Lecture 16: Cache Memories

• Last Time
  - AMAT - average memory access time
  - Basic cache organization

• Today
  - Take QUIZ 12 over P&H 5.7-10 before 11:59pm today
  - Read 5.4, 5.6 for 3/25
  - Homework 6 due Thursday March 25, 2010

  - Hardware cache organization
  - Reads versus Writes
  - Cache Optimization
Cache Memory Theory

- Small fast memory + big slow memory
- Looks like a big fast memory
The Memory Hierarchy

- **Registers**
  - Latency: 1 cyc
  - Bandwidth: 3-10 words/cycle
  - Allocation: compiler managed
  - Size: < 1KB

- **Level 1 Cache**
  - Latency: 1-3cy
  - Bandwidth: 1-2 words/cycle
  - Allocation: hardware managed
  - Size: 32KB - 1MB

- **Level 2 Cache**
  - Latency: 5-10cy
  - Bandwidth: 1 word/cycle
  - Allocation: hardware managed
  - Size: 1MB - 4MB

- **DRAM**
  - Latency: 30-100cy
  - Bandwidth: 0.5 words/cycle
  - Allocation: OS managed
  - Size: 64MB - 4GB

- **Disk**
  - Latency: 10^6-10^7cy
  - Bandwidth: 0.01 words/cycle
  - Allocation: OS managed
  - Size: 4GB+

- **Tape**
Direct Mapped

- Each block mapped to exactly 1 cache location

Cache location = (block address) MOD (# blocks in cache)
• Each block mapped to any cache location

Cache location = any
Set Associative

- Each block mapped to subset of cache locations

Set selection = (block address) MOD (# sets in cache)

2-way set associative = 2 blocks in set
This example: 4 sets
How do we use memory address to find block in the cache?
How Do We Find a Block in The Cache?

- **Our Example:**
  - Main memory address space = 32 bits (= 4GBytes)
  - Block size = 4 words = 16 bytes
  - Cache capacity = 8 blocks = 128 bytes

![Diagram of 32-bit Address Structure]

- index ⇒ which set
- tag ⇒ which data/instruction in block
- block offset ⇒ which word in block
- # tag/index bits determine the associativity
- tag/index bits can come from anywhere in block address
Finding a Block: Direct-Mapped

With cache capacity = 8 blocks
Finding A Block: 2-Way Set-Associative

- \( S \) - sets
- \( A \) - elements in each set
  - \( A \)-way associative
- \( S=4, A=2 \)
  - 2-way associative
  - 8-entry cache

Diagram:
- 4 sets
- 2 elements per set
- 2-way associative
- 8-entry cache
- Tag
- Index
- Address
- Hit
- Data
Finding A Block: Fully Associative
Set Associative Cache - cont'd

- All of main memory is divided into S sets
  - All addresses in set N map to same set of the cache
    - Addr = N mod S
    - A locations available
- Shares costly comparators across sets
- Low address bits select set
  - 2 in example
- High address bits are tag, used to associatively search the selected set
- Extreme cases
  - A=1: Direct mapped cache
  - S=1: Fully associative
- A need not be a power of 2
Cache Organization

- Where does a block get placed? - DONE
- How do we find it? - DONE
- Which one do we replace when a new one is brought in?
- What happens on a write?
Which Block Should Be Replaced on Miss?

- **Direct Mapped**
  - Choice is easy - only one option

- **Associative**
  - Randomly select block in set to replace
  - Least-Recently used (LRU)

- **Implementing LRU**
  - 2-way set-associative
  - >2 way set-associative
What Happens on a Store?

• Need to keep cache consistent with main memory
  - Reads are easy - no modifications
  - Writes are harder - when do we update main memory?

• Write-Through
  - On cache write - always update main memory as well
  - Use a write buffer to stockpile writes to main memory for speed

• Write-Back
  - On cache write - remember that block is modified (dirty bit)
  - Update main memory when dirty block is replaced
  - Sometimes need to flush cache (I/O, multiprocessing)
BUT: What if Store Causes Miss!

