Chapter 5

Large and Fast: Exploiting Memory Hierarchy
Principle of Locality

- Programs access a small proportion of their address space at any time
- Temporal locality
  - Items accessed recently are likely to be accessed again soon
  - E.g., instructions in a loop, induction variables
- Spatial locality
  - Items near those accessed recently are likely to be accessed soon
  - E.g., sequential instruction access, array data
Taking Advantage of Locality

- Memory hierarchy
- Store everything on disk
- Copy recently accessed (and nearby) items from disk to smaller DRAM memory
  - Main memory
- Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory
  - Cache memory attached to CPU
Memory Hierarchy

<table>
<thead>
<tr>
<th>Speed</th>
<th>Size</th>
<th>Cost ($/bit)</th>
<th>Current technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fastest</td>
<td>Smallest</td>
<td>Highest</td>
<td>SRAM</td>
</tr>
<tr>
<td>Slowest</td>
<td>Biggest</td>
<td>Lowest</td>
<td>Magnetic disk</td>
</tr>
</tbody>
</table>

*(__________ replaces magnetic disks in personal mobile devices)*
Memory Hierarchy Levels

- **Block (aka line):** unit of copying
  - May be multiple words

- **If accessed data is present in upper level**
  - **Hit:** access satisfied by upper level
    - Hit ratio: hits/accesses

- **If accessed data is absent**
  - Miss: block copied from lower level
    - Time taken: miss penalty
    - Miss ratio: misses/accesses
      - $= 1 - \text{hit ratio}$
    - Then accessed data supplied from upper level

* (Hierarchy: data cannot be present in level $i$ unless it is also present in level $(\_\_\_\_\_\_)$)
Memory Technology

- Static RAM (SRAM)* (___ tr/bit; on chip cache)
  - 0.5ns – 2.5ns, $2000 – $5000 per GB

- Dynamic RAM (DRAM)
  - 50ns – 70ns, $20 – $75 per GB

- Magnetic disk
  - 5ms – 20ms, $0.20 – $2 per GB

- Ideal memory
  - Access time of SRAM
  - Capacity and cost/GB of disk

* (Flash memory: ______ μs, $0.75 - $1.0 per __)
DRAM Technology

- Data stored as a charge in a capacitor
  * (Can be kept for a few _____)
- Single transistor used to access the charge
- Must periodically be refreshed
  - Read contents and write back
  - Performed on a DRAM “row”
DRAM Structure

Chapter 5 — Large and Fast: Exploiting Memory Hierarchy — 8
Advanced DRAM Organization

- Bits in a DRAM are organized as a rectangular array
  - DRAM accesses an entire row
  - Burst mode: supply successive words from a row with reduced latency

- Double data rate (DDR) DRAM
  - Transfer on rising and falling clock edges

- Quad data rate (QDR) DRAM
  - Separate DDR inputs and outputs

* (DDR4: million transfers/sec with MHz clock)
DRAM Generations

<table>
<thead>
<tr>
<th>Year</th>
<th>Capacity</th>
<th>$/GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1980</td>
<td>64Kbit</td>
<td>$1500000</td>
</tr>
<tr>
<td>1983</td>
<td>256Kbit</td>
<td>$500000</td>
</tr>
<tr>
<td>1985</td>
<td>1Mbit</td>
<td>$200000</td>
</tr>
<tr>
<td>1989</td>
<td>4Mbit</td>
<td>$50000</td>
</tr>
<tr>
<td>1992</td>
<td>16Mbit</td>
<td>$15000</td>
</tr>
<tr>
<td>1996</td>
<td>64Mbit</td>
<td>$10000</td>
</tr>
<tr>
<td>1998</td>
<td>128Mbit</td>
<td>$4000</td>
</tr>
<tr>
<td>2000</td>
<td>256Mbit</td>
<td>$1000</td>
</tr>
<tr>
<td>2004</td>
<td>512Mbit</td>
<td>$250</td>
</tr>
<tr>
<td>2007</td>
<td>1Gbit</td>
<td>$50</td>
</tr>
</tbody>
</table>

(2010 __Gbit $__)  (2012 __Gbit $__)
DRAM Performance Factors

- **Row buffer**
  - Allows several words to be read and refreshed in parallel

- **Synchronous DRAM**
  - Allows for consecutive accesses in bursts without needing to send each address
  - Improves bandwidth

- **DRAM banking**
  - Allows simultaneous access to multiple DRAMs
  - Improves bandwidth
Increasing Memory Bandwidth

- **4-word wide memory**
  - Miss penalty = 1 + 15 + 1 = 17 bus cycles
  - Bandwidth = 16 bytes / 17 cycles = 0.94 B/cycle

- **4-bank interleaved memory**
  - Miss penalty = 1 + 15 + 4×1 = 20 bus cycles
  - Bandwidth = 16 bytes / 20 cycles = 0.8 B/cycle
Flash Storage

- Nonvolatile semiconductor storage
  - $100 \times - 1000 \times$ faster than disk
  - Smaller, lower power, more robust
  - But more $/$GB (between disk and DRAM)
Flash Types

- NOR flash: bit cell like a NOR gate
  - Random read/write access
  - Used for instruction memory in embedded systems

