Virtual Memory

- Use main memory as a "cache" for secondary memory
  - Allows efficient and safe sharing of memory among multiple programs
  - Provides the ability to easily run programs larger than the size of physical memory
  - Simplifies loading a program for execution by providing for code relocation (i.e., the code can be loaded anywhere in main memory)

- What makes it work? – again the Principle of Locality
  - A program is likely to access a relatively small portion of its address space during any period of time
  - Each program is compiled into its own address space – a "virtual" address space
  - During run-time each virtual address must be translated to a physical address (an address in main memory)

Address Translation

- A virtual address is translated to a physical address by a combination of hardware and software

<table>
<thead>
<tr>
<th>Virtual Address (VA)</th>
<th>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual page number</td>
<td>12 11 ... 0</td>
</tr>
<tr>
<td>Page offset</td>
<td></td>
</tr>
</tbody>
</table>

- So each memory request first requires an address translation from the virtual space to the physical space
  - A virtual memory miss (i.e., when the page is not in physical memory) is called a page fault

Two Programs Sharing Physical Memory

- A program’s address space is divided into pages (all one fixed size) or segments (variable sizes)
  - The starting location of each page (either in main memory or in secondary memory) is contained in the program’s page table

Address Translation Mechanisms

- Virtual page # Offset
- Physical page # Offset

- Main memory
- Disk storage

Page Table

(Relative) size of the memory at each level

Inclusive – what is in L1$ is a subset of what is in L2$ is a subset of what is in MM that is a subset of is in SM

Take advantage of the principle of locality to present the user with as much memory as is available in the cheapest technology at the speed offered by the fastest technology
Virtual Addressing with a Cache

- Thus it takes an extra memory access to translate a VA to a PA.

- This makes memory (cache) accesses very expensive (if every access was really two accesses).

- The hardware fix is to use a Translation Lookaside Buffer (TLB) – a small cache that keeps track of recently used address mappings to avoid having to do a page table lookup.

Translation Lookaside Buffers (TLBs)

- Just like any other cache, the TLB can be organized as fully associative, set associative, or direct mapped.

- TLB access time is typically smaller than cache access time (because TLBs are much smaller than caches).

  - TLBs are typically not more than 128 to 256 entries even on high end machines.

Some Virtual Memory Design Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Paged VM</th>
<th>TLBs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total size (words)</td>
<td>16,000 to 250,000</td>
<td>16 to 512 entries</td>
</tr>
<tr>
<td>Total size (KB)</td>
<td>250,000 to 1,000,000,000</td>
<td>0.25 to 16</td>
</tr>
<tr>
<td>Block size (B)</td>
<td>4000 to 64,000</td>
<td>4 to 32</td>
</tr>
<tr>
<td>Miss penalty (clocks)</td>
<td>10,000,000 to 100,000,000,000</td>
<td>10 to 1000</td>
</tr>
<tr>
<td>Miss rates</td>
<td>0.00001% to 0.0001%</td>
<td>0.01% to 2%</td>
</tr>
</tbody>
</table>

Making Address Translation Fast

- TLB access time is typically smaller than cache access time (because TLBs are much smaller than caches).

A TLB in the Memory Hierarchy

- A TLB miss – is it a page fault or merely a TLB miss?
  - If the page is loaded into main memory, then the TLB miss can be handled (in hardware or software) by loading the translation information from the page table into the TLB.
  - Takes 10's of cycles to find and load the translation info into the TLB.
  - If the page is not in main memory, then it's a true page fault.
  - Takes 1,000,000's of cycles to service a page fault.

- TLB misses are much more frequent than true page faults.

Two Machines’ Cache Parameters

<table>
<thead>
<tr>
<th></th>
<th>Intel P4</th>
<th>AMD Opteron</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLB organization</td>
<td>1 TLB for instructions and 1 TLB for data</td>
<td>2 TLBs for instructions and 2 TLBs for data</td>
</tr>
<tr>
<td></td>
<td>Both 4-way set associative</td>
<td>Both L1 TLBs fully associative with –LRU replacement</td>
</tr>
<tr>
<td></td>
<td>Both use –LRU replacement</td>
<td>Both L2 TLBs are 4-way set associative with round-robin LRU</td>
</tr>
<tr>
<td></td>
<td>128 entries</td>
<td>Both have 128 entries</td>
</tr>
<tr>
<td></td>
<td>TLB misses handled in hardware</td>
<td>Both L1 TLBs have 40 entries</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Both L2 TLBs have 512 entries</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TLB misses handled in hardware</td>
</tr>
</tbody>
</table>
TLB Event Combinations

<table>
<thead>
<tr>
<th>TLB Event</th>
<th>Page Table</th>
<th>Cache Hit</th>
<th>Possible?</th>
<th>Under what circumstances?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hit</td>
<td>Hit</td>
<td>Hit</td>
<td>Yes</td>
<td>- what we want!</td>
</tr>
<tr>
<td>Hit</td>
<td>Hit</td>
<td>Miss</td>
<td>Yes</td>
<td>- although the page table is not checked if the TLB hits</td>
</tr>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Miss</td>
<td>Yes</td>
<td>- TLB miss, PA in page table</td>
</tr>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Miss</td>
<td>Yes</td>
<td>- TLB miss, PA in page table, but data not in cache</td>
</tr>
<tr>
<td>Miss</td>
<td>Miss</td>
<td>Miss</td>
<td>Yes</td>
<td>- page fault</td>
</tr>
<tr>
<td>Miss</td>
<td>Miss</td>
<td>Hit</td>
<td>Yes</td>
<td>- TLB translation not possible if page is not present in memory</td>
</tr>
<tr>
<td>Miss</td>
<td>Miss</td>
<td>Hit</td>
<td>Impossible- data not allowed in cache if page is not in memory</td>
<td></td>
</tr>
</tbody>
</table>

Reducing Translation Time

- Can overlap the cache access with the TLB access
  - Works when the high order bits of the VA are used to access the TLB while the low order bits are used as index into cache

Why Not a Virtually Addressed Cache?

- A virtually addressed cache would only require address translation on cache misses
  - CPU
  - VA
  - Translation
  - PA
  - Main Memory
  - But
  - Two different virtual addresses can map to the same physical address (when processes are sharing data), i.e., two different cache entries hold data for the same physical address - synonyms
  - Must update all cache entries with the same physical address or the memory becomes inconsistent

The Hardware/Software Boundary

- What parts of the virtual to physical address translation is done by or assisted by the hardware?
  - Translation Lookaside Buffer (TLB) that caches the recent translations
    - TLB access time is part of the cache hit time
    - May allot an extra stage in the pipeline for TLB access
  - Page table storage, fault detection and updating
    - Page faults result in interrupts (precise) that are then handled by the OS
    - Hardware must support (i.e., update appropriately) Dirty and Reference bits (e.g., ~LRU) in the Page Tables
  - Disk placement
    - Bootstrap (e.g., out of disk sector 0) so the system can service a limited number of page faults before the OS is even loaded

Summary

- The Principle of Locality:
  - Program likely to access a relatively small portion of the address space at any instant of time
    - Temporal Locality: Locality in Time
    - Spatial Locality: Locality in Space
- Caches, TLBs, Virtual Memory all understood by examining how they deal with the four questions
  1. Where can block be placed?
  2. How is block found?
  3. What block is replaced on miss?
  4. How are writes handled?
- Page tables map virtual address to physical address
  - TLBs are important for fast translation
Next Lecture and Reminders

- Next lecture
  - Reading assignment – PH 8.1-8.2

- Reminders
  - Lab 2 part 2 Due: today
  - HW 5 Due: One week from today, April 18