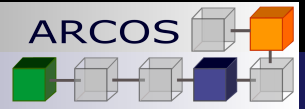


Basic cache memory

Computer Architecture

J. Daniel García Sánchez (coordinator)
David Expósito Singh
Francisco Javier García Blas

ARCOS Group
Computer Science and Engineering Department
University Carlos III of Madrid

Latency evolution

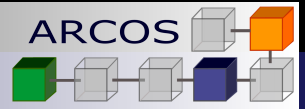
- Multiple views of performance
  - $performance = \frac{1}{latency}$
  - Useful for comparing processor and memory evolution.

- **Processors**
  - Yearly performance increase from 25% to 52%.
  - Combined effect from 1980 to 2010 $\rightarrow$ above $3,000$ times.

- **Memory**
  - Yearly performance increase around 7%
  - Combined effect from 1980 to 2010 $\rightarrow$ around 7.5 times.

Multi-core effect

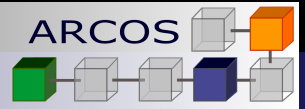
- **Intel Core i7**.
  - Two 64 bits data accesses per cycle.
  - 4 cores, 3.2 GHz $\rightarrow$ 25.6×10^9 accesses/sec
  - Instructions demand: 12.8×10^9 of 128 bits.
  - Peak bandwidth: 409.6 GB/sec
- **SDRAM memory**.
  - DDR2 (2003): 3.20 GB/sec – 8.50 GB/sec
  - DDR3 (2007): 8.53 GB/sec – 18.00 GB/sec
  - DDR4 (2014): 17.06 GB/sec – 25.60 GB/sec
- **Solutions**:
  - Multi-port memories, pipelined caches, multi-level caches, per-core caches, instruction/data caches separation.



Principle of locality

- **Principle of locality**.
  - It is **property of programs** exploited in the hardware design.
  - Programs are accessed in a relatively small portion of address space.

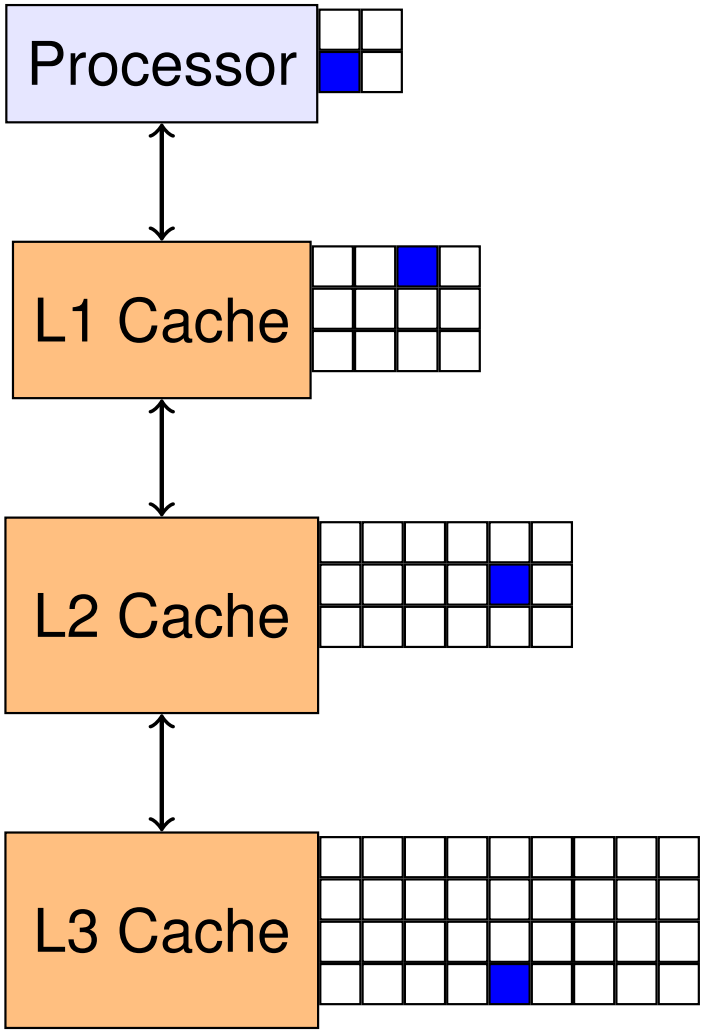
- **Types of locality**:
  - **Temporal locality**: Elements recently accessed tend to be accessed again.
    - Examples: Loops, variable reuse, ...
  - **Spatial locality**: Elements next to a recently accessed one tend to be accessed in the future.
    - Examples: Sequential execution of instructions, arrays, ...