- NAND flash: bit cell like a NAND gate
  - Denser (bits/area), but block-at-a-time access
  - Cheaper per GB
  - Used for USB keys, media storage, …

- Flash bits wears out after 1000’s of accesses
  - Not suitable for direct RAM or disk replacement
  - Wear leveling: remap data to less used blocks

* (FTL: F____ T_______ L____)
**Flash Types**

- **NOR flash**

- **NAND flash**
Disk Storage

- Nonvolatile, rotating magnetic storage
  \[ \star (\text{_____ ~ ______ revolution/min}) \]

\[ \star (10\text{'s of _____/surface}) \]

\[ \star (\text{____ s/track; } \text{_____ ~ ____ B}) \]
Disk Sectors and Access

Each sector records
- Sector ID
- Data (512 bytes, 4096 bytes proposed)
- Error correcting code (ECC)
  - Used to hide defects and recording errors
- Synchronization fields and gaps

Access to a sector involves
- Queuing delay if other accesses are pending
- Seek: move the heads\(^*\) (____ msec)
- Rotational latency
- Data transfer\(^*\) (____ ~ ____MB/sec; ____MB/sec with cache)
- Controller overhead
Disk Access Example

- Given
  - 512B sector, 15,000rpm, 4ms average seek time, 100MB/s transfer rate, 0.2ms controller overhead, idle disk

- Average read time
  - 4ms seek time
    + $\frac{1}{2} / \left(\frac{15,000}{60}\right) = 2\text{ms rotational latency}$
    + $\frac{512}{100\text{MB/s}} = 0.005\text{ms transfer time}$
    + 0.2ms controller delay
    = 6.2ms

- If actual average seek time is 1ms
  - Average read time = 3.2ms
Disk Performance Issues

- Manufacturers quote average seek time
  - Based on all possible seeks
  - Locality and OS scheduling lead to smaller actual average seek times

- Smart disk controller allocate physical sectors on disk
  - Present logical sector interface to host
  - SCSI, ATA, SATA

- Disk drives include caches
  - Prefetch sectors in anticipation of access
    Avoid seek and rotational delay

* (Small Computer System Interface; AT Attachment; Serial ATA)
### Cache Memory

- **Cache memory**
  - The level of the memory hierarchy closest to the CPU
- **Given accesses** $X_1, \ldots, X_{n-1}, X_n$

#### 5.3 The Basics of Caches
- How do we know if the data is present?
- Where do we look?

| $X_4$ | | $X_4$ |
|-------| |-------|
| $X_1$ | | $X_1$ |
| $X_{n-2}$ | | $X_{n-2}$ |
| $X_{n-1}$ | | $X_{n-1}$ |
| $X_2$ | | $X_2$ |
| $X_3$ | | $X_3$ |

a. Before the reference to $X_n$

b. After the reference to $X_n$
Direct Mapped Cache

- Location determined by address
- Direct mapped: only one choice
  - (Block address) modulo (#Blocks in cache)

- #Blocks is a power of 2
- Use low-order address bits
Tags and Valid Bits

- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
  - Actually, only need the high-order bits
  - Called the tag

- What if there is no data in a location?
  - Valid bit: 1 = present, 0 = not present
  - Initially 0
Cache Example

- 8-blocks, 1 word/block, direct mapped
- Initial state

<table>
<thead>
<tr>
<th>Index</th>
<th>V</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Cache Example

<table>
<thead>
<tr>
<th>Word addr</th>
<th>Binary addr</th>
<th>Hit/miss</th>
<th>Cache block</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>10 110</td>
<td>Miss</td>
<td>110</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index</th>
<th>V</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>Y</td>
<td>10</td>
<td>Mem[10110]</td>
</tr>
<tr>
<td>111</td>
<td>N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Cache Example

<table>
<thead>
<tr>
<th>Word addr</th>
<th>Binary addr</th>
<th>Hit/miss</th>
<th>Cache block</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>11 010</td>
<td>Miss</td>
<td>010</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index</th>
<th>V</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>Y</td>
<td>11</td>
<td>Mem[11010]</td>
</tr>
<tr>
<td>011</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>Y</td>
<td>10</td>
<td>Mem[10110]</td>
</tr>
<tr>
<td>111</td>
<td>N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Cache Example

<table>
<thead>
<tr>
<th>Word addr</th>
<th>Binary addr</th>
<th>Hit/miss</th>
<th>Cache block</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>10 110</td>
<td>Hit</td>
<td>110</td>
</tr>
<tr>
<td>26</td>
<td>11 010</td>
<td>Hit</td>
<td>010</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index</th>
<th>V</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>Y</td>
<td>11</td>
<td>Mem[11010]</td>
</tr>
<tr>
<td>011</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>Y</td>
<td>10</td>
<td>Mem[10110]</td>
</tr>
<tr>
<td>111</td>
<td>N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Cache Example

