Measuring Cache Performance

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

With simplifying assumptions:

Memory stall cycles

\[
\text{Memory accesses per program} \times \text{Miss rate} \times \text{Miss penalty} = \frac{\text{Instructions per program}}{\text{Instruction}} \times \frac{\text{Misses per instruction}}{\text{Instruction}} \times \text{Miss penalty}
\]
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 × Miss penalty
- Example
  - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, I-cache miss rate = 5%
  - AMAT = 1 + 0.05 × 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
    - \((\text{Block number}) \mod (\text{#Sets in cache})\)
  - Search all entries in a given set at once
  - \( n \) comparators (less expensive)
Associative Cache Example

Direct mapped

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

Set associative

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

Fully associative

| Data    |   |   |   |   |
| Tag     | 1 | 2 |   |   |
| Search  |   |   |   |   |
## 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>
<th>Tag</th>
<th>Data</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Eight-way set associative (fully associative)

<table>
<thead>
<tr>
<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>
<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>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 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></td>
<td></td>
<td></td>
<td>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[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>
</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]</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>hit</td>
<td>Mem[0]</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>miss</td>
<td>Mem[0]</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]</td>
</tr>
<tr>
<td>0</td>
<td>hit</td>
<td>Mem[0]</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
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
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 × 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 × 20 + 0.005 × 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
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
Nehalem Example

- 731M transistors (quad core)
- 32kB L1 Instruction Cache
- 32kB L1 Data Cache
- 256kB L2 cache / core
- 4-12MB L3 cache (shared)

- 12MB -> 576M Transistors!
Multilevel On-Chip Caches

Intel Nehalem 4-core processor

Per core: 32KB L1 I-cache, 32KB L1 D-cache, 512KB L2 cache
# 3-Level Cache Organization

<table>
<thead>
<tr>
<th></th>
<th>Intel Nehalem</th>
<th>AMD Opteron X4</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>L1 caches</strong></td>
<td>L1 I-cache: 32KB, 64-byte blocks, 4-way, approx LRU replacement, hit time n/a</td>
<td>L1 I-cache: 32KB, 64-byte blocks, 2-way, LRU replacement, hit time 3 cycles</td>
</tr>
<tr>
<td>(per core)</td>
<td>L1 D-cache: 32KB, 64-byte blocks, 8-way, approx LRU replacement, write-back/ allocate, hit time n/a</td>
<td>L1 D-cache: 32KB, 64-byte blocks, 2-way, LRU replacement, write-back/ allocate, hit time 9 cycles</td>
</tr>
<tr>
<td></td>
<td>256KB, 64-byte blocks, 8-way, approx LRU replacement, write-back/allocate, hit time n/a</td>
<td>512KB, 64-byte blocks, 16-way, approx LRU replacement, write-back/allocate, hit time n/a</td>
</tr>
<tr>
<td><strong>L2 unified cache</strong></td>
<td>8MB, 64-byte blocks, 16-way, replacement n/a, write-back/allocate, hit time n/a</td>
<td>2MB, 64-byte blocks, 32-way, replace block shared by fewest cores, write-back/allocate, hit time 32 cycles</td>
</tr>
<tr>
<td>(per core)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>L3 unified cache</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(shared)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>n/a: data not available</td>
<td></td>
</tr>
</tbody>
</table>

Chapter 5 — Large and Fast: Exploiting Memory Hierarchy — 21
# 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>
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)
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 (PTE)

- 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

Virtual address

31 30 29 28 27 15 14 13 12 11 10 9 8 3 2 1 0

Virtual page number
Page offset

Valid

Physical page number

If 0 then page is not present in memory

Physical address

Page table register

Page table

If 0 then page is not present in memory

Physical page number
Page offset

Chapter 5 — Large and Fast: Exploiting Memory Hierarchy — 27
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 — 28
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  

TLB  

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  

Physical memory  

Page table  

Valid Dirty Ref  

Physical page or disk address  

1 0 1  
1 0 0  
1 0 0  
1 0 1  
0 0 0  
1 0 1  
1 0 1  
1 0 1  
0 0 0  
1 1 1  
1 1 1  
1 1 1  
0 0 0  
1 1 1  

Disk storage  

Chapter 5 — Large and Fast: Exploiting Memory Hierarchy — 31
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 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
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
Simple ECE Review (will be on midterm #2)

- Power derivation: (energy dissipated / time)
- Why $P = C \cdot V_{dd}^2 \cdot f$ (dynamic energy)
  - $E = P \cdot t$
  - $I_{TOT} = I_{STAT} + I_{DYN}$
  - $I_{STAT}$ (leakage current; do nothing -> still waste)
  - $I_{DYN}$ (switching current; move charge to/from inverter)
  - $I_{DYN} = C \cdot \frac{dV}{dt}$ (change in capacitance charge/versus time) ($Q=CV$)
  - $P_{DYN} = I_{DYN} \cdot V = V \cdot C \cdot \frac{dV}{dt}$
  - $E_{DYN} = P_{DYN} \cdot dt = C \cdot V \cdot dV$
  - Now integrate
    - $E_{DYN} = P_{DYN} \cdot t = C \cdot V^2 / 2$

- What about cycle-time? How affected by $V_{dd}$?