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.

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

An example routine:
```
mfc0 k1,c0_CONTEXT
mfc0 k0,0,C0_EPC  # mfc0 delay
lw k1,0(k1)  # slot
# fault  (k0
nop
mfc0 k1,0,E0
nop
tlbwr
jr k0
rfe
```

How does this work?

Two-level Translation

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.
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 \Rightarrow 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

Code for VLA TLB refill handler
- Load address of instruction to return to
- 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 Sohrestr 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>System</th>
<th>Total TLB Size (kB)</th>
<th>LUS</th>
<th>LUS</th>
<th>LK</th>
<th>LK</th>
<th>Total</th>
<th>% of Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>COSS 1</td>
<td>324,220</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14%</td>
</tr>
<tr>
<td>Mach 3.0</td>
<td>256,256</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14%</td>
</tr>
<tr>
<td>Mach 3.0 + AFSconf</td>
<td>256,256</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System</th>
<th>Total TLB Size (kB)</th>
<th>LUS</th>
<th>LUS</th>
<th>LK</th>
<th>LK</th>
<th>Total</th>
<th>% of Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>COSS 1</td>
<td>324,220</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14%</td>
</tr>
<tr>
<td>Mach 3.0</td>
<td>256,256</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14%</td>
</tr>
<tr>
<td>Mach 3.0 + AFSconf</td>
<td>256,256</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14</td>
<td>14%</td>
</tr>
</tbody>
</table>

**Specialising the L2/L1K miss vector**

<table>
<thead>
<tr>
<th>Type of TIE Miss</th>
<th>Count</th>
<th>Previous Total Cost (sec)</th>
<th>New Total Cost (sec)</th>
<th>Time Saved (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1L1</td>
<td>30,131</td>
<td>36.15</td>
<td>30.15</td>
<td>6.00</td>
</tr>
<tr>
<td>L1L2</td>
<td>100,981</td>
<td>8.69</td>
<td>7.76</td>
<td>0.93</td>
</tr>
<tr>
<td>L1L3</td>
<td>2,445,265</td>
<td>42.89</td>
<td>20.89</td>
<td>22.00</td>
</tr>
<tr>
<td>L1K</td>
<td>618,001</td>
<td>13.85</td>
<td>19.71</td>
<td>5.86</td>
</tr>
<tr>
<td>L2</td>
<td>137,245</td>
<td>3.81</td>
<td>3.81</td>
<td>0.00</td>
</tr>
<tr>
<td>L3</td>
<td>100,041</td>
<td>27.70</td>
<td>27.70</td>
<td>0.00</td>
</tr>
<tr>
<td>Total</td>
<td>33,682,153</td>
<td>32.30</td>
<td>16.20</td>
<td>16.10</td>
</tr>
</tbody>
</table>

**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

That is it!