<table>
<thead>
<tr>
<th>Word addr</th>
<th>Binary addr</th>
<th>Hit/miss</th>
<th>Cache block</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>10 000</td>
<td>Miss</td>
<td>000</td>
</tr>
<tr>
<td>3</td>
<td>00 011</td>
<td>Miss</td>
<td>011</td>
</tr>
<tr>
<td>16</td>
<td>10 000</td>
<td>Hit</td>
<td>000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index</th>
<th>V</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>Y</td>
<td>10</td>
<td>Mem[10000]</td>
</tr>
<tr>
<td>001</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>Y</td>
<td>11</td>
<td>Mem[11010]</td>
</tr>
<tr>
<td>011</td>
<td>Y</td>
<td>00</td>
<td>Mem[00011]</td>
</tr>
<tr>
<td>100</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>Y</td>
<td>10</td>
<td>Mem[10110]</td>
</tr>
<tr>
<td>111</td>
<td>N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Cache Example

<table>
<thead>
<tr>
<th>Word addr</th>
<th>Binary addr</th>
<th>Hit/miss</th>
<th>Cache block</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>10 010</td>
<td>Miss</td>
<td>010</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index</th>
<th>V</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>Y</td>
<td>10</td>
<td>Mem[10000]</td>
</tr>
<tr>
<td>001</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>010</strong></td>
<td>Y</td>
<td><strong>10</strong></td>
<td><strong>Mem[10010]</strong></td>
</tr>
<tr>
<td>011</td>
<td>Y</td>
<td>00</td>
<td>Mem[00011]</td>
</tr>
<tr>
<td>100</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>Y</td>
<td>10</td>
<td>Mem[10110]</td>
</tr>
<tr>
<td>111</td>
<td>N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Address Subdivision

Address (showing bit positions)

31 30 · · · 13 12 11 · · · 2 1 0

Byte offset

Index

Tag

Hit

Data

Valid Tag

Index

0

1

2

...

1021

1022

1023

20

32

(4KB cache; actually much larger if tag and valid bit are included)
Example: Larger Block Size

- 64 blocks, 16 bytes/block
  - To what block number does address 1200 map?
  - Block address = \( \lfloor \frac{1200}{16} \rfloor = 75 \)
  - Block number = 75 modulo 64 = 11

<table>
<thead>
<tr>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>22 bits</td>
<td>6 bits</td>
<td>4 bits</td>
</tr>
</tbody>
</table>

Chapter 5 — Large and Fast: Exploiting Memory Hierarchy — 30
Block Size Considerations

- Larger blocks should reduce miss rate
  - Due to spatial locality
  - In fixed-sized cache
    - Larger blocks imply fewer
    - More competition implies increased miss rate
    - Larger blocks imply pollution

- Larger miss penalty
  - Can override benefit of reduced miss rate
  - Early restart and critical-word-first can help

* (Resume execution as soon as the requested word is returned; used for _________ access)
Cache Misses

- On cache hit, CPU proceeds normally
- On cache miss
  - Stall the CPU pipeline
  - Fetch block from next level of hierarchy
  - Instruction cache miss *(______ read ➔ ______ update ➔ Refetch from ____*)
    - Restart instruction fetch
  - Data cache miss
    - Complete data access
Write-Through

- On data-write hit, could just update the block in cache
  - But then cache and memory would be inconsistent
- Write through: also update 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 \times 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
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
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
Example: Intrinsity FastMATH

- Embedded MIPS processor
  - 12-stage pipeline
  - Instruction and data access on each cycle
- Split cache: separate I-cache and D-cache
  - Each 16KB: 256 blocks × 16 words/block
  - D-cache: write-through or write-back
    ★ (___-entry write buffer)
- SPEC2000 miss rates
  - I-cache: 0.4%
  - D-cache: 11.4%
  - Weighted average: 3.2%
Example: Intrinsity FastMATH

(Split cache vs unified cache; _____% vs _____% miss rate; ______ bandwidth)
Main Memory Supporting Caches

- Use DRAMs for main memory
  - Fixed width (e.g., 1 word)
  - Connected by fixed-width clocked bus
    - Bus clock is typically slower than CPU clock

- Example cache block read
  - 1 bus cycle for address transfer
  - 15 bus cycles per DRAM access
  - 1 bus cycle per data transfer

- For 4-word block, 1-word-wide DRAM
  - Miss penalty = $1 + 4 \times 15 + 4 \times 1 = 65$ bus cycles
  - Bandwidth = $16$ bytes / $65$ cycles = $0.25$ B/cycle

Chapter 5 — Large and Fast: Exploiting Memory Hierarchy — 38
Components of CPU time

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

With simplifying assumptions:

\[
\text{Memory stall cycles}^* = \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}
\]

\[
= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}
\]

(Different for read and write due to write buffer _____)
Cache Performance Example

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: $0.02 \times 100 = 2$
- D-cache: $0.36 \times 0.04 \times 100 = 1.44$

Actual CPI = $2 + 2 + 1.44 = 5.44$
- Ideal CPU is $5.44/2 = 2.72$ times faster
Average Access Time

- Hit time is also important for performance

- Average memory access time (AMAT)
  - AMAT = Hit time + Miss rate \times Miss penalty

- Example
  - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, L-cache miss rate = 5%
  - AMAT = 1 + 0.05 \times 20 = 2ns
    - 2 cycles per instruction
Performance Summary

- When CPU performance increased
  - Miss penalty becomes more significant
- Decreasing base 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
Associative Caches

