3)**

- Hit time is also important for performance
- Average memory access time (AMAT)

  AMAT = Hit time + Miss rate × Miss penalty

- Example

  CPU with 1ns clock, hit time = 1 cycle,

  miss penalty = 20 cycles, I-cache miss rate = 5%

  ➢AMAT = 1 + 0.05 × 20 = 2ns

  - 2 cycles per instruction

Cache Memory – **Measuring Performances (4)**

- When CPU performance increases
  - Miss penalty becomes more significant
- Decreasing based on CPI
  - Greater proportion of time spent on memory stalls
- Increasing clock rate
  - Memory stalls account for more CPU cycles

Can't neglect cache behavior when evaluating system performance

Cache Memory – Multilevel (1)

- Use multiple levels of caches
  - With advancing technology have more than enough room on the die for bigger L1 caches *or* for a second level of caches normally a unified L2 cache
    - i.e., holds both instructions and data
  - Many high-end systems already include unified L3 cache
- Design considerations for L1 and L2 caches are very different
  - Primary cache attached to CPU
    - focus on minimizing hit time (i.e. small, but fast)
      - » Smaller with smaller block sizes
  - Level-2 cache services misses from primary cache
    - focus on reducing miss rate (i.e. large, slower than L1)
      - to reduce the penalty of long main memory access times
        - » Larger with larger block sizes
        - » Higher levels of associativity

Cache Memory – Multilevel (2)

- Primary cache
  - Focus on minimal hit time
- L-2 cache
  - Focus on low miss rate to avoid main memory access
  - Hit time has less overall impact
- Results
  - L-1 cache usually smaller than a single cache
  - L-1 block size smaller than L-2 block size

$$t_{access} = t_{hitL1} + p_{missL1} \times t_{penaltyL1}$$
$$t_{penaltyL1} = t_{hitL2} + p_{missL2} \times t_{penaltyL2}$$
$$t_{access} = t_{hitL1} + p_{missL1} \times (t_{hitL2} + p_{missL2} \times t_{penaltyL2})$$

Cache Memory – Multilevel (3)

- Given
  - CPU base CPI = 1, clock rate = 4GHz
  - Miss rate/instruction = 2%
  - Main memory access time = 100ns
- With just primary cache
  - Miss penalty = 100ns/0.25ns = 400 cycles
  - Effective CPI = 1 + 0.02 × 400 = 9

Cache Memory – Multilevel (4)

- Now add L-2 cache
  - Access time = 5ns
  - Global miss rate to main memory = 0.5%
- Primary miss with L-2 hit
  - Penalty = 5ns/0.25ns = 20 cycles
- Primary miss with L-2 miss
  - Extra penalty = 400 cycles

$$CPI = 1 + 0.02 \times 20 + 0.005 \times 400 = 3.4$$

Performance ratio = 9/3.4 = 2.6

Cache Memory – Advanced CPU

- Out-of-order CPUs can execute instructions during cache miss
  - Pending store stays in load/store unit
  - Dependent instructions wait in reservation stations
    - Independent instructions continue
- Effect of miss depends on program data flow
  - Much harder to analyse
  - Use system simulation

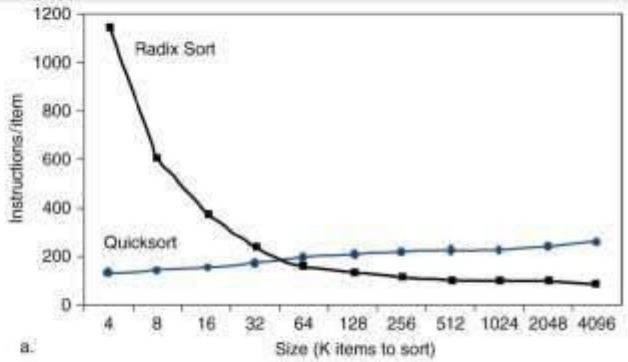
Cache Memory – Victim Cache

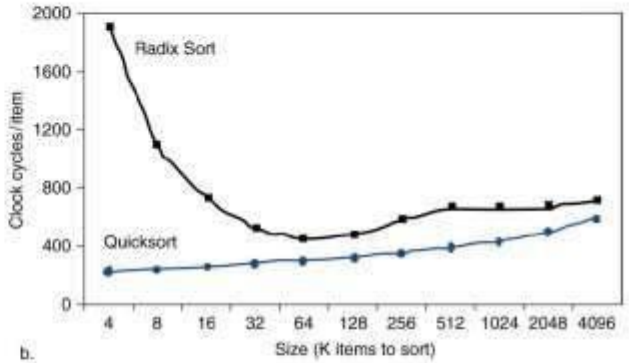
- Instead of completely discarding each block when it has to be replaced, temporarily keep it in a victim buffer.

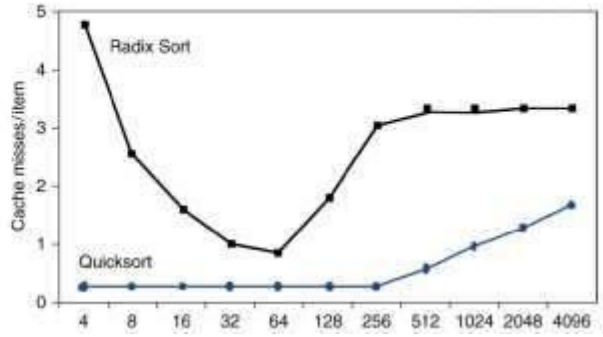
- Rather than stalling on a subsequent cache miss, the contents of the buffer are checked on a subsequent miss to see if they have the desired data before going to the next lower-level of memory.
  - Small cache (e.g., 4 to 16 positions)
  - Fully associative
  - Particularly efficient for small direct mapped caches (more than 25% reduction of the miss rate in a 4kB cache).

Cache Memory – Performance (1)

- Misses depend on memory access patterns
  - Algorithm behavior
  - Compiler optimization for memory access







Cache Memory – Performance (2)

1. Reduce the time to hit in the cache
   - smaller cache
   - direct mapped cache
   - smaller blocks
   - for writes
     - no write allocate – no "hit" on cache, just write to write buffer
     - write allocate – to avoid two cycles (first check for hit, then write) pipeline writes via a delayed write buffer to cache

2. Reduce the miss rate
   - bigger cache
   - more flexible placement (increase associativity)
   - larger blocks (16 to 64 bytes typical)
   - victim cache – small buffer holding most recently discarded blocks

Cache Memory – Performance (3)

3. Reduce the miss penalty
   - smaller blocks
   - use a write buffer to hold dirty blocks being replaced so don't have to wait for the write to complete before reading
   - check write buffer (and/or victim cache) on read miss
     - may get lucky
   - for large blocks fetch critical word first
   - use multiple cache levels – L2 cache not tied to CPU clock rate
   - faster backing store/improved memory bandwidth
     - wider buses
     - memory interleaving, DDR SDRAMs

Cache Memory - Conclusion

- Several interacting dimensions
  - cache size
  - block size
  - associativity
  - replacement policy
  - write-through vs write-back
  - write allocation