Caches and TLBs

February 26, 2014
Anthony D. Joseph
http://inst.eecs.berkeley.edu/~cs162
Review: Multi-Segment

- Segment map resides in processor
  - Segment number mapped into base/limit pair
  - Base added to offset to generate physical address
  - Error check catches offset out of range
- As many chunks of physical memory as entries
  - Segment addressed by portion of virtual address
  - However, could be included in instruction instead:
    » x86 Example: mov [es:bx],ax.
- What is “V/N” (valid / not valid)?
  - Can mark segments as invalid; requires check as well
Review: Paging

- Page Table (One per process)
  - Resides in physical memory
  - Contains physical page and permission for each virtual page
    » Permissions include: Valid bits, Read, Write, etc

- Virtual address mapping
  - Offset from Virtual address copied to Physical Address
    » Example: 10 bit offset ⇒ 1024-byte pages
  - Virtual page # is all remaining bits
    » Example for 32-bits: 32-10 = 22 bits, i.e. 4 million entries
    » Physical page # copied from table into physical address
  - Check Page Table bounds and permissions
Review: Two-level Page Table

- Tree of Page Tables
- Tables fixed size (1024 entries)
  - On context-switch: save single PageTablePtr register
- Valid bits on Page Table Entries
  - Don’t need every 2\textsuperscript{nd}-level table
  - Even when exist, 2\textsuperscript{nd}-level tables can reside on disk if not in use
Example

• Data in memory, no cache:

  Processor

  Access time = 100ns

  Main Memory (DRAM)

• Data in memory, 10ns cache:

  Processor

  Second Level Cache (SRAM)

  Average Access time = (Hit Rate x HitTime) + (Miss Rate x MissTime)

  • HitRate + MissRate = 1
Sources of Cache Misses

- **Compulsory** (cold start): first reference to a block
  - “Cold” fact of life: not a whole lot you can do about it

- **Capacity**:
  - Cache cannot contain all blocks access by the program
  - Solution: increase cache size

- **Conflict** (collision):
  - Multiple memory locations mapped to same cache location
  - Solutions: increase cache size, or increase associativity

- **Two others**:
  - **Coherence** (Invalidation): other process (e.g., I/O) updates mem
  - **Policy**: Due to non-optimal replacement policy

Average Access time = (Hit Rate x HitTime) + (Miss Rate x MissTime)
5min Break
Where does a Block Get Placed in a Cache?

- Example: Block 12 placed in 8 block cache

32-Block Address Space:

Direct mapped:
block 12 (01100) can go only into block 4 (12 mod 8)

Set associative:
block 12 can go anywhere in set 0

Fully associative:
block 12 can go anywhere
Caching Applied to Address Translation

• Question is one of page locality: does it exist?
  – Instruction accesses spend a lot of time on the same page (since accesses sequential)
  – Stack accesses have definite locality of reference
  – Data accesses have less page locality, but still some…

• Can we have a TLB hierarchy?
  – Sure: multiple levels at different sizes/speeds
Reducing Translation Time Further

• As described, TLB lookup is in serial with cache lookup:

Machines with TLBs go one step further: they overlap TLB lookup with cache access.
  – Works because offset available early

Virtual Address

<table>
<thead>
<tr>
<th>V page no.</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

TLB Lookup

<table>
<thead>
<tr>
<th>V</th>
<th>Access Rights</th>
<th>PA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

P page no.

<table>
<thead>
<tr>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
</tr>
</tbody>
</table>

Physical Address
Overlapping TLB & Cache Access

- Main idea:
  - Offset in virtual address exactly covers the “cache index” and “byte select”
  - Thus can select the cached byte(s) in parallel to perform address translation

<table>
<thead>
<tr>
<th>Virtual Page #</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag / page #</td>
<td>index</td>
</tr>
</tbody>
</table>

virtual address

physical address