- Fully associative
  - Allow a given block to go in any cache entry
  - Requires all entries to be searched at once
  - Comparator per entry (expensive)

- \( n \)-way set associative
  - Each set contains \( n \) entries
  - Block number determines which set
    - (Block number) modulo (\#Sets in cache)
  - Search all entries in a given set at once
  - \( n \) comparators (less expensive)
Associative Cache Example

- **Direct mapped**
  - Block #: 0 1 2 3 4 5 6 7
  - Tag: 1 2

- **Set associative**
  - Set #: 0 1 2 3
  - Tag: 1 2

- **Fully associative**
  - Data
  - Tag: 1 2

Chapter 5 — Large and Fast: Exploiting Memory Hierarchy — 44
Spectrum of Associativity

For a cache with 8 entries

One-way set associative
(direct mapped)

<table>
<thead>
<tr>
<th>Block</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Two-way set associative

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Four-way set associative

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Eight-way set associative (fully associative)

<table>
<thead>
<tr>
<th></th>
<th>Tag</th>
<th>Data</th>
<th>Tag</th>
<th>Data</th>
<th>Tag</th>
<th>Data</th>
<th>Tag</th>
<th>Data</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* (Increasing the degree of associativity _________ miss rate but _________ hit time)
Associativity Example

- Compare 4-block caches
  - Direct mapped, 2-way set associative, fully associative
  - Block access sequence: 0, 8, 0, 6, 8

- Direct mapped

<table>
<thead>
<tr>
<th>Block address</th>
<th>Cache index</th>
<th>Hit/miss</th>
<th>Cache content after access</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>miss</td>
<td>Mem[0]</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>miss</td>
<td>Mem[8]</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>miss</td>
<td>Mem[0]</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>miss</td>
<td>Mem[0]</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>miss</td>
<td>Mem[8]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cache index</th>
<th>Hit/miss</th>
<th>Cache content after access</th>
<th>Cache content after access</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
## Associativity Example

### 2-way set associative

<table>
<thead>
<tr>
<th>Block address</th>
<th>Cache index</th>
<th>Hit/miss</th>
<th>Cache content after access</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Set 0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>miss</td>
<td>Mem[0]</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>miss</td>
<td>Mem[0] Mem[8]</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>hit</td>
<td>Mem[0] Mem[8] <em>(LRU)</em></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>miss</td>
<td>Mem[0] Mem[6] <em>(LRU)</em></td>
</tr>
</tbody>
</table>

### Fully associative

<table>
<thead>
<tr>
<th>Block address</th>
<th>Hit/miss</th>
<th>Cache content after access</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Set 0</td>
</tr>
<tr>
<td>0</td>
<td>miss</td>
<td>Mem[0]</td>
</tr>
<tr>
<td>8</td>
<td>miss</td>
<td>Mem[0] Mem[8]</td>
</tr>
<tr>
<td>0</td>
<td>hit</td>
<td>Mem[0] Mem[8]</td>
</tr>
</tbody>
</table>
How Much Associativity

- Increased associativity decreases miss rate
  - But with diminishing returns

Simulation of a system with 64KB D-cache, 16-word blocks, SPEC2000
- 1-way: 10.3%
- 2-way: 8.6%
- 4-way: 8.3%
- 8-way: 8.1%
Set Associative Cache Organization

Address
31 30 ⋯ 12 11 10 9 8 ⋯ 3 2 1 0

Index
0
1
2
253
254
255

V Tag Data

V Tag Data

V Tag Data

V Tag Data

22
8

4-to-1 multiplexer

Hit

Data

* ( ____-way set associative?)
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
Multilevel Caches

- Primary cache attached to CPU
  - Small, but fast
- Level-2 cache services misses from primary cache
  - Larger, slower, but still faster than main memory
- Main memory services L-2 cache misses
- Some high-end systems include L-3 cache

* (Itanium 9500(Poulson): ___-bit microprocessor, ___nm, _ core, L2 I ___KB/D ___KB per core, L3 ___B, ____ GHz CPU clock, ___-wide issue architecture, multithreading, virtualization, die size ___ mm²)
Multilevel Cache Example

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 \times 400 = 9
Example (cont.)

- 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 = 500 cycles
- CPI = \(1 + 0.02 \times 20 + 0.005 \times 400 = 3.4\)
- Performance ratio = \(9/3.4 = 2.6\)
Multilevel Cache Considerations

- 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

* (L-2 has higher associativity to reduce __________ )
Interactions with Advanced CPUs

- 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
Interactions with Software

- Misses depend on memory access patterns
  - Algorithm behavior
  - Compiler optimization for memory access
Software Optimization via Blocking

- **Goal:** maximize accesses to data before it is replaced

- **Consider inner loops of DGEMM:**

```c
for (int j = 0; j < n; ++j)
{
    double cij = C[i+j*n];
    for( int k = 0; k < n; k++ )
        cij += A[i+k*n] * B[k+j*n];
    C[i+j*n] = cij;
}
```
DGEMM Access Pattern

- C, A, and B arrays

![Diagram showing DGEMM access pattern]

