Introduction

Cache Overview

Cache Optimizations

Virtual Memory

Concepts Introduced in Appendix B

Memory Hierarchy

- Exploits the principal of spatial and temporal locality.
- Smaller memories are faster, require less energy to access, and are more expensive per byte.
- Larger memories are slower, require more energy to access, and are less expensive per byte.
- Typically, data found in one level is also found in the level below.
- goals:
  - Catch most of the references in the fast memory.
  - Have the cost per byte almost as low as that at the slowest memory level.
- Will see that a memory hierarchy is used to implement protection schemes as well.

Memory Hierarchy Terms

- Hit - item found in that level of the hierarchy
- Miss - item not found in that level of the hierarchy
- Hit Time - time required to access the desired item
- Miss Penalty - the additional time required to service the miss
- Miss Rate - fraction of accesses that are not in that level
- Block - the amount of information that is retrieved from the next lower level on a miss

Equations

\[
\text{miss \_rate} = \frac{\text{number \_of \_misses}}{\text{total \_references}}
\]

\[
\text{hit \_rate} = \frac{\text{number \_of \_hits}}{\text{total \_references}}
\]

\[
\text{avg \_mem \_access \_time} = \text{hit \_time} + \text{miss \_rate} \times \text{miss \_penalty}
\]

\[
\text{total \_cycles} = \text{hit \_time} \times \text{total \_references} + \text{miss \_penalty} \times \text{total \_misses}
\]
Memory Hierarchy Evaluation Methods

- software approach
  - Generate traces of memory addresses by instrumentation, simulation, or traps.
  - Use a memory hierarchy simulator offline or on-the-fly.
- hardware approach
  - Use results from hardware performance counters.
  - Often cannot exactly reproduce results.

Memory Hierarchy Questions

- Where can a block be placed in the current level? (block placement)
- How is a block found if it is in the upper level? (block identification)
- Which block should be replaced on a miss? (block replacement)
- What happens on a write? (write strategy)

Cache Topics

- organization
- replacement policy
- write policy
- improving cache performance

Cache Organizations

- direct mapped
  - A memory block can be placed in only one cache line.
  - $set = block\_address \mod number\_of\_blocks\_in\_cache$
- fully associative
  - A memory block can be placed in any cache line.
- set associative
  - A memory block can be placed in any cache line within a single set of lines.
  - $set = block\_address \mod number\_of\_sets\_in\_cache$
Block Replacement

- If more than one block in a cache set, then a block must be selected to be replaced on a cache miss.
  - random
    - easy to implement in hardware
    - may not be able to reproduce behavior
  - LRU
    - least-recently assessed block is chosen
    - reduces miss rates
    - can be expensive to implement for high levels of associativity
  - FIFO
    - block loaded first is chosen
    - approximates LRU, but is less complex to implement

Write Policy

- write through
  - The data is written to both the current cache level and to the next level of the memory hierarchy.
  - Simpler to implement.
  - Can use write buffers to reduce stalls.
  - Cache and next memory hierarchy level are consistent.
- write back
  - The data is written to only the current cache level.
  - A modified cache block (dirty bit set) is written to the next memory hierarchy level when it is replaced.
  - Reduces traffic at the next memory hierarchy level.

Write Miss Policy

- write allocate
  - Block is loaded into the cache on a write miss.
  - Typically used with write back.
- no-write allocate
  - Block is not loaded into the cache on a write miss, but is updated in the next memory hierarchy level.
  - Typically used with write through.

Direct Mapped Example

- Assume a 16 byte line size, 128 cache sets, and an associativity of one (direct mapped).
- How is the physical address partitioned?

<table>
<thead>
<tr>
<th>Address</th>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x12c</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x130</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x4c0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x1134</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x1138</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x24c8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x128</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x130</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x4c4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x8c8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Set Associative Example

- Assume a 2-way set associative cache with a 16 byte line size, 64 cache sets, and an LRU replacement policy.
- How is the physical address partitioned?

<table>
<thead>
<tr>
<th>tag</th>
<th>index</th>
<th>offset</th>
</tr>
</thead>
</table>

- Fill in the information for the following memory references.

<table>
<thead>
<tr>
<th>Address</th>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x12c</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x130</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x134</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x138</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x24c8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x128</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x130</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x4c4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x8c8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Write-Through, No-Write Allocate

- Fill in what happens for each type of access for a write-through, no-write allocate cache.

