Lecture 13: Cache Basics and Cache Performance

Computer Engineering 585
Fall 2002

What Is Memory Hierarchy

A typical memory hierarchy today:

- Proc/Regs
- L1-Cache
- L2-Cache
- L3-Cache (optional)
- Memory
- Disk, Tape, etc.

Here we focus on L1/L2/L3 caches and main memory

Why Memory Hierarchy?

1980: no cache in µproc; 1995 2-level cache on chip
(1989 first Intel µproc with a cache on chip)

Processor-Memory Performance Gap:
grows 50% / year

µProc 60%/yr.

DRAM 7%/yr.

Generations of Microprocessors

- Time of a full cache miss in instructions executed:
  - 1st Alpha: 340 ns/5.0 ns = 68 clks x 2 or 136
  - 2nd Alpha: 266 ns/3.3 ns = 80 clks x 4 or 320
  - 3rd Alpha: 180 ns/1.7 ns = 108 clks x 6 or 648
- L2X latency x 3X clock rate x 3XInstr/clock ⇒ 4.5X

Area Costs of Caches

<table>
<thead>
<tr>
<th>Processor</th>
<th>% Area</th>
<th>% Transistors (-cost)</th>
<th>% Transistors (-power)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel 80386</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>Alpha 21164</td>
<td>37%</td>
<td>77%</td>
<td>20%</td>
</tr>
<tr>
<td>StrongArm SA110</td>
<td>61%</td>
<td>94%</td>
<td>31%</td>
</tr>
<tr>
<td>Pentium Pro</td>
<td>64%</td>
<td>88%</td>
<td>30%</td>
</tr>
<tr>
<td>Itanium</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 2 dies per package: Proc/I$/D$ + L2$
- Itanium 92%
- Caches store redundant data only to close performance gap

What Is Exactly Cache?

- Small, fast storage used to improve average access time to slow memory; usually made by SRAM
- Exploits locality: spatial and temporal
- In computer architecture, almost everything is a cache!
  - Register file is the fastest place to cache variables
  - First-level cache a cache on second-level cache
  - Second-level cache a cache on memory
  - Memory a cache on disk (virtual memory)
  - TLB a cache on page table
  - Branch-prediction a cache on prediction information
  - Branch-target buffer can be implemented as cache
  - Beyond architecture: file cache, browser cache, proxy cache
- Here we focus on L1 and L2 caches (L3 optional) as buffers to main memory
Example: 1 KB Direct Mapped Cache

- Assume a cache of \(2^n\) bytes, \(2^k\) blocks, block size of \(2^m\) bytes; \(N = M \times K\) (#block times block size)
- \((2^N - N)\)-bit cache tag, \(K\)-bit cache index, and \(M\)-bit cache
- The cache stores tag, data, and valid bit for each block
  - Cache index is used to select a block in SRAM (Recall LUT, BTB)
  - Block tag is compared with the input tag
  - A word in the data block may be selected as the output

<table>
<thead>
<tr>
<th>Block address</th>
<th>Tag</th>
<th>Example Cell</th>
<th>Index</th>
<th>Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>0000</td>
</tr>
<tr>
<td>0001</td>
<td>01</td>
<td>01</td>
<td>01</td>
<td>0001</td>
</tr>
<tr>
<td>0010</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>0010</td>
</tr>
<tr>
<td>0011</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>0011</td>
</tr>
</tbody>
</table>

Four Questions About Cache Design

- Block placement: Where can a block be placed?
- Block identification: How to find a block in the cache?
- Block replacement: If a new block is to be fetched, which of existing blocks to replace? (If there are multiple choices)
- Write policy: What happens on a write?

Where Can A Block Be Placed

- What is a block: divide memory space into blocks as cache is divided
  - A memory block is the basic unit to be cached
- Direct mapped cache: there is only one place in the cache to buffer a given memory block
- \(N\)-way set associative cache: \(N\) places for a given memory block
  - Like \(N\) direct mapped caches operating in parallel
  - Reducing miss rates with increased complexity, cache access time, and power consumption
- Fully associative cache: a memory block can be put anywhere in the cache

Set Associative Cache

- Example: Two-way set associative cache
  - Cache index selects a set of two blocks
  - The two tags in the set are compared to the input in parallel
  - Data is selected based on the tag comparison
- Set associative or direct mapped? Discuss later

How to Find a Cached Block

- Direct mapped cache: the stored tag for the cache block matches the input tag
- Fully associative cache: any of the stored \(N\) tags matches the input tag
- Set associative cache: any of the stored \(K\) tags for the cache set matches the input tag

Cache hit latency is decided by both tag comparison and data access

Which Block to Replace?