Chapter 5 — Large and Fast: Exploiting Memory Hierarchy — 58
Cache Blocked DGEMM

```c
#define BLOCKSIZE 32

void do_block (int n, int si, int sj, int sk, double *A, double *B, double *C)
{
  for (int i = si; i < si+BLOCKSIZE; ++i)
    for (int j = sj; j < sj+BLOCKSIZE; ++j)
    {
      double cij = C[i+j*n]; /* cij = C[i][j] */
      for (int k = sk; k < sk+BLOCKSIZE; k++)
        cij += A[i+k*n] * B[k+j*n]; /* cij += A[i][k]*B[k][j] */
      C[i+j*n] = cij; /* C[i][j] = cij */
    }
}

void dgemm (int n, double* A, double* B, double* C)
{
  for (int sj = 0; sj < n; sj += BLOCKSIZE)
    for (int si = 0; si < n; si += BLOCKSIZE)
      for (int sk = 0; sk < n; sk += BLOCKSIZE)
        do_block(n, si, sj, sk, A, B, C);
```
Blocked DGEMM Access Pattern

★ (For ______ locality)
★ (For ______ locality)
★ (All 3 matrices fit in the ______)
★ (_______: maximizing memory performance at run time considering cache size, associativity, block size, number of caches)

Chapter 5 — Large and Fast: Exploiting Memory Hierarchy — 60
Dependability

- **Fault**: failure of a component
  - May or may not lead to system failure

- **Service accomplishment**: Service delivered as specified
- **Service interruption**: Deviation from specified service

Restoration → Failure

* (Permanent or ________ )
Dependability Measures

- Reliability: mean time to failure (MTTF)
- Service interruption: mean time to repair (MTTR)
- Mean time between failures
  - $MTBF = MTTF + MTTR$
- Availability = $\frac{MTTF}{MTTF + MTTR}$
- Improving Availability
  - Increase MTTF: fault avoidance, fault tolerance, fault forecasting
  - Reduce MTTR: improved tools and processes for diagnosis and repair
The Hamming SEC Code

- Hamming distance
  - Number of bits that are different between two bit patterns
- Minimum distance = 2 provides single bit error detection
  - E.g. parity code
- Minimum distance = 3 provides single error correction, 2 bit error detection
Encoding SEC

- To calculate Hamming code:
  - Number bits from 1 on the left
  - All bit positions that are a power 2 are parity bits
  - Each parity bit checks certain data bits:

```
<table>
<thead>
<tr>
<th>Bit position</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encoded date bits</td>
<td>p1</td>
<td>p2</td>
<td>d1</td>
<td>p4</td>
<td>d2</td>
<td>d3</td>
<td>d4</td>
<td>p8</td>
<td>d5</td>
<td>d6</td>
<td>d7</td>
<td>d8</td>
</tr>
<tr>
<td>Parity bit coverage</td>
<td>p1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>p2</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p4</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
```
Decoding SEC

- Value of parity bits indicates which bits are in error
  - Use numbering from encoding procedure
  - E.g.
    - Parity bits = 0000 indicates no error
    - Parity bits = 1010 indicates bit 10 was flipped
SEC/DEC Code

- Add an additional parity bit for the whole word ($p_n$)
- Make Hamming distance = 4
- Decoding:
  - Let $H = \text{SEC parity bits}$
    - $H$ even, $p_n$ even, no error
    - $H$ odd, $p_n$ odd, correctable single bit error
    - $H$ even, $p_n$ odd, error in $p_n$ bit
    - $H$ odd, $p_n$ even, double error occurred
- Note: ECC DRAM uses SEC/DEC with 8 bits protecting each 64 bits
Virtual Machines

- Host computer emulates guest operating system and machine resources
  - Improved isolation of multiple guests
  - Avoids security and reliability problems
  - Aids sharing of resources
- Virtualization has some performance impact
  - Feasible with modern high-performance computers
- Examples
  - IBM VM/370 (1970s technology!)
  - VMWare
  - Microsoft Virtual PC
Virtual Machine Monitor

- Maps virtual resources to physical resources
  - Memory, I/O devices, CPUs
- Guest code runs on native machine in user mode
  - Traps to VMM on privileged instructions and access to protected resources
- Guest OS may be different from host OS
- VMM handles real I/O devices
  - Emulates generic virtual I/O devices for guest
Example: Timer Virtualization

- In native machine, on timer interrupt
  - OS suspends current process, handles interrupt, selects and resumes next process

- With Virtual Machine Monitor
  - VMM suspends current VM, handles interrupt, selects and resumes next VM

- If a VM requires timer interrupts
  - VMM emulates a virtual timer
  - Emulates interrupt for VM when physical timer interrupt occurs
Instruction Set Support

- User and System modes
- Privileged instructions only available in system mode
  - Trap to system if executed in user mode
- All physical resources only accessible using privileged instructions
  - Including page tables, interrupt controls, I/O registers
- Renaissance of virtualization support
  - Current ISAs (e.g., x86) adapting
Virtual Memory

- Use main memory as a “cache” for secondary (disk) storage
  - Managed jointly by CPU hardware and the operating system (OS)

- Programs share main memory
  - Each gets a private virtual address space holding its frequently used code and data
  - Protected from other programs

- CPU and OS translate virtual addresses to physical addresses
  - VM “block” is called a page
  - VM translation “miss” is called a page fault
Address Translation

