COMPUTER ARCHITECTURE

Cache Memory
Contents

- Introduction.
- Cache performance.
- Trade-offs in cache design.
- Basic cache optimizations.
Performance evolution (1/latency)

- Memory
- Processor

- 25% year
- 52% year
- 20% year
- 0% year
- DRAM: 7% year

Computer Architecture - 2014 - ARCOS@uc3m
- **Intel Core i7**
  - 2 data access (64 bits) per cycle.
  - 4 cores, 3.2 GHz → 25.6 x 10⁹ access/sec
  - Instruction demand: 12.8 x 10⁹ of 128 bits.
  - Peak bandwidth: 409.6 GB/sec

- **SDRAM Memory**
  - DDR2 (2003): 3.2 GB/sec – 8.5 GB/sec
  - DDR3 (2007): 6.4 GB/sec – 17.06 GB/sec
  - DDR4 (2014?): 17.05 GB/sec – 25.6 GB/sec

- **Solutions:**
  - Multi-gate memory, pipelined caches, multi-level caches, per-core caches, instruction/data separation.
Locality principle:

- Program property exploited in hardware design.
- Programs access to a relatively small portion of address space.

Types of locality:

- Temporal locality: Recently accessed elements tend to be accessed again.
  - Examples: loops, variable reuse, ...

- Spatial locality: Elements next to a recently accessed element tend to be accessed in near future.
  - Examples: sequential instruction execution, arrays, ...

Computer Architecture - 2014 - ARCOS@uc3m
SRAM – Static RAM
- Access time: 0.5 ns – 2.5 ns
- Cost per GB: 2000$ - 5000$

DRAM – Dynamic RAM
- Access time: 50ns – 70 ns
- Cost per GB: 20$ - 75$

Magnetic disk
- Access time: 5,000,000 ns – 20,000,000 ns
- Cost per GB: 0.20 $ - 2$
- **Block or line**: Unit of copy.
  - Usually several words.

- If accessed data present in higher level:
  - **Hit**: Delivered by higher level.
    - Hit rate = Hits / accesses.

- If accessed data is missing:
  - **Miss**: Block copied from lower level.
  - Needed time → Miss penalty.
  - Miss rate = Misses / accesses = 1 – Hit rate
Cache interconnection

CPU

Read/Write

Valid

Address 32

Write Data 32

Read Data 32

Ready

Cache

Read/Write

Valid

Address 32

Write Data 128

Read Data 128

Ready

Memory

Several cycles per access
### Example: Haswell Core i7

<table>
<thead>
<tr>
<th></th>
<th>Registers</th>
<th>L1I</th>
<th>L1D</th>
<th>L2</th>
<th>L3</th>
<th>DRAM</th>
<th>Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Size</strong></td>
<td>16</td>
<td>32KB</td>
<td>32KB</td>
<td>4x256KB</td>
<td>8MB</td>
<td>32GB</td>
<td>1.8TB</td>
</tr>
<tr>
<td><strong>Latency</strong></td>
<td>1 cycle</td>
<td>4-5 cycles</td>
<td>4-5 cycles</td>
<td>12 cycles</td>
<td>36 cycles + 57 ns</td>
<td>2.8 ms</td>
<td></td>
</tr>
</tbody>
</table>

#### Goal: Give the illusion of large, fast, and cheap memory

- Allow programs to address space scalable to disk size at register file speed.

*Computer Architecture - 2014 - ARCOS@uc3m*
- Introduction.
- **Cache performance.**
- Trade-offs in cache design.
- Basic cache optimizations.
Average memory access time:
\[ t_{\text{avg}} = t_H + (1-h) \cdot t_M \]

**Miss penalty:**
- Time to replace a block and deliver to CPU.
- **Access time:**
  - Time to get from lower level.
  - Dependent on lower level latency.
- **Transfer time:**
  - Time to transfer a block.
  - Depending on bandwidth across levels.
CPU execution time
- (CPU cycles + Memory stall cycles) x Cycle time

CPU cycles
- IC x CPI

Memory stall cycles
- Misses number x Miss penalty
- IC x Misses per instruction x Miss penalty
- IC x Instruction Memory access x Miss rate x Miss penalty

Beware: May be different for reads and writes.

Where:
- IC → Instruction Count.
- CPI → Cycles per instruction.
- Introduction.
- Cache performance.
- Trade-offs in cache design.
- Basic cache optimizations.
Four questions on memory hierarchy

☐ **Q1**: Where can a block be placed in the upper level?
   - Block placement.