- Direct mapped cache: Not an issue
- For set associative or fully associative* cache:
  - Random: Select candidate blocks randomly from the cache set
  - LRU (Least Recently Used): Replace the block that has been unused for the longest time
  - FIFO (First In, First Out): Replace the oldest block
- Usually LRU performs the best, but hard (and expensive) to implement

*Think fully associative cache as a set associative one with a single set
What Happens on Writes
Where to write the data if the block is found in cache?
- Write through: new data is written to both the cache block and the lower-level memory
  - Help to maintain cache consistency
- Write back: new data is written only to the cache block
  - Lower-level memory is updated when the block is replaced
  - A dirty bit is used to indicate the necessity
  - Help to reduce memory traffic
What happens if the block is not found in cache?
- Write allocate: fetch the block into cache, then write the data (usually combined with write back)
- No-write allocate: Do not fetch the block into cache (usually combined with write through)

Real Example: Alpha 21264 Caches
- 64KB 2-way associative instruction cache
- 64KB 2-way associative data cache

Cache performance
- Calculate average memory access time (AMAT)
  \[ AMAT = \text{hit time} + \text{miss rate} \times \text{miss penalty} \]
- Example: hit time = 1 cycle, miss time = 100 cycles, miss rate = 4%, then AMAT = 1 + 100\times 0.04 \times 5
- Calculate cache impact on processor performance
  \[ \text{CPU time} = \text{CPU execution cycles} + \text{Memory stall cycles} \times \text{Cycle time} \]
  \[ \text{CPU time} = \text{CPU cycles} \times \left( \text{Memory Stall Cycles} / \text{Instruction} \right) / \text{Cycle Time} \]
  - Note cycles spent on cache hit is usually counted into execution cycles
- If clock cycle is identical, better AMAT means better performance

Example: Evaluating Split Inst/Data Cache
- Unified vs Split Inst/data cache (Harvard Architecture)

<table>
<thead>
<tr>
<th>Pre</th>
<th>Def</th>
<th>Data1</th>
<th>Pre</th>
<th>Def</th>
<th>Data2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unified Cache 1</td>
<td>Unified Cache 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example on page 406/407
- Assume 36% data ops = 74% accesses from instructions (1.0/1.36)
- 16KB I:O: Inst miss rate=0.4%, data miss rate=11.4%, overall 3.24%
- 33KB unified: Aggregate miss rate=3.18%
- Which design is better?
  - Hit time=1, miss time=100
  - Note that data hit has stall for unified cache (only one port)

\[ \text{AMAT}_{\text{unified}} = 0.04 \times 100 + 0.96 \times 11.4 \times 100 = 4.24 \]
\[ \text{AMAT}_{\text{unified}} = 0.04 \times 100 + 0.96 \times 11.4 \times 100 = 4.44 \]

Disadvantage of Set Associative Cache
- Compare n-way set associative with direct mapped cache:
  - Has n comparators vs. 1 comparator
  - Has Extra MUX delay for the data
  - Data comes after hit/miss decision and set selection
  - In a direct mapped cache, cache block is available before hit/miss decision
  - Use the data assuming the access is a hit, recover if found otherwise
Example: Evaluating Set Associative Cache

Suppose a processor with
- 16GHz speed, Ideal (no misses) CPI = 2.0
- 1.5 memory references per instruction
- Two cache organization alternatives
  - Direct mapped, 1.4% miss rate, hit time 1 cycle, miss penalty 75ns
  - 2-way set associative, 1.0% miss rate, increase cycle time by 1.25x, hit time 1 cycle, miss penalty 75ns
- Performance evaluation by AMAT
  - Direct mapped: 10 * (0.014 x 75) = 0.205ns
  - 2-way set associative: 10 x 125 * (0.10 x 75) = 2.00ns
- Performance evaluation by CPU time
  - CPU Time 1 - IC x (2 x 1.0 x (1.5 x 0.014 x 75)) = 3.58 IC
  - CPU Time 2 - IC x (2 x 10 x 1.25 x 1.50 x 0.010 x 75) = 3.63 IC
- Better AMAT does not indicate better CPU time, since non-memory instructions are penalized.

Evaluating Cache Performance for Out-of-order Processors

Recall AMAT = hit time x miss rate x miss penalty
- Very difficult to define miss penalty to fit in this simple model, in the context of OOO processors
  - Consider overlapping between computation and memory accesses
  - Consider overlapping among memory accesses for more than one misses
- We may assume a certain percentage of overlapping
  - In practice, the degree of overlapping varies significantly
  - There are techniques to increase the overlapping, making the cache performance even unpredictable
- Cache hit time can also be overlapped
  - The increase of CPI is usually not counted in memory stall time

Simple Example

Consider an OOO processor into the previous example (slide 18)
- Slow clock (1.25x base cycle time)
- Direct mapped cache
  - Overlapping degree of 30%
  - Average miss penalty = 70% * 75ns = 52.5ns
  - AMAT = 1.0 x 1.25 * (0.014 x 52.5) = 1.99ns
  - CPU time = IC x (2 x 1.0 x 1.25 * (1.5 x 0.014 x 52.5)) = 3.60 x IC

Compare: 3.56 for in-order + direct mapped, 3.63 for in-order + two-way associative

This is only a simplified example; ideal CPI could be improved by OOO execution