- Fixed-size pages (e.g., 4K)

Diagram:
- Virtual addresses
- Physical addresses
- Disk addresses
- Address translation
- Virtual address
- Virtual page number
- Page offset
- Physical page number
- Page offset
- Physical address

Chapter 5 — Large and Fast: Exploiting Memory Hierarchy — 72
Page Fault Penalty

- On page fault, the page must be fetched from disk
  - Takes millions of clock cycles
  - Handled by OS code
- Try to minimize page fault rate
  - Fully associative placement
  - Smart replacement algorithms
Page Tables

- Stores placement information
  - Array of page table entries, indexed by virtual page number
  - Page table register in CPU points to page table in physical memory

- If page is present in memory
  - PTE stores the physical page number
  - Plus other status bits (referenced, dirty, …)

- If page is not present
  - PTE can refer to location in swap space on disk
Translation Using a Page Table

The diagram illustrates the process of translating a virtual address to a physical address using a page table. The virtual address consists of a virtual page number and a page offset. The page table register contains a virtual address, which is broken down into a virtual page number and a page offset. The page table is used to map virtual pages to physical pages. If the valid bit is 0, it indicates that the page is not present in memory. The physical address is constructed by combining the physical page number and the page offset.
Mapping Pages to Storage

Virtual page number

Page table
Physical page or
disk address

Valid

Physical memory

Disk storage

Chapter 5 — Large and Fast: Exploiting Memory Hierarchy — 76
Replacement and Writes

- To reduce page fault rate, prefer least-recently used (LRU) replacement
  - Reference bit (aka use bit) in PTE set to 1 on access to page
  - Periodically cleared to 0 by OS
  - A page with reference bit = 0 has not been used recently

- Disk writes take millions of cycles
  - Block at once, not individual locations
  - Write through is impractical
  - Use write-back
  - Dirty bit in PTE set when page is written
Fast Translation Using a TLB

- Address translation would appear to require extra memory references
  - One to access the PTE
  - Then the actual memory access

- But access to page tables has good locality
  - So use a fast cache of PTEs within the CPU
  - Called a Translation Look-aside Buffer (TLB)
    - Typical: 16–512 PTEs, 0.5–1 cycle for hit, 10–100 cycles for miss, 0.01%–1% miss rate
  - Misses could be handled by hardware or software
Fast Translation Using a TLB

Virtual page number | Valid Dirty Ref | Tag | Physical page address
--- | --- | --- | ---
1 0 1
1 1 1
1 1 1
1 0 1
0 0 0
1 0 1

Page table

Physical memory

Disk storage

Chapter 5 — Large and Fast: Exploiting Memory Hierarchy — 79
TLB Misses

- If page is in memory
  - Load the PTE from memory and retry
  - Could be handled in hardware
    - Can get complex for more complicated page table structures
  - Or in software
    - Raise a special exception, with optimized handler

- If page is not in memory (page fault)
  - OS handles fetching the page and updating the page table
  - Then restart the faulting instruction
TLB Miss Handler

- TLB miss indicates
  - Page present, but PTE not in TLB
  - Page not present
- Must recognize TLB miss before destination register overwritten
  - Raise exception
- Handler copies PTE from memory to TLB
  - Then restarts instruction
  - If page not present, page fault will occur
Page Fault Handler

- Use faulting virtual address to find PTE
- Locate page on disk
- Choose page to replace
  - If dirty, write to disk first
- Read page into memory and update page table
- Make process runnable again
  - Restart from faulting instruction
**TLB and Cache Interaction**

- If cache tag uses physical address
  - Need to translate before cache lookup
- Alternative: use virtual address tag
  - Complications due to aliasing
    - Different virtual addresses for shared physical address
Memory Protection

- Different tasks can share parts of their virtual address spaces
  - But need to protect against errant access
  - Requires OS assistance

- Hardware support for OS protection
  - Privileged supervisor mode (aka kernel mode)
  - Privileged instructions
  - Page tables and other state information only accessible in supervisor mode
  - System call exception (e.g., syscall in MIPS)
The Memory Hierarchy

The BIG Picture

- Common principles apply at all levels of the memory hierarchy
  - Based on notions of caching
- At each level in the hierarchy
  - Block placement
  - Finding a block
  - Replacement on a miss
  - Write policy
Block Placement

- Determined by associativity
  - Direct mapped (1-way associative)
    - One choice for placement
  - n-way set associative
    - n choices within a set
  - Fully associative
    - Any location

- Higher associativity reduces miss rate
  - Increases complexity, cost, and access time
## Finding a Block

<table>
<thead>
<tr>
<th>Associativity</th>
<th>Location method</th>
<th>Tag comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct mapped</td>
<td>Index</td>
<td>1</td>
</tr>
<tr>
<td>n-way set associative</td>
<td>Set index, then search entries within the set</td>
<td>n</td>
</tr>
<tr>
<td>Fully associative</td>
<td>Search all entries</td>
<td>#entries</td>
</tr>
<tr>
<td></td>
<td>Full lookup table</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Hardware caches**
  - Reduce comparisons to reduce cost

- **Virtual memory**
  - Full table lookup makes full associativity feasible
  - Benefit in reduced miss rate
Replacement