Situation (2008)

- **SRAM** $\rightarrow$ *Static RAM*.
  - **Access time**: 0.5 ns – 2.5 ns.
  - **Cost per GB**: 2,000$ - 5,000$
- **DRAM** – *Dynamic RAM*.
  - **Access time**: 50 ns – 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$.

Memory hierarchy



Memory hierarchy

- **Block or line**: Unit of copy operations.
  - Usually composed of multiple words.
- If accessed data is **present** in upper level:
  - **Hit**: Delivered by higher level.

$$h = \frac{hits}{acceses}$$

- If accessed data is **missing**.
  - **Miss**: Block copied from lower level.
  - Data access in upper level.
  - Needed time $\rightarrow$ **Miss penalty**.

$$m = \frac{misses}{acceses} = 1 - h$$

Metrics

- Average memory access time:

$$t_M = t_H + (1 - h)t_M$$

- **Miss penalty**:
  - Time to replace a block and deliver to CPU.
  - **Access time**.
    - Time to get from lower level.
    - Depends on lower level latency.
  - **Transfer time**.
    - Time to transfer a block.
    - Depends on the bandwidth across levels.

Metrics

- **CPU execution time**:

$$t_{CPU} = \left(cycles_{CPU} + cycles_{\text{memory stall}}\right) \times t_{cycle}$$

- **CPU clock cycles**:

$$cycles_{CPU} = IC \times CPI$$

- **Memory stall cycles**:

$$cycles_{\text{memory stall}} = n_{misses} \times \text{penalty}_{miss} =$$

$$IC \times miss_{instr} \times \text{penalty}_{miss} =$$

$$IC \times memory\_accesses_{instr} \times (1 - h) \times \text{penalty}_{miss}$$

Four questions about memory hierarchy

1. Where is a block placed in the upper level?
   - **Block placement**.
2. How is a block found in the upper level?
   - **Block identification**.
3. Which block must be replaced on a miss?
   - **Block replacement**.
4. What happens on a write?
   - **Write strategy**.

Q1: Block placement

- **Direct mapping**.
  - Placement $\rightarrow$ *block* mod n_{blocks}
- **Fully associative mapping**.
  - Placement $\rightarrow$ Anywhere.
- **Set associative mapping**.
  - Set placement $\rightarrow$ *block* mod n_{sets}
  - Block placement within set $\rightarrow$ Anywhere in set.

Q2: Block identification

- **Block address**:
  - **Tag**: Identifies entry address.
    - Validity bit in every entry to signal whether content is valid.
  - **Index**: Selects the set.
- **Block offset**:
  - Selects data within block.
- **Higher associativity means**:
  - Less **index** bits.
  - More **tag** bits.

Q3: Block replacement

- **Relevant** for **associative mapping** and **set associative mapping**:
  - **Random**.
    - Easy to implement.
  - **LRU**: Least Recently Used.
    - Increasing complexity as associative increases.
  - **FIFO**.
    - Approximates LRU with a lower complexity.

	2 ways			4 ways			8 ways		
Tam.	LRU	Rand	FIFO	LRU	Rand	FIFO	LRU	Rand	FIFO
16 KB	114.1	117.3	115.5	111.7	115.1	113.3	109.0	111.8	110.4
64 KB	103.4	104.3	103.9	102.4	102.3	103.1	99.7	100.5	100.3
256 KB	92.2	92.1	92.5	92.1	92.1	92.5	92.1	92.1	92.5

Misses per 1000 instr., SPEC 2000.