<table>
<thead>
<tr>
<th>Access Type</th>
<th>Update Cache</th>
<th>Access Next Mem Hier Level</th>
<th>Update Next Mem Hier Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>read hit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>read miss</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>write hit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>write miss</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Write-Through, No-Write Allocate Example

- Assume a 2-way set associative cache with a 16 byte line size, 64 cache sets, an LRU replacement policy, and a write-through, no-write allocate policy.
- Fill in the information for the following memory references.

<table>
<thead>
<tr>
<th>R/W</th>
<th>Address</th>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
<th>Result</th>
<th>Access Next Mem Hier Level</th>
<th>Update Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>0x12c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>0x130</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>0x1134</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>0x1138</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>0x2130</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>0x2134</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>0x130</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Write-Back, Write Allocate

- Fill in what happens for each type of access for a write-back, write allocate cache.

<table>
<thead>
<tr>
<th>Access Type</th>
<th>Update Cache</th>
<th>Access Next Mem Hier Level</th>
<th>Update Next Mem Hier Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>read hit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>read miss</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>write hit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>write miss</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Write-Back, Write Allocate Example

- Assume a 2-way set associative cache with a 16 byte line size, 64 cache sets, an LRU replacement policy, and a write-back, write allocate policy.
- Fill in the information for the following memory references.

<table>
<thead>
<tr>
<th>R/W</th>
<th>Address</th>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
<th>Result</th>
<th>Access Next</th>
<th>Memory Level</th>
<th>Cache</th>
<th>Update Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>0x12c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>0x130</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>0x1134</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>0x1138</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>0x2130</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>0x2134</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>0x130</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Improving Cache Performance

\[
\text{avg\_mem\_access\_time} = \text{hit\_time} + \text{miss\_rate} \times \text{miss\_penalty}
\]

- reducing cache miss rate
- reducing cache miss penalty
- reducing cache hit time

Reducing Cache Miss Rate

- larger block size
- larger cache size
- higher associativity

Cache Miss Categories

- compulsory - The first access to a block has to be loaded into cache.
- capacity - These blocks are replaced and later retrieved since the cache cannot contain all the blocks needed during execution.
- conflict - Occurs when too many blocks map to the same cache set.
## Larger Block Size

- **Advantages**
  - Exploits spatial locality.
  - Reduces compulsory misses.
- **Disadvantages**
  - Lengthens miss penalty.
  - May waste bandwidth bringing in bytes that are unused.

## Larger Cache Size

- **Advantages**
  - Reduces capacity and conflict misses.
- **Disadvantages**
  - Requires more space.
  - May increase hit time.
  - Requires more power for each access.

## Higher Associativity

- **Advantages**
  - Reduces miss rate.
- **Disadvantages**
  - May increase hit time.
  - Requires more space and power.

## Reducing Cache Miss Penalty

- **Multi-level caches**
  - Give priority to read misses before write misses.
Multi-Level Caches

- Became popular as the miss penalty for primary caches continued to increase.
- Most general purpose machines have 3 or more levels of cache and now all on the same chip.
- L2 (L3) caches are typically much bigger than L1 (L2) caches with larger block sizes and higher associativity levels.
- L2 and L3 caches tend to be unified.

\[
\text{avg \_ access \_ time} = \text{hit \_ time \_ L1} + \text{miss \_ rate \_ L1} \times \text{miss \_ penalty \_ L1} \\
\text{miss \_ penalty \_ L1} = \text{hit \_ time \_ L2} + \text{miss \_ rate \_ L2} \times \text{miss \_ penalty \_ L2} \\
\text{local \_ miss \_ rate} = \frac{\text{misses \_ in \_ cache}}{\text{accesses \_ to \_ cache}} \\
\text{global \_ miss \_ rate} = \frac{\text{misses \_ in \_ cache}}{\text{accesses \_ to \_ L1 \_ cache}}
\]

Giving Priority to Read Misses over Write Misses

- The CPU can check the addresses in the write buffer on a read miss in case the desired item is in the buffer.
- **Advantages**
  - The CPU is much more likely to use the value read from memory sooner than a value written to memory.
  - The CPU can initiate the read miss to the next memory hierarchy level without waiting for the write buffer to drain.
  - The read access to the next memory hierarchy level can be avoided if it can be obtained from the write buffer.
  - Write buffers can be used to make write-back caches more effective.
    - First, copy the dirty block to a write buffer.
    - Second, load the block from the next memory hierarchy level to cache.
    - Third, write the dirty block to the next memory hierarchy level.
- **Disadvantage** is the additional complexity.

Reducing Cache Hit Time

- Access the cache with a virtual instead of a physical address.
- **Problems**
  - Have to purge the cache on a context switch unless include the PID as part of the tag.
  - Can have synonyms (aliases), which are multiple copies of the same physical data with different virtual addresses.
  - Page protection information from the DTLB would have to be added to each virtual cache line.
  - I/O and L2/L3 accesses are typically performed using physical addresses and would require mapping to virtual addresses to deal with the virtually addressed cache for I/O and L2/L3 block replacements.

