Chapter Seven

Large and Fast: Exploiting Memory Hierarchy
Memories: Review

- **SRAM:**
  - value is stored on a pair of inverting gates
  - very fast but takes up more space than DRAM (4 to 6 transistors)

- **DRAM:**
  - value is stored as a charge on capacitor (must be refreshed)
  - very small but slower than SRAM (factor of 5 to 10)
Users want large and fast memories!

SRAM access times are 2 - 25ns at cost of $100 to $250 per Mbyte.
DRAM access times are 60-120ns at cost of $5 to $10 per Mbyte.
Disk access times are 10 to 20 million ns at cost of $.10 to $.20 per Mbyte.

Try and give it to them anyway
  - build a memory hierarchy
Locality

- A principle that makes having a memory hierarchy a good idea

- If an item is referenced,

  temporal locality: it will tend to be referenced again soon
  spatial locality: nearby items will tend to be referenced soon.

*Why does code have locality?*

- Our initial focus: two levels (upper, lower)
  - block: minimum unit of data
  - hit: data requested is in the upper level
  - miss: data requested is not in the upper level
Cache

- Two issues:
  - How do we know if a data item is in the cache?
  - If it is, how do we find it?
- Our first example:
  - block size is one word of data
  - "direct mapped"

  For each item of data at the lower level, there is exactly one location in the cache where it might be.

  e.g., lots of items at the lower level share locations in the upper level
Direct Mapped Cache

- Mapping: address is modulo the number of blocks in the cache
Example

- Memory: 32 words ==> 5 bits
- Cache: 8 words ==> 3 bits
- 00001, 01001, 10001, 11001 in memory maps to cache 001 (use the lower log2(8) bits, equivalent to mod 8)
- Similarly, 00101, 01101, 10101, 11101 maps to cache 101
Tags

• The mapping from memory location to cache location is straightforward (many-to-one mapping), but how about the other way around (one-to-many mapping)?
• In other words, how do we know whether the data in the cache corresponds to a requested word?
• Need to add a set of tags to the cache. These tags contain the address information required to identify whether a hit occurs.
• Tags contain the information we throw away in the many-to-one mapping, i.e., the upper portion of the address.
• For example, we need 2-bit tags in Figure 7.5.
• Also need an additional bit, called the valid bit, to indicate whether a cache block has valid information. (cache is empty at start up)
• Figure 7.6 illustrates the contents of an 8-word direct-mapped cache as it responds to a series of requests from the processor.
Direct Mapped Cache

- For MIPS:

What kind of locality are we taking advantage of?
Cache Size

- Each unit: Block size + tag size + valid field size
- Assuming 32-bit byte address, a direct-mapped cache of size \(2^n\) words with one-word (4-byte) block will require:

\[
2^n \times (32 + (32 - n - 2) + 1) = 2^n \times (63-n)
\]

- Example: How many total bits are required for a direct-mapped cache with 64KB of data and one-word blocks, assuming 32-bit address?

64 KB = 16 K words = \(2^{14}\) words = \(2^{14}\) blocks
Each block has 32 bits of data plus a tag, which is 32-14-2 bits, plus a valid bit.
Total cache size = \(2^{14} \times (32 + (32 -14 -2) +1) = 2^{14} \times 49 = 784\times 2^{10} = 784\) Kbits = 96 KB.
Hits vs. Misses

- **Read hits**
  - this is what we want!

- **Read misses**
  - stall the CPU, fetch block from memory, deliver to cache, restart
  - different from pipelining since we must continue executing some instructions while stalling others there.