☐ **Q2**: How is a block found in the upper level?
   - Block identification.

☐ **Q3**: Which block should be replaced on a miss?
   - Block replacement.

☐ **Q4**: What happens on a write?
   - Write strategy.
**Q1: Block Placement**

- **Direct mapping.**
  - Placement $\rightarrow$ block MOD blocks_in_cache

- **Fully associative mapping.**
  - Placement $\rightarrow$ Anywhere.

- **Set associative mapping.**
  - Set placement $\rightarrow$ block MOD number_of_sets.
  - Placement within set $\rightarrow$ Anywhere.

![Diagram of cache and memory mapping]

**Computer Architecture - 2014 - ARCOS@uc3m**
- **Block Address**:  
  - **Tag**: Identifies the entry address.  
    - Validity bit in every entry to flag if content is valid.  
  - **Index**: Selects address.  

- **Block offset**:  
  - Select data within a block.  

- **Higher associativity** means:  
  - Less bits for Index.  
  - More bits for Tag.
Q3: Block Replacement

- Relevant for associative and set associative mappings:
  - Random.
    - Easy to implement.
  - LRU: Less Recently Used.
    - Increasing complexity as associativity increases.
  - FIFO: First In First Out.
    - Approximates LRU with lower complexity.

<table>
<thead>
<tr>
<th></th>
<th>2 ways</th>
<th>4 ways</th>
<th>8 ways</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LRU</td>
<td>Rand</td>
<td>FIFO</td>
</tr>
<tr>
<td>Size</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16 KB</td>
<td>114.1</td>
<td>117.3</td>
<td>115.5</td>
</tr>
<tr>
<td>64 KB</td>
<td>103.4</td>
<td>104.3</td>
<td>103.9</td>
</tr>
<tr>
<td>256 KB</td>
<td>92.2</td>
<td>92.1</td>
<td>92.5</td>
</tr>
</tbody>
</table>

Misses per1000 instr., SPEC 2000

From Hennessy & Patterson, 5Ed Appendix C
**Q4: Write strategy**

### Write through

- All writes sent to bus and memory.
- Easy to implement.
- Performance issues in SMPs.

### Write back

- Many writes are a hit.
- Write hits **not** sent to bus and memory.
- Propagation and serialization issues.
- More complex.

---

![Diagram of write through and write back strategies](image-url)
<table>
<thead>
<tr>
<th>Policy</th>
<th>Write-Through</th>
<th>Write-Back</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Data written to cache blocks.</td>
<td>Data only written to cache. Update next levels when block evicted from cache</td>
</tr>
<tr>
<td></td>
<td>Also written to next level in memory.</td>
<td></td>
</tr>
<tr>
<td>Debugging</td>
<td>Easy</td>
<td>Difficult</td>
</tr>
<tr>
<td>Write on miss?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Repeated writes sent to next level?</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
Write Buffers in caches with write through

- Why a buffer?
  - To avoid CPU stalls.

- Why a buffer instead of a register?
  - Write bursts are common.

- Are RAW hazards possible?
  - Yes.
  - Alternatives:
    - Flush buffer before a read.
    - Check buffer before a read.
Miss penalty and Out-of-Order execution

- **Miss penalty definition:**
  - Miss total latency.
  - Exposed latency (generating CPU stall).

- **Miss penalty:**
  - Memory stalls / Instructions
  - \((\text{Misses/Instructions}) \times (\text{Total latency} - \text{overlapped latency})\)
- Introduction.
- Cache performance.
- Trade-offs in cache design.
- Basic cache optimizations.
- **Reduce miss rate.**
  - Larger block size.
  - Larger cache size.
  - Higher associativity.

- **Reduce miss penalty.**
  - Multi-level caches.
  - Prioritize read over writes.

- **Reduce hit time.**
  - Avoid address translation in cache indexing.
☐ **Goal:** Reduce miss rate.
   - Improve spatial locality exploitation.

☐ **Increases miss penalty.**
   - Upon a miss, larger blocks need to be transferred.
   - More misses due to cache with less blocks.

☐ **Balance needed:**
   - High latency and high bandwidth memory:
     - Increase block size.
   - Low latency and low bandwidth memory:
     - Reduced block size.
Miss rate and block size

Block size

Miss rate

© 2007 Elsevier, Inc. All rights reserved.
**Goal:** Reduce miss rate.
- More data fit in cache.

