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 multi-level and inverted page tables.

Two-level Translation

Virtual Linear Array page table

• Uses a page table array indexed by page number
• Page table array is in virtual memory with only used pages of the array allocated in physical memory
• A second page table root node has translations for the page table itself

Virtual Linear Array Operation

• Index into page table array 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)
R3000 TLB Refill

- Dedicated exception handler
- 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

An example routine:
```
mfc0 k1, C0_CONTEXT
mfc0 k0, C0_EPC # mfc0 delay slot
lw k1, 0(k1) # may double fault (k0 = orig EPC)
nop
mtc0 k0, C0_EPC
mtc0 k1, C0_EPC
TL0
tlbwr
jr k0
rfe
```

How does this work?

c0 Context Register

```
| 31 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| PTEBase | Bad VPN | 0 |
```

- 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.

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
```

Code for VLA TLB refill handler

```
mfc0 k1, C0_CONTEXT
mfc0 k0, C0_EPC # mfc0 delay slot
lw k1, 0(k1), # may double fault (k0 = orig EPC)
nop
mtc0 k0, C0_EPC
mtc0 k1, C0_EPC
TL0
tlbwr
jr k0
rfe
```

Load PTE address from context register
Move the PTE into EntryLo.
Write EntryLo into random TLB entry.
Return from the exception

Load the PTE: Note: 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?

Figure 1: Tapeworm
The Tapeworm TLB simulator is built into the operating system and is invoked whenever there is a real TLB miss. The simulator uses the real TLB misses to simulate its own TLB configuration(s). Because the simulator resides in the operating system, Tapeworm captures the dynamic nature of the system and avoids the problems associated with simulators driven by static traces.

Figure 2: Page Table Structure in OSF/1 and Mach 3.0
The Mach page table has a simple structure with the relatively small number of page sizes needed. The OSF/1 page table, on the other hand, is implemented as a tree of page tables. The top of the page table structure contains the policy functions, while the leaf nodes contain the page table entries. The OSF/1 page table is depicted in the figure. The page table structure is similar for both OSF/1 and Mach. However, Mach uses a small number of page sizes, which are depicted as the leaves of the structure. The leaf nodes contain the page table entries for each of the page sizes.

<table>
<thead>
<tr>
<th>TLB Miss Type</th>
<th>Ultrix</th>
<th>OSF/1</th>
<th>Mach 3.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1U</td>
<td>18</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>L1N</td>
<td>355</td>
<td>355</td>
<td>355</td>
</tr>
<tr>
<td>L2</td>
<td>454</td>
<td>511</td>
<td>457</td>
</tr>
<tr>
<td>L3</td>
<td>354</td>
<td>354</td>
<td>286</td>
</tr>
<tr>
<td>Modify</td>
<td>375</td>
<td>436</td>
<td>406</td>
</tr>
<tr>
<td>Invalid</td>
<td>339</td>
<td>277</td>
<td>267</td>
</tr>
</tbody>
</table>

Table 2: Costs for Different TLB Miss Types
This table shows the number of machine cycles (as 60 nanoseconds) required to service different types of TLB misses. To determine these values, a simulator was used to collect a trace of actual timings for each type of miss. The simulator TLB miss type in the first column describes the type of miss. The number of cycles is reported in the second column. The entries in the third column are for Mach 3.0, while the entries in the last column are for Ultrix. The number of cycles is reported for each type of miss. The entries in the third column are for Mach 3.0, while the entries in the last column are for Ultrix.
Note the TLB miss costs

• What is expected to be the common case?

Measurement Results

<table>
<thead>
<tr>
<th>System</th>
<th>Total Run Time (sec)</th>
<th>L1/0</th>
<th>L1/1</th>
<th>L2</th>
<th>L3</th>
<th>Invalid</th>
<th>Modify</th>
<th>% of TLB Misses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unix</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Mach 3.0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Mach 3.0 + AFSOut</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Specialising the L2/L1K miss vector

<table>
<thead>
<tr>
<th>Type of PTE Miss</th>
<th>Counts</th>
<th>Previous Total Cost from Table 5 (sec)</th>
<th>New Total Cost (sec)</th>
<th>Time Saved (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mach 3.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L1/0</td>
<td>123456</td>
<td>123456</td>
<td>123456</td>
<td>123456</td>
</tr>
<tr>
<td>L1/1</td>
<td>123456</td>
<td>123456</td>
<td>123456</td>
<td>123456</td>
</tr>
<tr>
<td>L2</td>
<td>123456</td>
<td>123456</td>
<td>123456</td>
<td>123456</td>
</tr>
<tr>
<td>L3</td>
<td>123456</td>
<td>123456</td>
<td>123456</td>
<td>123456</td>
</tr>
<tr>
<td>Invalid</td>
<td>123456</td>
<td>123456</td>
<td>123456</td>
<td>123456</td>
</tr>
</tbody>
</table>

Table 7: Computed Cost of TLB Misses Given Additional Miss Vectors (Mach 3.0)
Itanium Page Table

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

That is it!