Virtual Memory III

Miki Lustig

Address Translation Using TLB

Virtual Address

VPN

TLB Tag | TLB Index | Page Offset

Virtual Address

Tag | Index | Offset

Data

Cache

Physical Address

VPN

TLB Tag | TLB Index | Page Offset

Page Table

PPP | Page Offset

Physical Address

Tag | Index | Offset

Note: TIO for VA & PA unrelated

Fetching Data on a Memory Read

1) Check TLB (input: VPN, output: PPN)
   - TLB Hit: Fetch translation, return PPN
   - TLB Miss: Check page table (in memory)
     - Page Table Hit: Load page table entry into TLB
     - Page Table Miss (Page Fault): Fetch page from disk to memory, update corresponding page table entry, then load entry into TLB

2) Check cache (input: PPN, output: data)
   - Cache Hit: Return data value to processor
   - Cache Miss: Fetch data value from memory, store it in cache, return it to processor

Page Faults

- Load the page off the disk into a free page of memory
  - Switch to some other process while we wait
- Interrupt thrown when page loaded and the process' page table is updated
  - When we switch back to the task, the desired data will be in memory
- If memory full, replace page (LRU), writing back if necessary, and update both page table entries
  - Continuous swapping between disk and memory called "thrashing"

Where Are TLBs Located?

- Which should we check first: Cache or TLB?
  - Can cache hold requested data if corresponding page is not in physical memory? No
  - With TLB first, does cache receive VA or PA?

- Can cache hold requested data if corresponding page is not in physical memory? No
- PA split two different ways!

Question: How many bits wide are the following?

- 16 KiB pages
- 40-bit virtual addresses
- 64 GiB physical memory
- 2-way set associative TLB with 512 entries

Valid | Dirty | Ref | Access Rights | TLB Tag | PPN

X | X | X | XX

TLB Tag | TLB Index | TLB Entry

A) 12 14 38
B) 18 8 45
C) 14 12 40
D) 17 9 43

Notice that it is now the TLB that does translation, not the Page Table!
Performance Metrics

- VM performance also uses Hit/Miss Rates and Miss Penalties
  - **TLB Miss Rate**: Fraction of TLB accesses that result in a TLB Miss
  - **Page Table Miss Rate**: Fraction of PT accesses that result in a page fault
- Caching performance definitions remain the same
  - Somewhat independent, as TLB will always pass PA to cache regardless of TLB hit or miss

**Question**: A program tries to load a word at X that causes a TLB miss but not a page fault. Are the following statements TRUE or FALSE?

1) The page table does not contain a valid mapping for the virtual page corresponding to the address X
2) The word that the program is trying to load is present in physical memory

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>B</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>C</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>D</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

Data Fetch Scenarios

- Are the following scenarios for a single data access possible?
  - TLB Miss, Page Fault
  - TLB Hit, Page Table Hit
  - TLB Miss, Cache Hit
  - Page Table Hit, Cache Miss
  - Page Fault, Cache Hit

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>B</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>C</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>D</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

VM Performance

- Virtual Memory is the level of the memory hierarchy that sits below main memory
  - TLB comes before cache, but affects transfer of data from disk to main memory
  - Previously we assumed main memory was lowest level, now we just have to account for disk accesses
- Same CPI, AMAT equations apply, but now treat main memory like a mid-level cache

Typical Performance Stats

**Caching**
- cache entry
- cache block (=32 bytes)
- cache miss rate (1% to 20%)
- cache hit (=1 cycle)
- cache miss (=100 cycles)

**Demand paging**
- page frame
- page (=4Ki bytes)
- page miss rate (<0.001%)
- page hit (=100 cycles)
- page miss (=5M cycles)

Impact of Paging on AMAT (1/2)

- Memory Parameters:
  - L1 cache hit = 1 clock cycles, hit 95% of accesses
  - L2 cache hit = 10 clock cycles, hit 60% of L1 misses
  - DRAM = 200 clock cycles (=100 nanoseconds)
  - Disk = 20,000,000 clock cycles (=10 milliseconds)
- **Average Memory Access Time (no paging):**
  - 1 + 5%×10 + 5%×40%×200 = 5.5 clock cycles
- **Average Memory Access Time (with paging):**
  - 5.5 (AMAT with no paging) + ?
Impact of Paging on AMAT (2/2)

- Average Memory Access Time (with paging) =
  - $5.5 + 5\% \times 40\% \times (1-HR_{\text{Mem}}) \times 20,000,000$
- AMAT if $HR_{\text{Mem}} = 99\%$
  - $5.5 + 0.02 \times 0.01 \times 20,000,000 = 4005.5 (\approx 728\times \text{slower})$
  - 1 in 20,000 memory accesses goes to disk: 10 sec program takes 2 hours!
- AMAT if $HR_{\text{Mem}} = 99.9\%$
  - $5.5 + 0.02 \times 0.001 \times 20,000,000 = 405.5$
- AMAT if $HR_{\text{Mem}} = 99.9999\%$
  - $5.5 + 0.02 \times 0.000001 \times 20,000,000 = 5.9$

Impact of TLBs on Performance

- Each TLB miss to Page Table $\sim$ L1 Cache miss
- TLB Reach: Amount of virtual address space that can be simultaneously mapped by TLB:
  - TLB typically has 128 entries of page size 4-8 KiB
  - $128 \times 4 \text{ KiB} = 512 \text{ KiB} = \text{just 0.5 MiB}$
- What can you do to have better performance?
  - Multi-level TLBs $\quad$ Conceptually same as multi-level caches
  - Variable page size (segments)
  - Special situationally-used “superpages”

Aside: Context Switching

- How does a single processor run many programs at once?
  - Context switch: Changing of internal state of processor (switching between processes)
    - Save register values (and PC) and change value in Page Table Base register
- What happens to the TLB?
  - Current entries are for different process
  - Set all entries to invalid on context switch

Virtual Memory Summary

- User program view:
  - Contiguous memory
  - Start from some set VA
    - “Ininitely” large
  - Is the only running program
  - Non-contiguous memory
  - Start wherever available memory is
    - Finite size
  - Many programs running simultaneously
- Virtual memory provides:
  - Illusion of contiguous memory
  - All programs starting at same set address
    - Illusion of “infinite memory” ($2^{32}$ or $2^{64}$ bytes)
  - Protection, Sharing
- Implementation:
  - Divide memory into chunks (pages)
  - OS controls page table that maps virtual into physical addresses
    - Memory as a cache for disk
  - TLB is a cache for the page table