- Choice of entry to replace on a miss
  - Least recently used (LRU)
    - Complex and costly hardware for high associativity
  - Random
    - Close to LRU, easier to implement
- Virtual memory
  - LRU approximation with hardware support
Write Policy

- Write-through
  - Update both upper and lower levels
  - Simplifies replacement, but may require write buffer

- Write-back
  - Update upper level only
  - Update lower level when block is replaced
  - Need to keep more state

- Virtual memory
  - Only write-back is feasible, given disk write latency
Sources of Misses

- Compulsory misses (aka cold start misses)
  - First access to a block

- Capacity misses
  - Due to finite cache size
  - A replaced block is later accessed again

- Conflict misses (aka collision misses)
  - In a non-fully associative cache
  - Due to competition for entries in a set
  - Would not occur in a fully associative cache of the same total size
## Cache Design Trade-offs

<table>
<thead>
<tr>
<th>Design change</th>
<th>Effect on miss rate</th>
<th>Negative performance effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Increase cache size</td>
<td>Decrease capacity misses</td>
<td>May increase access time</td>
</tr>
<tr>
<td>Increase associativity</td>
<td>Decrease conflict misses</td>
<td>May increase access time</td>
</tr>
<tr>
<td>Increase block size</td>
<td>Decrease compulsory misses</td>
<td>Increases miss penalty. For very large block size, may increase miss rate due to pollution.</td>
</tr>
</tbody>
</table>
Cache Control

- Example cache characteristics
  - Direct-mapped, write-back, write allocate
  - Block size: 4 words (16 bytes)
  - Cache size: 16 KB (1024 blocks)
  - 32-bit byte addresses
  - Valid bit and dirty bit per block
  - Blocking cache
    - CPU waits until access is complete

<table>
<thead>
<tr>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>18 bits</td>
<td>10 bits</td>
<td>4 bits</td>
</tr>
</tbody>
</table>
Interface Signals

Multiple cycles per access

Chapter 5 — Large and Fast: Exploiting Memory Hierarchy — 93
Finite State Machines

- Use an FSM to sequence control steps
- Set of states, transition on each clock edge
  - State values are binary encoded
  - Current state stored in a register
  - Next state \( = f_n \) (current state, current inputs)
- Control output signals \( = f_o \) (current state)
Cache Controller FSM

Chapter 5 — Large and Fast: Exploiting Memory Hierarchy — 95
## Cache Coherence Problem

Suppose two CPU cores share a physical address space
- Write-through caches

<table>
<thead>
<tr>
<th>Time step</th>
<th>Event</th>
<th>CPU A’s cache</th>
<th>CPU B’s cache</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>CPU A reads X</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>CPU B reads X</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>CPU A writes 1 to X</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Coherence Defined

- Informally: Reads return most recently written value
- Formally:
  - P writes X; P reads X (no intervening writes) ⇒ read returns written value
  - P₁ writes X; P₂ reads X (sufficiently later) ⇒ read returns written value
    - c.f. CPU B reading X after step 3 in example
  - P₁ writes X, P₂ writes X ⇒ all processors see writes in the same order
    - End up with the same final value for X
Cache Coherence Protocols

- Operations performed by caches in multiprocessors to ensure coherence
  - Migration of data to local caches
    - Reduces bandwidth for shared memory
  - Replication of read-shared data
    - Reduces contention for access
- Snooping protocols
  - Each cache monitors bus reads/writes
- Directory-based protocols
  - Caches and memory record sharing status of blocks in a directory
Invalidating Snooping Protocols

- Cache gets exclusive access to a block when it is to be written
  - Broadcasts an invalidate message on the bus
  - Subsequent read in another cache misses
    - Owning cache supplies updated value

<table>
<thead>
<tr>
<th>CPU activity</th>
<th>Bus activity</th>
<th>CPU A’s cache</th>
<th>CPU B’s cache</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>CPU A reads X</td>
<td>Cache miss for X</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>CPU B reads X</td>
<td>Cache miss for X</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CPU A writes 1 to X</td>
<td>Invalidate for X</td>
<td>1</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>CPU B read X</td>
<td>Cache miss for X</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Memory Consistency

When are writes seen by other processors
- “Seen” means a read returns the written value
- Can’t be instantaneously

Assumptions
- A write completes only when all processors have seen it
- A processor does not reorder writes with other accesses

Consequence
- P writes X then writes Y
  - all processors that see new Y also see new X
- Processors can reorder reads, but not writes
## Multilevel On-Chip Caches