- **Write hits:**
  - can replace data in cache and memory (write-through)
  - write the data to the cache and the write buffer (write-buffer)
  - write the data only into the cache (write-back the cache later)

- **Write misses:**
  - read the entire block into the cache, then write the word
Dealing with instruction cache miss

- Send the original PC value (current PC - 4) to the memory
- Instruct the main memory to perform a read and wait for the memory to complete its access.
- Write the cache entry.
- Restart the instruction execution at the first step, which will refetch the instruction, this time finding it in the cache.
- What about data cache miss?
DECStation 3100 Cache

- One instruction cache, one data cache
- Each cache 64 KB, or 16K words, with a one-word block
- Use write-through scheme, writing the data into both the memory and the cache.
Direct Mapped Cache

- Taking advantage of spatial locality:
  Cache block = (Block address) mod (Number of Cache Blocks)
Mapping an Address to a Multiword Cache Block

• Consider a cache with 64 blocks and a block size of 16 bytes. What block number does byte address 1200 map to?
Miss Rate versus Block Size

![Graph showing miss rate versus block size with various block sizes and miss rates.](image-url)
Hardware Issues

- Make reading multiple words easier by using banks of memory

- It can get a lot more complicated...
Performance

- Increasing the block size tends to decrease miss rate:

![Miss rate graph]

- Use split caches because there is more spatial locality in code:

<table>
<thead>
<tr>
<th>Program</th>
<th>Block size in words</th>
<th>Instruction miss rate</th>
<th>Data miss rate</th>
<th>Effective combined miss rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td>1</td>
<td>6.1%</td>
<td>2.1%</td>
<td>5.4%</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2.0%</td>
<td>1.7%</td>
<td>1.9%</td>
</tr>
<tr>
<td>spice</td>
<td>1</td>
<td>1.2%</td>
<td>1.3%</td>
<td>1.2%</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.3%</td>
<td>0.6%</td>
<td>0.4%</td>
</tr>
</tbody>
</table>
Performance

- Simplified model:
  \[
  \text{execution time} = (\text{execution cycles} + \text{stall cycles}) \times \text{cycle time}
  \]
  \[
  \text{stall cycles} = \# \text{ of instructions} \times \text{miss ratio} \times \text{miss penalty}
  \]

- Two ways of improving performance:
  - decreasing the miss ratio
  - decreasing the miss penalty

*What happens if we increase block size?*
Calculating Cache Performance

• Assume an instruction cache miss rate for gcc of 2% and a data cache miss rate of 4%. If a machine has a CPI of 2 without any memory stalls and the miss penalty is 40 cycles for all misses, determined how much faster a machine would run with a perfect cache that never missed. (Use instruction frequencies for gcc from fig. 4.54)
Cache Performance with Increased Clock Rate

- Relative cache penalties increase as machine becomes faster!
Direct-mapped vs Fully-Associative Scheme

- In a direct-mapped scheme, there is a direct mapping from any block address in memory to a single location in the upper level of the hierarchy.
- There exist other possibilities. For example, a block can be placed in any location in the cache. This is called the fully associative scheme.
- In a fully associative scheme, we need to search all in entries in the cache to find a given block.
- Additional hardware (comparator) is required. Thus fully associative scheme is practical only for caches with small number of blocks.
- Between direct-mapped and fully-associative is the set-associative scheme.
Set-Associative Scheme

- In a set-associative cache, there are a fixed number of locations (at least 2) where each block can be placed.
- A set-associative cache with n locations for a block is called an n-way associative cache.
- An n-way associative cache consists of a number of sets, each of which consists of n blocks.
- The set containing the memory block:
  
  \[(\text{Block number}) \mod (\text{Number of sets in the cache})\]

- We then need to perform a search to find the block in the set.
Location of Memory Block -- Example

- Given a memory block whose address is 12:
Decreasing miss ratio with associativity

<table>
<thead>
<tr>
<th>Block</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(direct mapped)

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Two-way set associative

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Four-way set associative

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Eight-way set associative (fully associative)

| Tag | Data | Tag | Data | Tag | Data | Tag | Data | Tag | Data | Tag | Data | Tag | Data | Tag | Data | Tag | Data |
|-----|------|-----|------|-----|------|-----|------|-----|------|-----|------|-----|------|-----|------|
|     |      |     |      |     |      |     |      |     |      |     |      |     |      |     |      |

• Note, however, that hit time is increased with increasing degree of associativity.
Locating a Block in the Cache