Source: Computer Architecture: A Quantitative Approach. 5 Ed
Hennessy and Patterson. Morgan Kaufmann. 2012.

Q4: Write strategy

Write-through

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

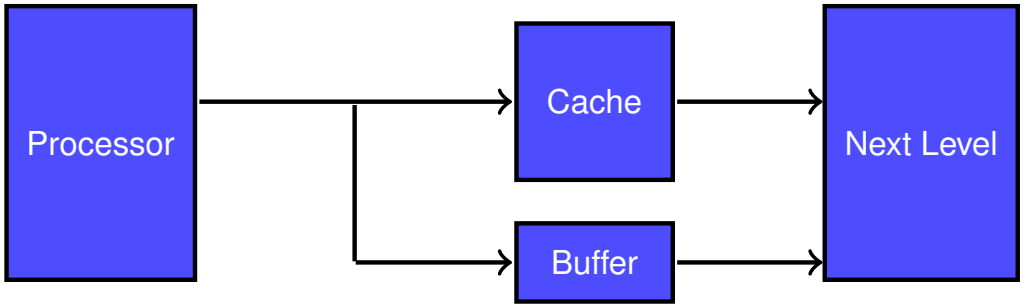
Write-back

- Many writes are a hit.
- Write hits do not go to bus and memory.
- Propagation and serialization problems.
- More complex.

Q4: Write strategy

- **Where is write done?**:
  - **write-through**: In cache block and next level in memory.
  - **write-back**: Only in cache block.
- **What happens when a block is evicted from cache?**:
  - **write-through**: Nothing else.
  - **write-back**: Next level in memory is updated.
- **Debugging**:
  - **write-through**: Easy.
  - **write-back**: Difficult.
- **Miss causes write?**:
  - **write-through**: No.
  - **write-back**: Yes.
- **Repeated write goes to next level?**:
  - **write-through**: Yes.
  - **write-back**: No.

Write buffer



- **Why a buffer?**
  - To avoid stalls in CPU.
- **Why a buffer instead of a register?**
  - Write bursts are frequent.
- **Are RAW hazards possible?**
- Yes.
- **Alternatives**:
  - Flush buffer before a read.
  - Check buffer before a read.

Miss penalty

- **Miss penalty**:
  - Total latency miss.
  - Exposed latency (generating CPU stalls).

Miss penalty

$$\frac{\text{stall\_cycles}_{memory}}{IC} =$$

$$\frac{misses}{IC} \times \left(latency_{total} - latency_{overlapped}\right)$$

Cache basic optimizations

- **Decrease** the **miss rate**.
  - Increase block size.
  - Increase cache size.
  - Increase associativity.
- **Decrease miss penalty**.
  - Multi-level caches.
  - Prioritize reads over writes.
- **Decrease** the **hit time**.
  - Avoid address translation in cache indexing.

3 Basic optimizations

- Decrease miss rate
- Decrease miss penalty
- Decrease hit time

1: Increase block size

- Larger block size $\rightarrow$ **Lower miss rate**.
  - Better exploitation of spatial locality.

- Larger block size $\rightarrow$ **Higher miss penalty**.
  - Upon miss, larger blocks need to be transferred.
  - More misses as cache has less blocks.

Need to balance:

- Memory with high latency and high bandwidth:
  - Increase block size.
- Memory with low latency and low bandwidth:
  - Decrease block size.

2: Increase cache size

- Larger cache size $\rightarrow$ **lower miss rate**.
  - More data fit in cache.
- It may **increase hit time**.
  - More time needed to find a block.
- **Higher cost**.
- **Higher energy consumption**
- **Need to find a balance**.
  - Specially in on-chip caches.

3: Increase associativity

- Higher associativity $\rightarrow$ **Lower miss rate**.
  - Less conflicts as more ways in the same set may be used.
- It may **increase hit time**.
  - More time needed to find a block.
- **Consequence**:
  - 8 ways $\approx$ Fully associative.

