Memory-Hierarchy Design

Why is memory important?
- Memory wall due to emphasis on speed for processor density for DRAM

The Goal
Provide memory system with cost of cheapest level of memory and with speed as fast as fastest level of memory

Memory Hierarchy

- The correctness criteria
  - The execution results of a program should be as if it were executed on a system without any cache memory
General Caching Principles

- **Locality**
  - Temporal Locality: referenced again soon
  - Spatial Locality: nearby items referenced soon

- **Locality + smaller HW is faster = memory hierarchy**
  - Levels: each smaller, faster, more expensive/byte than level below
  - inclusive: data found in top also found in the bottom

Some Definitions

- **Upper** is closer to processor
- **Block**: minimum unit of information that can be present in cache
- **Hit time**: time to access cache, including hit determination
- **Hit rate**: fraction found in the cache
  - Miss rate = 1 - Hit Rate
- **Miss penalty**: time to fetch a block from lower level, including time to replace in CPU
  - access time: time to access lower level
  - transfer time: time to transfer block

Differences in Memory Levels

<table>
<thead>
<tr>
<th></th>
<th>First level cache</th>
<th>Second level cache</th>
<th>Virtual memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block size</td>
<td>4-128 bytes</td>
<td>32-256 bytes</td>
<td>4-64K, 16-1MB</td>
</tr>
<tr>
<td>Hit time</td>
<td>1-2 clock cycles</td>
<td>6-57 clock cycles</td>
<td>30-900 clock cycles</td>
</tr>
<tr>
<td>Miss penalty</td>
<td>4-40 clock cycles</td>
<td>30-200 clock cycles</td>
<td>400-6,000,000 clock cycles</td>
</tr>
<tr>
<td>Miss rate (local)</td>
<td>~2-200</td>
<td>~25-300</td>
<td>0.0001-0.0005</td>
</tr>
<tr>
<td>Size</td>
<td>1-128 KB</td>
<td>256 KB-16 MB</td>
<td>16-643 MB</td>
</tr>
<tr>
<td>Backing store</td>
<td>Second level cache Page-mode DRAM Disk</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q1: block placement</td>
<td>Direct mapped</td>
<td>Direct mapped or link associations</td>
<td>Fully associative</td>
</tr>
<tr>
<td>Q2: block identification</td>
<td>TagBlock</td>
<td>TagBlock</td>
<td>Table</td>
</tr>
<tr>
<td>Q3: block replacement</td>
<td>N.A. (always mapped) Random</td>
<td>LRU</td>
<td></td>
</tr>
<tr>
<td>Q4: write strategy</td>
<td>Write through or write back</td>
<td>Write back</td>
<td>Write back</td>
</tr>
</tbody>
</table>

Four Questions for Memory Hierarchy Designers

- **Q1**: Where can a block be placed in the upper level? (Block placement)
- **Q2**: How is a block found if it is 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: Where can a block be placed in the upper level?

- **Direct Mapped:** Each block has only one place that it can appear in the cache.
- **Fully associative:** Each block can be placed anywhere in the cache.
- **Set associative:** Each block can be placed in a restricted set of places in the cache.
  - If there are n blocks in a set, the cache placement is called n-way set associative

### Associativity Examples

- **Fully associative:**
  - Block 12 can go anywhere
- **Direct mapped:**
  - Block no. = (Block address) mod (# of blocks in cache)
  - Block 12 can go only into block 4 (12 mod 8)
- **Set associative:**
  - Set no. = (Block address) mod (# of sets in cache)
  - Block 12 can go only into set 0 (12 mod 4) but any block in that set

Q2: How is a block found if it is in the upper level?