**May increase hit time.**
- More time needed to find block.

**Higher cost.**

**Higher energy consumption.**

**Need to find balance:**
- Specially in on-chip caches.
Goal: Reduce miss rate.
- Less conflicts as more ways in a set can be used.

May increase hit time.
- More time needed to find a block.

Consequence:
- 8 ways ≈ Fully associative
□ **Goal:** Reduce miss penalty.

□ **Evolution:**
- Increasing performance gap between DRAM and CPU.
- Miss penalty cost increased over time.

□ **Alternatives:**
- Faster caches.
- Larger caches.

□ **Solution:**
- Both of them!
- Several cache levels.
Local and global miss rate

- **Average access time:**
  - \(\text{Hit time}_{L1} + \text{Miss rate}_{L1} \times \text{Miss penalty}_{L1}\)
  - \(\text{Hit time}_{L1} + \text{Miss rate}_{L1} \times \left(\text{Hit time}_{L2} + \text{Miss rate}_{L2} \times \text{Miss penalty}_{L2}\right)\)

- **Local miss rate:**
  - Misses at a cache level over accesses to that cache level.
  - L1: Miss rate\(_{L1}\)
  - L2: Miss rate\(_{L2}\)

- **Global miss rate:**
  - Misses at a cache level over all memory accesses.
  - L1: Miss rate\(_{L1}\)
  - L2: Miss rate\(_{L1}\) \times Miss rate\(_{L2}\)
Miss rate versus cache size for multilevel caches

- Local miss rate
- Global miss rate
- Single cache miss rate

© 2007 Elsevier, Inc. All rights reserved.
Intel Nehalem 4-core processor

Per core: 32KB L1 I-cache, 32KB L1 D-cache, 512KB L2 cache
5: Prioritize read misses over writes

- **Goal:** Reduce miss penalties.
  - Avoid that a read miss has to wait until writes are completed.

- **Write-through caches:**
  - Write buffer could contain an updated value for the read address.
    - A) Wait until write buffer is empty.
    - B) Check contents of write buffer.
      - Continue with read miss if no conflict with buffer.

- **Write-back caches:**
  - A read miss could replace a modified block.
    - Copy modified block to buffer, read, and dump block to memory.
    - Apply options A or B to buffer.
6: Avoid Address Translation during Indexing

- **Goal:** Reduce hit time.

- **Translation process:**
  - Virtual address → Physical address.
  - May require additional accesses to memory.
    - Or at least to TLB.

- **Idea:** Optimize the most common case (hits).
  - Use virtual addresses for the cache.

- **Tasks:**
  - Indexing the cache.
  - Comparing tags.
- **Protection:**
  - Page-level protection checked during virtual-to-physical translation.
  - Solution: Copy protection info from TLB on misses.

- **Process switch.**
  - Virtual addresses refer to different physical addresses.
    - Old process virtual addresses.
  - Solutions:
    - Flush the cache.
    - Add to cache address a PID tag.
- **Aliasing:**
  - Two different virtual addresses for the same physical address.
  - Anti-aliasing hardware: to guarantee that every cache block corresponds to a unique physical address.
    - Check multiple addresses and invalidate.
  - Page coloring: Force all aliases to have identical their n last bits.
    - Makes impossible two alias to be at the same time in cache.

- **I/O addresses:**
  - I/O typically uses physical addresses.
  - Mapping to virtual addresses to interact with virtual caches.
Solution:
- Virtual indexing and physically tagging.

Tasks:
- Indexing the cache \(ightarrow\) Use page offset.
  - This part is identical in physical and virtual addresses
- Comparing tags \(\rightarrow\) Use translated physical address.
  - Tag matching uses physical address.
Caching as a **solution** to mitigate the memory wall.

Cache performance due to **locality principle** (spatial and temporal).

**Miss penalty** dependent on **access time** and **transfer time**.

Four key dimensions in **cache design**:
- Block placement, block identification, block replacement and write strategy.

Six **basic cache optimizations**:
- **Reduce miss rate**: larger block size, larger cache size, higher associativity.
- **Reduce miss penalty**: multilevel caches, prioritize reads over writes.
- **Reduce hit time**: Avoid address translation when indexing.
  Hennessy y Patterson.
  **Sections:** B.1, B.2, B.3

- **Exercises:**
  B.1, B.2, B.3, B.4, B.5, B.6, B.7, B.8, B.9, B.10, B.11