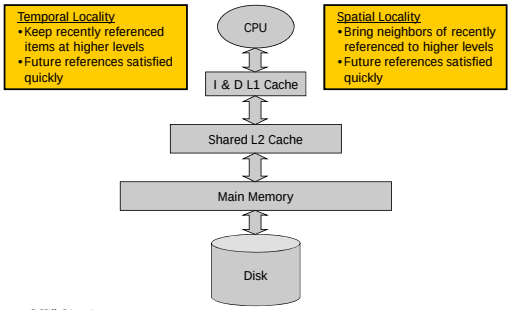
ECE/CS 552: Cache Performance

Instructor: Mikko H Lipasti

Fall 2010
University of Wisconsin-Madison

Lecture notes based on notes by Mark Hill
Updated by Mikko Lipasti

Memory Hierarchy



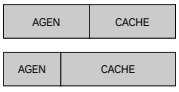
Caches and Performance

- Caches
  - Enable design for common case: cache hit
    - Cycle time, pipeline organization
    - Recovery policy
  - Uncommon case: cache miss
    - Fetch from next level
      - Apply recursively if multiple levels
    - What to do in the meantime?
- What is performance impact?
- Various optimizations are possible

Performance Impact

- Cache hit latency
  - Included in "pipeline" portion of CPI
    - E.g. IBM study: 1.15 CPI with 100% cache hits
  - Typically 1-3 cycles for L1 cache
    - Intel/HP McKinley: 1 cycle
      - Heroic array design
      - No address generation: load r1, (r2)
    - IBM Power4: 3 cycles
      - Address generation
      - Array access
      - Word select and align

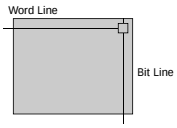
Cache Hit continued



- Cycle stealing common
  - Address generation < cycle
  - Array access > cycle
  - Clean, FSD cycle boundaries violated
- Speculation rampant
  - "Predict" cache hit
  - Don't wait for tag check
  - Consume fetched word in pipeline
  - Recover/flush when miss is detected
    - Reportedly 7 (!) cycles later in Pentium-IV

Cache Hits and Performance

- Cache hit latency determined by:
  - Cache organization
    - Associativity
      - Parallel tag checks expensive, slow
      - Way select slow (fan-in, wires)
    - Block size
      - Word select may be slow (fan-in, wires)
    - Number of block (sets x associativity)
      - Wire delay across array
      - "Manhattan distance" = width + height
      - Word line delay: width
      - Bit line delay: height



- Array design is an art form
  - Detailed analog circuit/wire delay modeling

Cache Misses and Performance

- Miss penalty
  - Detect miss: 1 or more cycles
  - Find victim (replace line): 1 or more cycles
    - Write back if dirty
  - Request line from next level: several cycles
  - Transfer line from next level: several cycles
    - (block size) / (bus width)
  - Fill line into data array, update tag array: 1+ cycles
  - Resume execution
- In practice: 6 cycles to 100s of cycles

Cache Miss Rate

- Determined by:
  - Program characteristics
    - Temporal locality
    - Spatial locality
  - Cache organization
    - Block size, associativity, number of sets

Improving Locality

- Instruction text placement
  - Profile program, place unreferenced or rarely referenced paths "elsewhere"
    - Maximize temporal locality
  - Eliminate taken branches
    - Fall-through path has spatial locality

Improving Locality

- Data placement, access order
  - Arrays: "block" loops to access subarray that fits into cache
    - Maximize temporal locality
  - Structures: pack commonly-accessed fields together
    - Maximize spatial, temporal locality
  - Trees, linked lists: allocate in usual reference order
    - Heap manager usually allocates sequential addresses
    - Maximize spatial locality
- Hard problem, not easy to automate:
  - C/C++ disallows rearranging structure fields
  - OK in Java

Cache Miss Rates: 3 C's [Hill]

- Compulsory miss
  - First-ever reference to a given block of memory
- Capacity
  - Working set exceeds cache capacity
  - Useful blocks (with future references) displaced
- Conflict
  - Placement restrictions (not fully-associative) cause useful blocks to be displaced
  - Think of as *capacity within set*

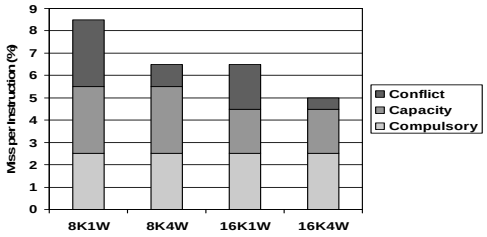
Cache Miss Rate Effects

- Number of blocks (sets x associativity)
  - Bigger is better: fewer conflicts, greater capacity
- Associativity
  - Higher associativity reduces conflicts
  - Very little benefit beyond 8-way set-associative
- Block size
  - Larger blocks exploit spatial locality
  - Usually: miss rates improve until 64B-256B
  - 512B or more miss rates get worse
    - Larger blocks less efficient: more capacity misses
    - Fewer placement choices: more conflict misses

Cache Miss Rate

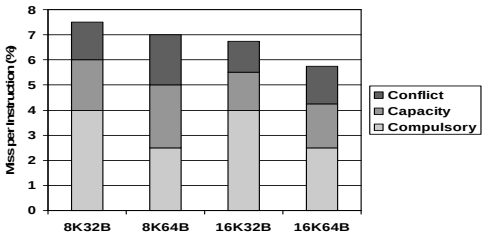
- Subtle tradeoffs between cache organization parameters
  - Large blocks reduce compulsory misses but increase miss penalty
    - #compulsory = (working set) / (block size)
    - #transfers = (block size)/(bus width)
  - Large blocks increase conflict misses
    - #blocks = (cache size) / (block size)
  - Associativity reduces conflict misses
  - Associativity increases access time
- Can associative cache ever have higher miss rate than direct-mapped cache of same size?

