ECE/CS 472/572
Computer Architecture:
Cache Advanced

Prof. Lizhong Chen
Spring 2017
Measuring Cache Performance

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

With simplifying assumptions:

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

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

- Calculate actual CPI considering miss penalty

  **Given**
  - I-cache miss rate = 2%
  - Miss penalty = 100 cycles
  - D-cache miss rate = 4%
  - 36% of instructions are load & store
  - Base CPI (ideal cache) = 2

- Miss cycles per instruction
  - I-cache: \(0.02 \times 100 = 2\)
  - D-cache: \(0.36 \times 0.04 \times 100 = 1.44\)

- Actual CPI = \(2 + 2 + 1.44 = 5.44\)
  - Perfect Cache 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: AMAT for instruction cache**
  - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, average cache miss rate = 5%
  - AMAT = \((1 + 0.05 \times 20) \times 1\text{ns}\) = 2ns
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
Associative Cache Example

**Direct mapped**

- **Block #**
  - 0, 1, 2, 3, 4, 5, 6, 7
- **Data**
- **Tag**
- **Search**

**Set associative**

- **Set #**
  - 0, 1, 2, 3
- **Data**
- **Tag**
- **Search**

**Fully associative**

- **Data**
- **Tag**
- **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>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></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>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</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>
</tr>
</thead>
</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>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>Block address</th>
<th>Cache index</th>
<th>Hit/miss</th>
<th>Cache content after access</th>
</tr>
</thead>
</table>
## Associativity Example

### 2-way set associative (LRU)

<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><strong>Set 0</strong></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>miss</td>
<td><strong>Mem[0]</strong></td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>miss</td>
<td><strong>Mem[0]</strong> <strong>Mem[8]</strong></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>hit</td>
<td><strong>Mem[0]</strong> <strong>Mem[8]</strong></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>miss</td>
<td><strong>Mem[0]</strong> <strong>Mem[6]</strong></td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>miss</td>
<td><strong>Mem[8]</strong> <strong>Mem[6]</strong></td>
</tr>
</tbody>
</table>

### Fully associative (LRU)

<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><strong>Mem[0]</strong></td>
</tr>
<tr>
<td>0</td>
<td>miss</td>
<td><strong>Mem[0]</strong> <strong>Mem[8]</strong></td>
</tr>
<tr>
<td>0</td>
<td>hit</td>
<td><strong>Mem[0]</strong> <strong>Mem[8]</strong></td>
</tr>
<tr>
<td>6</td>
<td>miss</td>
<td><strong>Mem[0]</strong> <strong>Mem[8]</strong> <strong>Mem[6]</strong></td>
</tr>
<tr>
<td>8</td>
<td>hit</td>
<td><strong>Mem[0]</strong> <strong>Mem[8]</strong> <strong>Mem[6]</strong></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%
- Other overhead of increasing associativity?
4-way Set Associative, 1-word Block Size

- 4KB cache
Replacement Policy

- Direct mapped: no choice
- Set associative
  - Prefer non-valid entry, if there is one
  - Otherwise, choose among entries in the set
- Optimal (OPT): oracle & mirror future accesses
- 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 ($L1)
  - Small, but fast
- Level-2 cache services misses from primary cache ($L2)
  - Larger, slower, but still faster than main memory
- Main memory services L2 cache misses
- Some high-end systems include L3 cache
Multilevel Cache Example

- **Given**
  - CPU base CPI = 1, clock rate = 4GHz
  - Miss rate/instruction = 2%
  - Main memory access time = 100ns

- **Actual CPI with only primary cache:**
  - Miss penalty = 100ns/0.25ns = 400 cycles
  - Effective CPI = 1 + 0.02 × 400 = 9
Multilevel Cache 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 = 400 cycles
- CPI = \(1 + 0.02 \times 20 + 0.005 \times 400 = 3.4\)
- Speedup = \(9/3.4 = 2.6\)
Multilevel Cache Considerations

- Primary cache
  - Focus on minimizing hit time

- L-2 cache
  - Focus on low miss rate to avoid DRAM access
  - Hit time has less overall impact

- Results
  - L-1: small and fast
  - L-2: large and slow
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 queues
    - Independent instructions continue

- Effect of miss depends on program data flow
  - Much harder to analyze
  - Use system simulation
    - e.g., gem5: full-system simulator
Interactions with Software

- Standard algorithm analysis often ignores the impact of memory hierarchy.
Software Optimization via Blocking

- Goal: maximize accesses to data before it is replaced
- Consider inner loops of DGEMM:
  - Double-precision GEneral Matrix Multiply (DGEMM)

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

- **C, A, and B arrays**

![Access Pattern Diagram](image)

- Older accesses
- New accesses
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
Blocked DGEMM Access Pattern

- Optimized (Blocked): less than 10% slowdown even with 900 times larger
Sources of Misses: 3Cs

- Cold misses
  - First access to a block

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

- Conflict 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 cold misses</td>
<td>Increases miss penalty. For very large block size, may increase miss rate due to pollution.</td>
</tr>
</tbody>
</table>
## Multilevel On-Chip Caches

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

- Cache performance
  - I/D, miss rate, miss penalty
- Using associativity to reduce miss rate
- Using multilevel cache to reduce miss penalty
- Software optimizations to improve the effectiveness of caches