- The address can be divided into two main parts
  - Block offset: selects the data from the block
    - offset size = log2 (block size)
  - Block address: tag + index
    - index: selects set in cache
      - index size = log2 (# of sets)
    - tag: compared to tag in cache to determine hit
      - tag size = address size - index size - offset size
Mapping Function

Consider
- 64KB cache
- 4 Byte data transfer (block) size between main memory and cache
- Cache organized in 16K block frames (slots) of 4 Bytes each

- Main memory has 16MBytes
  (Or we can treat the memory as having 4M blocks of 4 Bytes each)
  → 24 bit memory address

Mapping Structure

<table>
<thead>
<tr>
<th>Mapping</th>
<th>Rows</th>
<th>Columns</th>
<th>Rows</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main Memory</td>
<td>256</td>
<td>16K</td>
<td>1</td>
<td>16K</td>
</tr>
<tr>
<td>Direct</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Associative</td>
<td>4M</td>
<td>1</td>
<td>16K</td>
<td>1</td>
</tr>
<tr>
<td>Set Associative</td>
<td>512</td>
<td>8K</td>
<td>2</td>
<td>8K</td>
</tr>
</tbody>
</table>

Direct Mapping

8 Bits Tag | 14 Bits Index No | 2
Associative Mapping

Set Associative Mapping

Address Match to Cache

Q3: Which Block Should be Replaced on a Miss?

- Direct Mapped:
  - No Choice (only one candidate)
- Set Associative or Fully Associative:
  - Random - easier to implement but less efficient
  - Least Recently used - harder to implement but more efficient
- Miss rates for caches with different size, associativity and replacement algorithm.

<table>
<thead>
<tr>
<th>Size</th>
<th>Two-way LRU</th>
<th>Two-way Random</th>
<th>Four-way LRU</th>
<th>Four-way Random</th>
<th>Eight-way LRU</th>
<th>Eight-way Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 KB</td>
<td>5.18%</td>
<td>5.89%</td>
<td>4.67%</td>
<td>5.29%</td>
<td>4.39%</td>
<td>4.96%</td>
</tr>
<tr>
<td>64 KB</td>
<td>1.08%</td>
<td>3.01%</td>
<td>1.34%</td>
<td>1.66%</td>
<td>1.39%</td>
<td>1.53%</td>
</tr>
<tr>
<td>256 KB</td>
<td>1.15%</td>
<td>1.17%</td>
<td>1.13%</td>
<td>1.12%</td>
<td>1.12%</td>
<td>1.12%</td>
</tr>
</tbody>
</table>
Q4: What Happens on a Write?

• Write through: The information is written to both the block in the cache and to the block in the lower-level memory → combined with write buffers so it won’t wait for lower level memory access
• Write back: The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.
  - Is block clean or dirty? (add a dirty bit to each block)
• Pros and Cons of each (obvious ones):
  - Write through
    - Read misses cannot result in writes to memory,
    - Easier to implement
    - Main memory is “always” up-to-date (less coherence problem)
  - Write back
    - Less memory traffic
    - Perform writes at the speed of the cache

Q4: What Happens on a Write?

• Since data does not have to be brought into the cache on a write miss, there are two options:
  - Write allocate
    - The block is brought into the cache on a write miss
    - Used mostly with write-back caches
    - Hope subsequent accesses to the block hit in cache
  - No-write allocate
    - The block is modified in memory, but not brought into the cache
    - Used with write-through caches
    - Writes have to go to memory anyway, so why bring the block into the cache

Cache Measures

• Hit rate: fraction found in the cache
  - So high that we usually talk about Miss rate = 1 - Hit Rate
• Hit time: time to access the cache
• Miss penalty: time to fetch a block from lower level, including time to replace in CPU
  - Access time: time to access lower level
  - Transfer time: time to transfer block
• Average memory access time
  = Hit time + Miss rate x Miss penalty (ns or clocks)

Miss Rate Pitfall

• Miss Rate to Memory Hierarchy Performance is like MIPS to CPU Performance
• Increasing block size generally decreases miss rate but the accompanying increase in miss penalty may outweigh the decrease in miss rate
• Average memory access time is a more reliable measure of cache performance
Implications For CPU

- Fast hit check for every memory access
  - Hit is the common case
- Unpredictable memory access time
  - 10s of clock cycles: wait
  - 1000s of clock cycles:
    - Interrupt & switch & do something else
    - New style: multithreaded execution
- How to handle miss (10s \(\rightarrow\) HW, 1000s \(\rightarrow\) SW)?

