ECSE 425 Lecture 21: More Cache Basics; Cache Performance

H&P Appendix C

Last Time

- Two Questions
  - Q1: Block placement
  - Q2: Block identification

Today

- Two more questions
  - Q3: Block replacement
  - Q4: Write strategy
- Performance of CPUs with cache
  - Appendix C.2

Q3: Block Replacement

- Which block should be replaced on a miss?
- Direct mapped—no choice!
  - Index specifies the cache block for replacement
  - Advantage: simple logic
  - Disadvantage: higher miss rates
- Fully associative or set associative
  - Several blocks to choose from
  - Make good choices to reduce miss rates

Three Replacement Strategies

- Random—simple!
- Least-Recently Used (LRU)
  - Recently accessed blocks will likely be accessed again
  - The converse is also true: blocks not recently accessed are less likely to be accessed again
- True LRU is too expensive; estimate it
  - E.g., record when items are accessed
- First in, first out (FIFO)
  - Replace oldest block
  - Another approximation of LRU

Q4: Write Strategy

- Most cache accesses are reads
  - All instruction accesses are reads
  - MIPS: 10% stores, 26% loads
- Writes represent 7% of overall memory traffic
  - 28% of traffic to the data cache
- Two conflicting design principles
  - Make the common case fast: optimize for reads
  - Amdahl's law: don't neglect writes

Reads vs. Writes

- Reads are the easiest to make fast
  - Read block at same time as tag check
  - Discard data if tags don't match
    - Power is the only drawback
- Why are writes slow?
  - Writes cannot begin until after tag check
  - Writes can be of variable width
    - Reads too; power is the only drawback of reading more

Two Write Strategies

- Write back
  - Write information only to block in the cache
  - Write the modified cache block to the main memory only when it is replaced
- Write through
  - Write information to the block in the cache and
  - Write to the block in lower-level memory

Write Back

- Advantages:
  - Writes occur at speed of cache
  - Multiple writes to a block coalesced into one write to main memory
    - Reduces pressure on memory bandwidth
    - Reduces power dissipated in the memory system
- "Dirty bit" kept for each block
  - Indicates whether the block is "dirty" (modified while in the cache) or "clean" (not modified)
  - On replacement, "clean" blocks are discarded
  - Only "dirty" blocks trigger writing to the next level

Write Through

- Simple, easier to implement
- Cache is always clean
  - Read misses never result in writes to lower level
  - Main memory always has current data
  - Reduces the complexity of cache coherency

Stalling on Writes

- Avoid stalling on writes by using a write buffer
  – Once data is written to the buffer, continue
- Loads must then check the write buffer for data
  – If the data is cached, but the write is still buffered, cache doesn't contain the current value

Two Write Miss Strategies

- Write allocate
  - Make write misses act like read misses
  - Retrieve the block, then proceed as if access was a hit
- No-write allocate
  - Don’t retrieve the block: modify it directly in lower-level memory instead
- Normally
  - WB caches write allocate (benefit from locality)
  - WT caches don’t write allocate (avoid redundant writes to multiple levels of memory)

Average Memory Access Time

- Miss rate
  - Accesses that miss / Total accesses
  - Convenient metric
  - Independent of the speed of the hardware
- A better measure Average Memory Access Time
  - AMAT = HitTime + MissRate × MissPenalty

Example: Unified vs. Split Caches

- Instruction and data streams are different
  - Instructions are fixed size and read-only
  - Instruction stream is predictable
- Divide cache capacity between two caches
  - An instruction cache (I$) accessed during IF
  - A data cache (D$) accessed during MEM

Example: Unified vs. Split Caches, Cont'd

- 16 KB I$ and 16 KB D$ vs. 32 KB U$
  - 16 KB I$ has 3.83 misses per 1000 instructions
  - 16 KB D$ has 40.9 misses per 1000 instructions
  - 32 KB U$ has 43.3 misses per 1000 instructions
- Assume
  - 10% of instructions are stores
  - 26% of instructions are loads
  - 1 cycle hit time
  - 100 cycle miss penalty
  - 1 extra cycle penalty for U$—structural hazard
- What is the AMAT in each case?
- Does miss rate predict AMAT?

CPU Performance with Imperfect Caches

- When caches are perfect …
  - $\text{CPUTime} = \text{IC} \times \text{CPI}_{\text{base}} \times \text{CCTime}$
  - One cycle of "hit" time is included in CPI_{base}
- When caches aren't perfect, stalls!

$$\begin{aligned}\text{CPUTime} &= \text{IC} \times \left(\text{CPI}_{\text{base}} + \frac{\text{MemoryStalls}}{\text{Instruction}}\right) \times \text{CCTime} \\ &= \text{IC} \times \left(\text{CPI}_{\text{base}} + \frac{\text{MemoryAccesses}}{\text{Instruction}} \times \text{MissRate} \times \text{MissPenalty}\right) \times \text{CCTime}\end{aligned}$$

CPU Performance Example 1

- Assume
  - In-order processor
  - Miss penalty of 200 CC
  - CPI = 1 (when we ignore memory stalls)
  - 1.5 memory accesses per instruction on average
  - 30 misses / 1000 instructions
- Compare CPUTime with, and without cache

Cache Impact on Performance

- As CPI decreases
  - The relative penalty of a cache miss increases
- With faster clocks, a fixed memory delay yields more stall clock cycles!
- Amdahl's law states that
  - If we decrease CPI,
  - But average memory access time is fixed, then
  - The overall speedup is limited by the fraction of time spent computing relative to total execution time

CPU Performance Example 2

- With a perfect cache
  - CPI = 1.6
  - CC = 0.35 ns
  - 1.4 mem refs / instruction
- Compare two 128 KB caches
  - 64 bytes blocks
  - Miss penalty = 65 ns
  - Hit time = 1 cc
- Direct mapped
  - Miss rate = 2.1%
- 2-way set associative
  - $CC_{2\text{-way}} = 1.35\ CC_{1\text{-way}}$
  - Miss rate = 1.9%
- What is CPUTime in each case?
- Does AMAT predict CPUTime?

AMAT is not CPU Time!

- In the previous example:
  - AMAT is lower for 2-way
  - CPUTime is lower for 1-way
  - Full simulation is the best predictor of performance
- 2-way set associative cache increases the clock cycle for ALL instructions
  - Slower ALU operations
  - Slower hits (the common case)
  - Degrades performance despite improving miss rate

Out-of-Order Execution

- Miss penalty does not mean the same thing
  - The machine does not totally stall on a cache miss
  - Other instructions allowed to proceed
- What is the miss penalty for OOO Execution?
  - Full latency of the memory miss? No
  - The non-overlapped latency when the CPU must stall
- Define:

$$\frac{\text{MemoryStallCycles}}{\text{Instruction}} = \frac{\text{Misses}}{\text{Instruction}} \times (\text{TotalMissLatency} - \overset{\substack{\textbf{Because of O-O-E}\\ \downarrow}}{\text{OverlappedMissLatency}})$$

Summary

- Block replacement
  - Random, LRU, FIFO
- Write strategy
  - Write-back
  - Write-through
- Write misses
  - Write allocate
  - No-write allocate
- Performance with Caches
  - Miss Rate
  - Average Memory Access Time
  - CPU Time
- OOO-E vs. IO-E
  - Some of the miss penalty can be overlapped

Next Time

- Basic cache optimizations
  - Appendix C.3