- Each block in a set-associative cache includes an address tag that gives the block address.
- The index value is used to select the set containing the address of interest.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Index</th>
<th>Block Offset</th>
</tr>
</thead>
</table>
An implementation
Performance

![Graph showing performance with different associativities and cache sizes. The y-axis represents miss rate, and the x-axis shows associativities from One-way to Eight-way. The graph includes lines for different cache sizes: 1 KB, 2 KB, 4 KB, 8 KB, 16 KB, 32 KB, 64 KB, and 128 KB.]
Decreasing miss penalty with multilevel caches

- **Add a second level cache:**
  - often primary cache is on the same chip as the processor
  - use SRAMs to add another cache above primary memory (DRAM)
  - miss penalty goes down if data is in 2nd level cache

- **Example:**
  - CPI of 1.0 on a 500Mhz machine with a 5% miss rate, 200ns DRAM access
  - Adding 2nd level cache with 20ns access time decreases miss rate to 2%

- **Using multilevel caches:**
  - try and optimize the hit time on the 1st level cache
  - try and optimize the miss rate on the 2nd level cache
Example
Virtual Memory

- Main memory can act as a cache for the secondary storage (disk)

- Advantages:
  - illusion of having more physical memory
  - program relocation
  - protection
Pages: virtual memory blocks

- Page faults: the data is not in memory, retrieve it from disk
  - huge miss penalty, thus pages should be fairly large (e.g., 4KB)
  - reducing page faults is important (LRU is worth the price)
  - can handle the faults in software instead of hardware
  - using write-through is too expensive so we use writeback
Page Tables
Page Tables

Virtual address

Virtual page number

Page offset

Valid

Physical page number

Page table

If 0 then page is not present in memory

Physical page number

Page offset

Physical address
Making Address Translation Fast

- A cache for address translations: translation lookaside buffer
TLBs and caches

Virtual address

- TLB access
  - TLB hit?
    - Yes: Physical address
    - No: TLB miss exception
  - TLB miss exception

- Write?
  - Yes: Write access bit on?
    - Yes: Write data into cache, update the tag, and put the data and the address into the write buffer
    - No: Write protection exception
  - No: Try to read data from cache

- Cache hit?
  - Yes: Deliver data to the CPU
  - No: Cache miss stall
Modern Systems

- Very complicated memory systems:

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Intel Pentium Pro</th>
<th>PowerPC 604</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual address</td>
<td>32 bits</td>
<td>52 bits</td>
</tr>
<tr>
<td>Physical address</td>
<td>32 bits</td>
<td>32 bits</td>
</tr>
<tr>
<td>Page size</td>
<td>4 KB, 4 MB</td>
<td>4 KB, selectable, and 256 MB</td>
</tr>
<tr>
<td>TLB organization</td>
<td>A TLB for instructions and a TLB for data&lt;br&gt;Both four-way set associative&lt;br&gt;Pseudo-LRU replacement&lt;br&gt;Instruction TLB: 32 entries&lt;br&gt;Data TLB: 64 entries&lt;br&gt;TLB misses handled in hardware</td>
<td>A TLB for instructions and a TLB for data&lt;br&gt;Both two-way set associative&lt;br&gt;LRU replacement&lt;br&gt;Instruction TLB: 128 entries&lt;br&gt;Data TLB: 128 entries&lt;br&gt;TLB misses handled in hardware</td>
</tr>
</tbody>
</table>

Cache organization: Split instruction and data caches

Cache size: 8 KB each for instructions/data

Cache associativity: Four-way set associative

Replacement: Approximated LRU replacement

Block size: 32 bytes

Write policy: Write-back

©1998 Morgan Kaufmann Publishers
Some Issues

- Processor speeds continue to increase very fast — much faster than either DRAM or disk access times
- Design challenge: dealing with this growing disparity
- Trends:
  - synchronous SRAMs (provide a burst of data)
  - redesign DRAM chips to provide higher bandwidth or processing
  - restructure code to increase locality
  - use prefetching (make cache visible to ISA)