Example: Alpha 21064 Data Cache

- The data cache of the Alpha 21064 has the following features
  - 8 KB of data
  - 32 byte blocks
  - Direct mapped placement
  - Write through (no-write allocate, 4-block write buffer)
  - 34 bit physical address composed of
    - 5 bit block offset
    - 9 bit index : 256 blocks = 8192/(32x1)
    - 21 bit tag

Writes in Alpha 21064

- No write merging vs. write merging in write buffer

Example: Alpha 21064 Data Cache

A cache read has 4 steps
1. The address from the cache is divided into the tag, index, and block offset
2. The index selects block
3. The address tag is compared with the tag in the cache, the valid bit is checked, and data to be loaded is selected
4. If the valid bit is set, the data is loaded into the processor

If there is a write, the data is also sent to the write buffer
2-Way Set Associative Cache

Features of an 8 KB 2-way set associative cache
5 bit block offset
5 bit index
22 bit tag

The set associative cache has extra hardware for 2 tag comparisons mux to select data

Compared to the direct mapped cache, the set associative cache will tend to have a smaller miss rate but a larger hit time.

Split vs. Unified Cache

- Unified cache (mixed cache): Data and instructions are stored together
- Split cache: Data and instructions are stored separately
- Why do instruction caches have a lower miss ratio?

<table>
<thead>
<tr>
<th>Size</th>
<th>Instruction cache</th>
<th>Data cache</th>
<th>Unified cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 KB</td>
<td>3.06%</td>
<td>24.61%</td>
<td>13.34%</td>
</tr>
<tr>
<td>2 KB</td>
<td>2.26%</td>
<td>20.57%</td>
<td>9.78%</td>
</tr>
<tr>
<td>4 KB</td>
<td>1.78%</td>
<td>15.98%</td>
<td>7.24%</td>
</tr>
<tr>
<td>8 KB</td>
<td>1.10%</td>
<td>10.19%</td>
<td>4.57%</td>
</tr>
<tr>
<td>16 KB</td>
<td>0.64%</td>
<td>6.47%</td>
<td>2.87%</td>
</tr>
<tr>
<td>32 KB</td>
<td>0.39%</td>
<td>4.82%</td>
<td>1.99%</td>
</tr>
<tr>
<td>64 KB</td>
<td>0.15%</td>
<td>3.77%</td>
<td>1.35%</td>
</tr>
<tr>
<td>128 KB</td>
<td>0.02%</td>
<td>2.88%</td>
<td>0.95%</td>
</tr>
</tbody>
</table>

Cache Performance

CPU time = IC x (CPI_{instruction} + Memory accesses per instruction x Miss rate x Miss penalty) x Clock cycle time

Misses per instruction = Memory accesses per instruction x Miss rate

CPU time = IC x (CPI_{instruction} + Misses per instruction x Miss penalty) x Clock cycle time

Improving Cache Performance

- Average memory-access time
  \[
  \text{hit time} + \text{Miss rate} \times \text{Miss penalty}
  \]

- Improve performance by:
  1. Reduce the miss rate,
  2. Reduce the miss penalty, or
  3. Reduce the time to hit in the cache.
Summary

• CPU-Memory gap is major performance obstacle for achieving high performance
• Memory hierarchies
  – Take advantage of program locality
  – Closer to processor → smaller, faster, more expensive
  – Further from processor → bigger, slower, less expensive
• 4 questions for memory hierarchy
  – Block placement, block identification, block replacement, and write strategy
• Cache parameters
  – Cache size, block size, associativity