3 Basic optimizations

- Decrease miss rate
- Decrease miss penalty
- Decrease hit time

4: Multi-level caches

- **Goal**: **Decrease miss penalty**.
- **Evolution**:
  - **Higher distance** from DRAM and CPU performance over time.
  - **Increasing** miss penalty **cost**.
- **Alternatives**:
  - Make cache **faster**.
  - Make cache **larger**.
- **Solution**:
  - **Both of them!**
  - **Several cache levels**.

Global and local miss rates

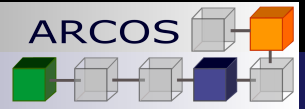
- **Local miss rate**:
  - Misses at a cache level over accesses to that cache level.
  - L1 miss rate $\rightarrow m(L1)$
  - L2 miss rate $\rightarrow m(L2)$
- **Global miss rate**:
  - Misses at a cache level over to all memory accesses.
  - L1 miss rate $\rightarrow m(L1)$
  - L2 miss rate $\rightarrow m(L1) \times m(L2)$
- **Average access time**:

$$t_h(L1) + m(L1) \times t_m(L1) =$$

$$t_h(L1) + m(L1) \times (t_h(L2) + m(L2) \times t_m(L2))$$

5: Prioritize read misses over writes

- **Goal**: **Decrease miss penalty**.
  - Avoid that a read miss has to wait until writes are completed.
- **Write-through caches**.
  - Write buffer **might contain** the data being read.
    a) Wait for write buffer to be empty.
    b) Check values in write buffer.
- **Write-back caches**.
  - A read miss **might replace a modified block**.
    - Copy modified block to buffer, read, and dump block to memory.
    - Apply options **a** or **b** to buffer.



3 Basic optimizations

- Decrease miss rate
- Decrease miss penalty
- Decrease hit time

6: Avoid address translation during indexing

- **Goal**: **Decrease hit time**.
- **Translation procedure**:
  - Virtual address → Physical address.
  - May require additional memory accesses.
    - Or at least to TLB.
- **Idea**: **Optimize the most common case** (hits).
  - Use virtual addresses for cache.
- **Tasks**:
  - **Indexing the cache** → Use offset.
  - **Comparing tags** → Use translated physical addresses.

Problems with virtual caches (I)

- **Protection**.
  - Page-level protection checked during **virtual-to-physical translation**.
  - **Solution**: Copy protection information from TLB on misses.

- **Process switch**.
  - Virtual addresses refer to **other physical addresses**.
  - **Solutions**:
    - Flush the cache.
    - Add PID to tag.

Problems with virtual caches (II)

- **Aliasing**:
  - Two different virtual addresses for the same physical address.
  - **Anti-aliasing hardware**: 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 last **n** bits.
    - Makes impossible that two alias to be at the same time in cache.
- **Input/output addresses**:
  - I/O typically uses physical addresses.
  - Mapping to virtual addresses to interact with virtual caches.

Address translation

- **Solution**:
  - Virtual indexing and physically tagging.

- **Tasks**:
  - **Indexing the cache** → Use page offset.
    - This part is identical in physical and virtual addresses.
  - **Comparing tags** → Use translated physical address.
    - Tag matching uses physical address.

Summary

- Caching as a solution to mitigate the **memory wall**.
- Technology **evolution** and principle of **locality** place more **pressure** over caches.
- **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**:
  - **Decrease miss rate**: Increase block size, increase cache size, increase associativity.
  - **Decrease miss penalty**: Multi-level caches, prioritize reads over writes.
  - **Decrease hit time**: Avoid address translation.

References

- **Computer Architecture. A Quantitative Approach**
  5th Ed.
  Hennessy and Patterson.
  Sections: B.1, B.2, B.3.

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

Basic cache memory

Computer Architecture

J. Daniel García Sánchez (coordinator)
David Expósito Singh
Francisco Javier García Blas

ARCOS Group
Computer Science and Engineering Department
University Carlos III of Madrid