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>ARM Cortex-A8</th>
<th>Intel Nehalem</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache organization</td>
<td>Split instruction and data caches</td>
<td>Split instruction and data caches</td>
</tr>
<tr>
<td>L1 cache size</td>
<td>32 KiB each for instructions/data</td>
<td>32 KiB each for instructions/data per core</td>
</tr>
<tr>
<td>L1 cache associativity</td>
<td>4-way (I), 4-way (D) set associative</td>
<td>4-way (I), 8-way (D) set associative</td>
</tr>
<tr>
<td>L1 replacement</td>
<td>Random</td>
<td>Approximated LRU</td>
</tr>
<tr>
<td>L1 block size</td>
<td>64 bytes</td>
<td>64 bytes</td>
</tr>
<tr>
<td>L1 write policy</td>
<td>Write-back, Write-allocate(?)</td>
<td>Write-back, No-write-allocate</td>
</tr>
<tr>
<td>L1 hit time (load-use)</td>
<td>1 clock cycle</td>
<td>4 clock cycles, pipelined</td>
</tr>
<tr>
<td>L2 cache organization</td>
<td>Unified (instruction and data)</td>
<td>Unified (instruction and data) per core</td>
</tr>
<tr>
<td>L2 cache size</td>
<td>128 KiB to 1 MiB</td>
<td>256 KiB (0.25 MiB)</td>
</tr>
<tr>
<td>L2 cache associativity</td>
<td>8-way set associative</td>
<td>8-way set associative</td>
</tr>
<tr>
<td>L2 replacement</td>
<td>Random(?)</td>
<td>Approximated LRU</td>
</tr>
<tr>
<td>L2 block size</td>
<td>64 bytes</td>
<td>64 bytes</td>
</tr>
<tr>
<td>L2 write policy</td>
<td>Write-back, Write-allocate (?)</td>
<td>Write-back, Write-allocate</td>
</tr>
<tr>
<td>L2 hit time</td>
<td>11 clock cycles</td>
<td>10 clock cycles</td>
</tr>
<tr>
<td>L3 cache organization</td>
<td>–</td>
<td>Unified (instruction and data)</td>
</tr>
<tr>
<td>L3 cache size</td>
<td>–</td>
<td>8 MiB, shared</td>
</tr>
<tr>
<td>L3 cache associativity</td>
<td>–</td>
<td>16-way set associative</td>
</tr>
<tr>
<td>L3 replacement</td>
<td>–</td>
<td>Approximated LRU</td>
</tr>
<tr>
<td>L3 block size</td>
<td>–</td>
<td>64 bytes</td>
</tr>
<tr>
<td>L3 write policy</td>
<td>–</td>
<td>Write-back, Write-allocate</td>
</tr>
<tr>
<td>L3 hit time</td>
<td>–</td>
<td>35 clock cycles</td>
</tr>
</tbody>
</table>
## 2-Level TLB Organization

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>ARM Cortex-A8</th>
<th>Intel Core i7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual address</td>
<td>32 bits</td>
<td>48 bits</td>
</tr>
<tr>
<td>Physical address</td>
<td>32 bits</td>
<td>44 bits</td>
</tr>
<tr>
<td>Page size</td>
<td>Variable: 4, 16, 64 KiB, 1, 16 MiB</td>
<td>Variable: 4 KiB, 2/4 MiB</td>
</tr>
<tr>
<td>TLB organization</td>
<td>1 TLB for instructions and 1 TLB for data</td>
<td>1 TLB for instructions and 1 TLB for data per core</td>
</tr>
<tr>
<td></td>
<td>Both TLBs are fully associative, with 32 entries, round robin replacement</td>
<td>Both L1 TLBs are four-way set associative, LRU replacement</td>
</tr>
<tr>
<td></td>
<td>TLB misses handled in hardware</td>
<td>L1 I-TLB has 128 entries for small pages, 7 per thread for large pages</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L1 D-TLB has 64 entries for small pages, 32 for large pages</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The L2 TLB is four-way set associative, LRU replacement</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The L2 TLB has 512 entries</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TLB misses handled in hardware</td>
</tr>
</tbody>
</table>
Supporting Multiple Issue

- Both have multi-banked caches that allow multiple accesses per cycle assuming no bank conflicts

- Core i7 cache optimizations
  - Return requested word first
  - Non-blocking cache
    - Hit under miss
    - Miss under miss
  - Data prefetching
Combining cache blocking and subword parallelism can improve performance in DGEMM calculations. The chart illustrates the GFLOPS for different block sizes and optimization levels, showing significant improvements with cache blocking and unrolling techniques.
Pitfalls

- Byte vs. word addressing
  - Example: 32-byte direct-mapped cache, 4-byte blocks
    - Byte 36 maps to block 1
    - Word 36 maps to block 4

- Ignoring memory system effects when writing or generating code
  - Example: iterating over rows vs. columns of arrays
  - Large strides result in poor locality
In multiprocessor with shared L2 or L3 cache

- Less associativity than cores results in conflict misses
- More cores $\Rightarrow$ need to increase associativity

Using AMAT to evaluate performance of out-of-order processors

- Ignores effect of non-blocked accesses
- Instead, evaluate performance by simulation
Pitfalls

- Extending address range using segments
  - E.g., Intel 80286
  - But a segment is not always big enough
  - Makes address arithmetic complicated

- Implementing a VMM on an ISA not designed for virtualization
  - E.g., non-privileged instructions accessing hardware resources
  - Either extend ISA, or require guest OS not to use problematic instructions
Concluding Remarks

- Fast memories are small, large memories are slow
  - We really want fast, large memories 😞
  - Caching gives this illusion 😊

- Principle of locality
  - Programs use a small part of their memory space frequently

- Memory hierarchy
  - L1 cache ↔ L2 cache ↔ … ↔ DRAM memory ↔ disk

- Memory system design is critical for multiprocessors