Page Tables Revisited

Learning Outcomes

- An understanding of virtual linear array page tables, and their use on the MIPS R3000.
- Exposure to alternative page table structures beyond two-level and inverted page tables.

Two-level Translation

R3000 TLB Refill

- Can be optimised for TLB refill only
  - Does not need to check the exception type
  - Does not need to save any registers
    - It uses a specialised assembly routine that only uses k0 and k1.
    - Does not check if PTE exists
  - Assumes virtual linear array – see extended OS notes
- With careful data structure choice, exception handler can be made very fast

How does this work?

Virtual Linear Array page table

- Assume a 2-level PT
- Assume 2nd-level PT nodes are in virtual memory
- Assume all 2nd-level nodes are allocated contiguously ⇒ 2nd-level nodes form a contiguous array indexed by page number
Virtual Linear Array Operation

- Index into 2nd level page table **without** referring to root PT
- Simply use the full page number as the PT index!
- Leave unused parts of PT unmapped!
- If access is attempted to unmapped part of PT, a secondary page fault is triggered
  - This will load the mapping for the PT from the root PT
  - Root PT is kept in physical memory (cannot trigger page faults)

Virtual Linear Array Page Table

- Use Context register to simply load PTE by indexing a PTE array in virtual memory
- Occasionally, will get double faults
  - A TLB miss, while servicing a TLB miss
  - Handled by general exception handler

PTEbase in virtual memory in kseg2
- Protected from user access

c0 Context Register

- `c0_Context = PTEBase + 4 * PageNumber`
  - PTEs are 4 bytes
  - PTEBase is the base local of the page table array (note: aligned on 4 MB boundary)
  - PTEBase is (re)initialised by the OS whenever the page table array is changed
    - E.g. on a context switch
  - After an exception, `c0_Context` contains the address of the PTE required to refill the TLB.

Code for VLA TLB refill handler

- Move the PTE into EntryLo.
- Write EntryLo into random TLB entry.
- Load address of instruction to return to
- Load the PTE
  - This load can cause a TLB refill miss itself, but this miss is handled by the general exception vector.
  - The general exception vector has to understand this situation and deal with it appropriately

Software-loaded TLB

- **Pros**
  - Can simplify hardware design
  - Provide greater flexibility in page table structure
- **Cons**
  - Typically have slower refill times than hardware managed TLBs.
Design Tradeoffs for Software-Managed TLBs
David Nagle, Richard Uhlig, Tim Stanley, Stuart Sechrest, Trevor Mudge, Richard Brown
ISCA '93 Proceedings of the 20th annual international symposium on computer architecture

Trends at the time

- Operating systems
  - moving functionality into user processes
  - making greater use of virtual memory for mapping data structures held within the kernel.
- RAM is increasing
  - TLB capacity is relatively static
- Statement:
  - Trends place greater stress upon the TLB by increasing miss rates and hence, decreasing overall system performance.
  - True/False? How to evaluate?

Note the TLB miss costs

- What is expected to be the common case?
Measurement Results

<table>
<thead>
<tr>
<th></th>
<th>L27</th>
<th>L2L</th>
<th>L2K</th>
<th>L3</th>
<th>Instal</th>
<th>Miss</th>
<th>Validity</th>
<th>% of Total Misses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mach 3.0</td>
<td>56.82</td>
<td>45.68</td>
<td>45.68</td>
<td>56.82</td>
<td>45.68</td>
<td>45.68</td>
<td>56.82</td>
<td>45.68</td>
</tr>
<tr>
<td>Mach 3.0+AFScuf</td>
<td>56.82</td>
<td>45.68</td>
<td>45.68</td>
<td>56.82</td>
<td>45.68</td>
<td>45.68</td>
<td>56.82</td>
<td>45.68</td>
</tr>
</tbody>
</table>

Table 5: Number of TLB Misses

<table>
<thead>
<tr>
<th></th>
<th>Time Total TLB Misses (sec)</th>
<th>L2L</th>
<th>L2K</th>
<th>L3</th>
<th>Instal</th>
<th>Miss</th>
<th>Validity</th>
<th>% of Total Misses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mach 3.0</td>
<td>56.82</td>
<td>45.68</td>
<td>45.68</td>
<td>56.82</td>
<td>45.68</td>
<td>45.68</td>
<td>56.82</td>
<td>45.68</td>
</tr>
<tr>
<td>Mach 3.0+AFScuf</td>
<td>56.82</td>
<td>45.68</td>
<td>45.68</td>
<td>56.82</td>
<td>45.68</td>
<td>45.68</td>
<td>56.82</td>
<td>45.68</td>
</tr>
</tbody>
</table>

Table 6: Time Spent Handling TLB Misses

Specialising the L2/L1K miss vector

<table>
<thead>
<tr>
<th>Type of PTE Miss</th>
<th>Count</th>
<th>Precision Total Cost from Table 6 (sec)</th>
<th>New Total Cost (sec)</th>
<th>Time Saved (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mach 3.0+AFScuf</td>
<td>L1L2</td>
<td>36,123,456</td>
<td>36,123,456</td>
<td>36,123,456</td>
</tr>
<tr>
<td>Mach 3.0</td>
<td>L1L2</td>
<td>36,123,456</td>
<td>36,123,456</td>
<td>36,123,456</td>
</tr>
</tbody>
</table>

Table 7: Reconstructed Cost of TLB Misses Given Additional Miss Vectors (Mach 3.2)

Other performance improvements?

- In Paper
  - Pinned slots
  - Increased TLB size
  - TLB associativity
- Other options
  - Bigger page sizes
  - Multiple page sizes
Itanium Page Table

• Takes a bet each way
• Loading
  – software
  – two different format hardware walkers
• Page table
  – software defined
  – Virtual linear array
  – Hashed

what about the P4?

• i.e. 32-bit x86 architecture.

P4

• Sophisticated, supports:
  – demand paging
  – pure segmentation
  – segmentation with paging
• Heart of the VM architecture
  – Local Descriptor Table (LDT)
  – Global Descriptor Table (GDT)
• LDT
  – 1 per process
  – describes segments local to each process (code, stack, data, etc.)
• GDT
  – shared by all programs
  – describes system segments (including OS itself)

P4

• To access a segment P4
  – loads a selector in 1 of the segment registers

• a P4 selector:
• a P4 selector:
  - when selector is in register, corresponding segment descriptor is
    - fetched by MMU
    - loaded in internal MMU registers
  - Next, segment descriptor is used to handle memory reference (discussed later)

- finds a P4 code segment descriptor

• calculating a linear address from selector+offset
  - 12 bit linear address

IF no paging used: we are done
  ➔ this is the physical address

ELSE
  ➔ linear address interpreted as virtual address
  ➔ paging again!
P4

• Many OSs:
  – BASE=0
  – LIMIT=MAX

• no segmentation at all

That is it!