Lec 3 – Memory Hierarchy Review: Caches

Since 1980, CPU has outpaced DRAM ...

Four-issue 2GHz superscalar accessing 100ns DRAM could execute 800 instructions during time for one memory access!

Gap grew 50% per year

CPU
60% per yr
2X in 1.5 yrs

DRAM
9% per yr
2X in 10 yrs

Performance
(1/latency)

1980
1990
2000
Year
1000
100
10

9/4/07 CS252-Fall'07
Addressing the Processor-Memory Performance GAP

- **Goal:** Illusion of large, fast, cheap memory. Let programs address a memory space that scales to the disk size, at a speed that is usually as fast as register access.

- **Solution:** Put smaller, faster “cache” memories between CPU and DRAM. Create a “memory hierarchy”.

Levels of the Memory Hierarchy

<table>
<thead>
<tr>
<th>Capacity</th>
<th>Access Time</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Registers</td>
<td>100s Bytes</td>
<td>&lt;10s ns</td>
</tr>
<tr>
<td>Cache</td>
<td>K Bytes</td>
<td>10-100 ns</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1-0.1 cents/bit</td>
</tr>
<tr>
<td>Main Memory</td>
<td>M Bytes</td>
<td>$0.0001 - .00001 cents /bit</td>
</tr>
<tr>
<td>Disk</td>
<td>G Bytes, 10 ms</td>
<td>(10,000,000 ns)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$10^{-5} - 10^{-8}$ cents/bit</td>
</tr>
<tr>
<td>Tape</td>
<td>infinite sec-min</td>
<td>10^{-8}</td>
</tr>
</tbody>
</table>

Today’s Focus: Registers

Upper Level

- Registers
- Instr. Operands
- Blocks
- Pages
- Memory

Middle Level

- Cache

Lower Level

- Disk
- Files
- Tape

Staging Xfer Unit

- progs/compilers
- cache ctrl
- OS
- user/operator
- Mbytes

Upper Level

- faster

Lower Level

- Larger

- Today's Focus: Registers

9/4/07  CS252-Fall'07

3

9/4/07  CS252-Fall'07

4
Common Predictable Patterns

Two predictable properties of memory references:

• **Temporal Locality:** If a location is referenced, it is likely to be referenced again in the near future (e.g., loops, reuse).

• **Spatial Locality:** If a location is referenced it is likely that locations near it will be referenced in the near future (e.g., straightline code, array access).

Memory Reference Patterns

Caches

Caches exploit both types of predictability:

- Exploit temporal locality by remembering the contents of recently accessed locations.
- Exploit spatial locality by fetching blocks of data around recently accessed locations.

Cache Algorithm (Read)

Look at Processor Address, search cache tags to find match. Then either

**HIT** - Found in Cache
- Return copy of data from cache

**MISS** - Not in cache
- Read block of data from Main Memory

**Hit Rate** = fraction of accesses found in cache

**Miss Rate** = 1 – Hit rate

**Hit Time** = RAM access time +
- time to determine HIT/MISS

**Miss Time** = time to replace block in cache +
- time to deliver block to processor
Inside a Cache

Processor

CACHE

Main Memory

Address

Data

Address

Data

copy of main memory location 100

copy of main memory location 101

Address

Tag

Data Block

Line

100

Data

Byte

100

Data

Byte

6848

Data

Byte

416

Data

Byte

416

Data

Byte

-----

-----

-----

-----

-----

-----

-----

-----


4 Questions for Memory Hierarchy

- Q1: Where can a block be placed in the cache? (Block placement)
- Q2: How is a block found if it is in the cache? (Block identification)
- Q3: Which block should be replaced on a miss? (Block replacement)
- Q4: What happens on a write? (Write strategy)
Q1: Where can a block be placed?

Block Number

Memory

Set Number

Cache

Block 12 can be placed

Q2: How is a block found?

• Index selects which set to look in
• Tag on each block
  – No need to check index or block offset
• Increasing associativity shrinks index, expands tag. Fully Associative caches have no index field.
**Direct-Mapped Cache**

- **Tag**
- **Index**
- **Block Offset**

![Diagram of Direct-Mapped Cache]

**2-Way Set-Associative Cache**

- **Tag**
- **Index**
- **Block Offset**

![Diagram of 2-Way Set-Associative Cache]
What causes a MISS?

- **Three Major Categories of Cache Misses:**
  - **Compulsory Misses:** first access to a block
  - **Capacity Misses:** cache cannot contain all blocks needed to execute the program
  - **Conflict Misses:** block replaced by another block and then later retrieved - (affects set assoc. or direct mapped caches)

Nightmare Scenario: ping pong effect!
Block Size and Spatial Locality

Block is unit of transfer between the cache and memory

<table>
<thead>
<tr>
<th>Tag</th>
<th>Word0</th>
<th>Word1</th>
<th>Word2</th>
<th>Word3</th>
</tr>
</thead>
</table>

4 word block, \( b=2 \)

Split CPU address

- \( 32 \)-b bits
- \( b \) bits

\( 2^b = \) block size \( a.k.a \) line size (in bytes)

Larger block size has distinct hardware advantages
- less tag overhead
- exploit fast burst transfers from DRAM
- exploit fast burst transfers over wide busses

What are the disadvantages of increasing block size?

Fewer blocks => more conflicts. Can waste bandwidth.

Q3: Which block should be replaced on a miss?

- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - Least Recently Used (LRU)
    » LRU cache state must be updated on every access
    » true implementation only feasible for small sets (2-way)
    » pseudo-LRU binary tree often used for 4-8 way
  - First In, First Out (FIFO) a.k.a. Round-Robin
    » used in highly associative caches
- Replacement policy has a second order effect since replacement only happens on misses
Q4: What happens on a write?

- **Cache hit:**
  - *write through:* write both cache & memory
    » generally higher traffic but simplifies cache coherence
  - *write back:* write cache only
    (memory is written only when the entry is evicted)
    » a dirty bit per block can further reduce the traffic

- **Cache miss:**
  - *no write allocate:* only write to main memory
  - *write allocate (aka fetch on write):* fetch into cache

- **Common combinations:**
  - write through and no write allocate
  - write back with write allocate

5 Basic Cache Optimizations

- **Reducing Miss Rate**
  1. Larger Block size (compulsory misses)
  2. Larger Cache size (capacity misses)
  3. Higher Associativity (conflict misses)

- **Reducing Miss Penalty**
  4. Multilevel Caches

- **Reducing hit time**

- **Giving Reads Priority over Writes**
  - E.g., Read complete before earlier writes in write buffer