Cache Miss Rates: 3 C's



- Vary size and associativity
  - Compulsory misses are constant
  - Capacity and conflict misses are reduced

Cache Miss Rates: 3 C's



- Vary size and block size
  - Compulsory misses drop with increased block size
  - Capacity and conflict can increase with larger blocks

Cache Misses and Performance

- How does this affect performance?
- Performance = Time / Program

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}$$

(code size) (CPI) (cycle time)

- Cache organization affects cycle time
  - Hit latency
- Cache misses affect **CPI**

Cache Misses and CPI

$$CPI = \frac{cycles}{inst} = \frac{cycles_{hit}}{inst} + \frac{cycles_{miss}}{inst}$$
$$= \frac{cycles_{hit}}{inst} + \frac{cycles}{miss} \times \frac{miss}{inst}$$
$$= \frac{cycles_{hit}}{inst} + Miss\_penalty \times Miss\_rate$$

- Cycles spent handling misses are strictly additive
- Miss_penalty is recursively defined at next level of cache hierarchy as weighted sum of hit latency and miss latency

Cache Misses and CPI

$$CPI = \frac{cycles_{hit}}{inst} + \sum_{l=1}^{n} P_l \times MPI_l$$

- P_l is miss penalty at each of n levels of cache
- MPI_l is miss rate per instruction at each of n levels of cache
- Miss rate specification:
  - Per instruction: easy to incorporate in CPI
  - Per reference: must convert to per instruction
    - Local: misses per local reference
    - Global: misses per ifetch or load or store

Cache Performance Example

- Assume following:
  - L1 instruction cache with 98% per instruction hit rate
  - L1 data cache with 96% per instruction hit rate
  - Shared L2 cache with 40% local miss rate
  - L1 miss penalty of 8 cycles
  - L2 miss penalty of:
    - 10 cycles latency to request word from memory
    - 2 cycles per 16B bus transfer, 4x16B = 64B block transferred
    - Hence 8 cycles transfer plus 1 cycle to fill L2
    - Total penalty 10+8+1 = 19 cycles

Cache Performance Example

$$CPI = \frac{cycles_{hit}}{inst} + \sum_{l=1}^{n} P_l \times MPI_l$$

$$CPI = 1.15 + \frac{8cycles}{miss} \times \left(\frac{0.02miss}{inst} + \frac{0.04miss}{inst} \right) + \frac{19cycles}{miss} \times \frac{0.40miss}{ref} \times \frac{0.06ref}{inst}$$

$$= 1.15 + 0.48 + \frac{19cycles}{miss} \times \frac{0.024miss}{inst}$$

$$= 1.15 + 0.48 + 0.456 = 2.086$$

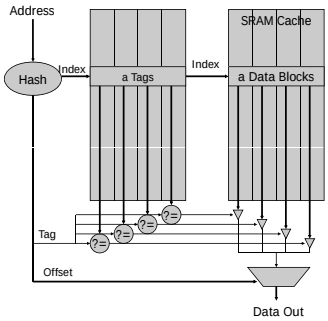
Cache Misses and Performance

- CPI equation
  - Only holds for misses that cannot be overlapped with other activity
  - Store misses often overlapped
    - Place store in store queue
    - Wait for miss to complete
    - Perform store
    - Allow subsequent instructions to continue in parallel
  - Modern out-of-order processors also do this for loads
    - Cache performance modeling requires detailed modeling of entire processor core

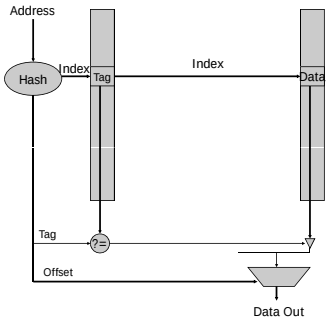
Caches Summary

- Four questions
  - Placement
    - Direct-mapped, set-associative, fully-associative
  - Identification
    - Tag array used for tag check
  - Replacement
    - LRU, FIFO, Random
  - Write policy
    - Write-through, writeback

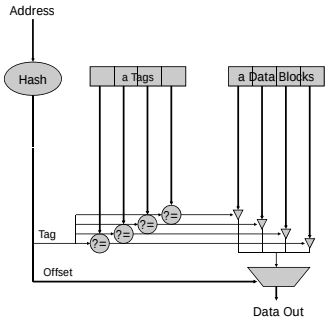
Caches: Set-associative



Caches: Direct-Mapped



Caches: Fully-associative



Caches Summary

$$CPI = \frac{cycles_{hit}}{inst} + \sum_{l=1}^{n} P_l \times MPI_l$$

- Hit latency
  - Block size, associativity, number of blocks
- Miss penalty
  - Overhead, fetch latency, transfer, fill
- Miss rate
  - 3 C's: compulsory, capacity, conflict
  - Determined by locality, cache organization