Chapter 2
(Starting with Appendix B)

Instructor: Josep Torrellas
CS433
Memory Hierarchy Design

1980 microprocessor $\rightarrow$ no cache
1995 $\rightarrow$ 2 level of caches

Why?
- Programmers want more memory
- Principle of locality
- Processor - memory performance gap
- Smaller hardware is faster
- faster $\Rightarrow$ more expensive
Figure 2.1: Memory Hierarchy

(a) Memory hierarchy for a server

- CPU
  - Registers
- L1 Cache
  - Register reference
  - Size: 1000 bytes
  - Speed: 300 ps
- L2 Cache
  - Level 1 Cache reference
  - Size: 64 KB
  - Speed: 1 ns
- L3 Cache
  - Level 2 Cache reference
  - Size: 256 KB
  - Speed: 3–10 ns
  - Memory reference
  - Size: 2–4 MB
  - Speed: 10–20 ns
- Memory
  - Size: 4–16 GB
  - Speed: 50–100 ns
- I/O bus
- Disk storage
  - Disk memory reference
  - Size: 4–16 TB
  - Speed: 5–10 ms

(b) Memory hierarchy for a personal mobile device

- CPU
  - Registers
- L1 Cache
  - Register reference
  - Size: 500 bytes
  - Speed: 500 ps
- L2 Cache
  - Level 1 Cache reference
  - Size: 64 KB
  - Speed: 2 ns
- L3 Cache
  - Level 2 Cache reference
  - Size: 256 KB
  - Speed: 10–20 ns
- Memory
  - Memory reference
  - Size: 256–512 MB
  - Speed: 50–100 ns
- I/O bus
- Storage
  - Flash memory reference
  - Size: 4–8 GB
  - Speed: 25–50 us
Cache Blocks (Lines)

Block $\rightarrow$ minimum unit of information that can be present in the cache

Hit / Miss
Block placement
Block identification
Block replacement
Write strategy
Block Placement

Block address in memory: 10
Direct mapped: (Block addr) Mod (# blocks in cache)
Block #: 10 MOD 8 = 2
Set Associative: (Block Addr) Mod (# sets in cache)
Set #: 10 MOD 4 = 2
Fully Associative: Anywhere
Block Placement

- Fully associative: block 12 can go anywhere
- Direct mapped: block 12 can go only into block 4 (12 MOD 8)
- Set associative: block 12 can go anywhere in set 0 (12 MOD 4)

Figure B.2
# Block Identification

<table>
<thead>
<tr>
<th>Block Address</th>
<th>Block Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td>Index</td>
</tr>
</tbody>
</table>

- Index and block offset not stored
- Tag
- Valid bit
- Associativity $\uparrow \Rightarrow$ Size of tag $\uparrow$
Block Replacement

- Random
- Least recently used (LRU)
Write Policy

Writes form 25% of data cache traffic
   7% of total memory traffic
• Make common case fast ⇒ optimize for reads
• Also processor waits for reads to complete; need not wait for writes
• Easy to make reads fast:
   read data in parallel with reading and comparing tag
• Writes : Block cannot be modified until tag matches
• Size of write
Two Basic Write Policies

- Write through
- Write back
- Dirty bit saves writes
- Write back $\Rightarrow$ less memory bandwidth
- Write through $\Rightarrow$ read miss never results in write to lower level. Easier to implement
- Write stall during write through
  - use a write buffer
Write Miss Policies

On a write miss:

• Write allocate (fetch on write)
• No-write allocate (write around)

• Write miss policy and write policy are independent
• Write back caches generally use write allocate
• Write through caches often use no write allocate
Cache Performance

Avg Memory Access Time = Hit time + Miss rate * Miss Penalty

Example: Compare a 32-KB unified cache with a 16KB I-cache and 16 KB D-cache.

Given: Miss rates are 1.99%, 0.64% and 6.47% respectively.

75% of memory accesses are instruction references.

Hit 1 clock cycle
Miss 50 clock cycles

Load/Store access in unified cache: adds 1 extra clock cycle due to structural hazard
For the split cache, overall miss rate is
\[(75\% \times 0.64\%) + (25\% \times 6.47\%) = 2.10\%\]

Average memory access time split
\[= 75\% \times (1 + 0.64\% \times 50) + 25\% \times (1 + 6.47\% \times 50)\]
\[= 2.05\]

Avg memory access time unified
\[= 75\% \times (1 + 1.99\% \times 50) + 25\% \times (1 + 1.99\% \times 50)\]
\[= 2.24\]
CPU time = \left[ \text{CPU Execution Clock cycles} + \text{Memory stall Clock cycles} \right] \times \text{clock cycle time}

Assumption: All memory stalls are due to cache misses.

\left[ \text{Memory stall clock cycles} \right] = \text{Reads} \times \text{Read miss rate} \times \text{Read miss Penalty} \\
+ \text{Writes} + \text{Write miss Rate} \times \text{Write miss Penalty}
\quad = \text{Memory accesses} \times \text{Miss rate} \times \text{Miss Penalty}
Consider a machine with:

- Cache miss penalty = 50 cycles
- CPI_{execution} = 2.0
- Miss Rate = 2%
- Memory References per inst = 1.33

CPU time = IC \times \left( CPI_{execution} + \frac{\text{Memory stall}}{\text{Instruction}} \right) \times \text{Clock cycle time}

= IC \times (2.0 + (1.33 \times 2\% \times 50)) \times \text{Clock cycle time}

= IC \times 3.33 \times \text{Clock Cycle time}

Impact of cache: 1.67x
If no cache: 68.5 30x
If perfect CPI_{execution}: 2.33x
Cache misses have double-barreled impact on a CPU with low CPI and fast clock

1. Lower CPI \textit{execution} $\rightarrow$ relative impact of FIXED number of cache miss cycles increases

2. Identical memory hierarchy $\rightarrow$ CPU with faster clock sees more stall cycles

Note: in ooo execution processors: part of the memory access latency is overlapped with computation!