Virtual Caches

- **virtual caches**
- **virtually indexed, physically tagged caches**
Virtually Indexed, Physically Tagged Caches

- Index into the cache with the page offset.
- Do the tag comparison after the virtual to physical address translation.
- Advantage is that the access to the data in the cache can start sooner.
- Limitation is that one way of a VIPT cache can be no larger than the page size.

<table>
<thead>
<tr>
<th>Address</th>
<th>Virtual Page Number</th>
<th>Page Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Tag</td>
<td>Index</td>
</tr>
</tbody>
</table>

Overlays

- A programmer divides their programs into pieces.
- The programmer determines which pieces are never needed to be used at the same time.
- A portion of the program loads from disk or stores to disk these pieces during execution.
- The programmer ensures that the maximum number of program pieces used at the same time fits into physical memory.

Virtual Memory

- Each process has its own separate virtual address space.
- Divides physical memory into blocks and allocates these blocks to different processes.
- Provides a mapping between blocks in physical memory and blocks on disk.
- Allows a process to execute with only portions of the process being in main memory.
- Also reduces program startup time.
- Provides protection to prevent processes from accessing blocks inappropriately.

Virtual Memory Terms

- Page is the name used for a block.
- Page fault is the name for a miss.
- Virtual address is the address produced by a CPU.
- Physical address is the address used to access main memory and typically cache as well.
- Page table is the data structure containing the mappings between virtual and physical addresses.
- Translation Lookaside Buffer is a cache that contains a portion of the page table.
**Page Tables**

- Number of entries in page table is equal to the number of virtual pages.
- Number of virtual pages is the size of the virtual address space divided by the page size.
- Each process has its own page table.
- A page table entry will typically contain a physical page number, resident bit, dirty bit, use bit, protection field (e.g., read only), disk address.

**Virtual Memory Questions**

- *Where can a block be placed in main memory?* It can be placed anywhere (fully associative) to reduce the miss rate.
- *How is a block found if it is in main memory?* The page table is indexed by the virtual page number and contains the physical page number.
- *Which block should be replaced on a virtual memory miss?* Use bits are used to approximate LRU.
- *What happens on a write?* A write-back, write-allocate policy is always used.

**Translation Lookaside Buffers**

- Most machines use special caches called TLBs that contain a portion of the entries in a page table.
- Each entry in the TLB contains a tag (portion of the virtual page number) and most of the information in a page table entry.
- TLBs are typically invalidated when a context switch is performed.
- TLBs are typically quite small to provide a very fast translation.
- There are typically separate TLBs for instructions and data to support simultaneous access due to pipelining.
- Many processors have multiple levels of TLBs, where the L1 TLBs are separate and the higher levels are unified.

**Virtual Memory Supports Multiprogramming**

- *Multiprogramming* means that several processes can concurrently share a computer.
- A process is the code, data, and any state information used in the execution of a program.
- A context switch means transferring control of the machine from one process to another.
- Proper protection must be provided to prevent a process from inappropriately affecting another process or itself.
- Virtual memory systems keep bits in the page table and TLB entries to indicate the type and level of access that the process has to each of the pages.
TLB and Page Table Example

- Page size is 512 bytes. TLB is direct-mapped and has 64 sets.

<table>
<thead>
<tr>
<th>TLB</th>
<th>Index</th>
<th>Page #</th>
<th>Tag</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>7</td>
<td>?</td>
<td>?</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>63</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page Table</th>
<th>Index</th>
<th>Page #</th>
<th>Resident</th>
<th>Dirty</th>
<th>Disk Addr</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>71</td>
<td>9</td>
<td>Yes</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- Given virtual address 0x8FDF, what is the physical address?
- Given virtual address 0x10DF, what is the physical address?

Selecting a Page Size

- Advantages of Using a Large Page Size
  - The bigger the page size, the fewer the entries in the page table and the smaller the page table.
  - Larger pages may allow the cache access to occur in parallel with the virtual to physical address translation.
  - Transferring larger pages to/from disk is more efficient since fewer seeks are required.
  - Will probably result in fewer TLB misses since there are fewer distinct pages referenced.

- Advantages of Using a Smaller Page Size
  - Will waste less space.
  - The miss penalty for a page fault and program startup time will decrease.

Fallacies and Pitfalls

- Pitfall: Too small an address space.
- Pitfall: Ignoring the impact of the operating system on the performance of the memory hierarchy.