- Write-Allocate
  - Bring written block into cache
  - Update word in block
  - Anticipate further use of block

- No-write Allocate
  - Main memory is updated
  - Cache contents unmodified
Improving cache performance
How Do We Improve Cache Performance?

\[ AMAT = t_{hit} + p_{miss} \cdot \text{penalty}_{miss} \]
How Do We Improve Cache Performance?

\[ AMAT = t_{hit} + p_{miss} \cdot penalty_{miss} \]

- Reduce hit time
- Reduce miss rate
- Reduce miss penalty
Questions to think about

• As the block size goes up, what happens to the miss rate?
• ... what happens to the miss penalty?
• ... what happens to hit time?
• As the associativity goes up, what happens to the miss rate?
• ... what happens to the hit time?
Reducing Miss Rate: Increase Associativity

- Reduce conflict misses
- Rules of thumb
  - 8-way = fully associative
  - Direct mapped size N = 2-way set associative size N/2

- But!
  - Size N associative is larger than Size N direct mapped
  - Associative typically slower than direct mapped ($t_{hit}$ larger)
Reducing Hit Time

- **Make Caches small and simple**
  - Hit Time = 1 cycle is good (3.3ns!)
  - L1 - low associativity, relatively small

- **Even L2 caches can be broken into sub-banks**
  - Can exploit this for faster hit time in L2
Reducing Miss Rate: Increase Block Size

- Fetch more data with each cache miss
  - 16 bytes $\Rightarrow$ 64, 128, 256 bytes!
  - Works because of Locality (spatial)
Reduce Miss Penalty: Transfer Time

• Should we transfer the whole block at once?

• Wider path to memory
  - Transfer more bytes/cycle
  - Reduces total time to transfer block
  - Limited by wires

• Two ways to do this:
  - Wider path to each memory
  - Separate paths to multiple memories
    “multiple memory banks”

• Block size and transfer unit not necessarily equal!
Reduce Miss Penalty: Deliver Critical word first

- Only need one word from block immediately

- Don’t write entire word into cache first
  - Fetch word 2 first (deliver to CPU)
  - Fetch order: 2 3 0 1

LW R3, 8(R5)
Reduce Miss Penalty: More Cache Levels

- Average access time = 
  \[ \text{HitTime}_{L1} + \text{MissRate}_{L1} \times \text{MissPenalty}_{L1} \]
- \[ \text{MissPenalty}_{L1} = \text{HitTime}_{L2} + \text{MissRate}_{L2} \times \text{MissPenalty}_{L2} \]
- etc.
- Size/Associativity of higher level caches?
Reduce Miss Penalty: Read Misses First

- Let reads pass writes in Write buffer

SW 512(R0),R3
LW R1,1024(R0)
LW R2,512(R0)
Reduce Miss Penalty: 
Lockup (nonblocking) Free Cache

- Let cache continue to function while miss is being serviced

LW R1,1024(R0)  
LW R2,512(R0)  

MAIN MEMORY

CPU

write buffer

Data

Tag

LW R2,512(R0)

LW R1,1024(R0)

MISS

UTCS 352, Lecture 16
Reducing Miss Rate: Prefetching

• Fetching Data that you will probably need

• Instructions
  - Alpha 21064 on cache miss
    • Fetches requested block into instruction stream buffer
    • Fetches next sequential block into cache

• Data
  - Automatically fetch data into cache (spatial locality)
  - Issues?

• Compiler controlled prefetching
  - Inserts prefetching instructions to fetch data for later use
  - Registers or cache
Reducing Miss Rate: Use a “Victim” Cache

- Small cache (< 8 fully associative entries)
  - Jouppi 1990
  - Put evicted lines in the victim FIRST
  - Search in both the L1 and the victim cache
  - Accessed in parallel with main cache
  - Captures conflict misses
VC: Victim Cache Example

Given direct mapped L1 of 4 entries, fully associative 1 entry VC
Address access sequence
- 8, 9, 10, 11, 8, 12, 9, 10, 11, 12, 8
  ^       ^             ^
  A       B             C
First access to 12 misses, put 8 in VC, put 12 in L1, third access to 8 hits in VC

After: A                       B                                       C

8 9 10 11 8 9 10 11 12 9 10 11 12 8 9 10 11 8
Summary

• Recap
  - Using a memory address to find location in cache
  - Deciding what to evict from the cache
  - Improving cache performance

• Next Time
  - Homework 6 is due March 25, 2010
  - Reading: P&H 5.4, 5.6
    - Virtual Memory
    - TLBs