Embra

TLB Layout

<table>
<tr>
<td>Embra</td>
<td>MMU Reloc Array (All entries have same ASID, Indexed by VPN)
<table>
<tr><th>Phys Page Addr</th><th>Protect Bits</th></tr>
<tr><td></td><td></td></tr>
<tr><td></td><td></td></tr>
</table>
(Embra) TLB (Virtual → Physical Address)
<table>
<tr><th>ASID</th><th>VPN</th><th>PPN</th></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td></tr>
</table>
</td>
</tr>
<tr>
<td>Host</td>
<td>(Host) TLB (Physical → Machine Address)
<table>
<tr><th>ASID</th><th>PPN</th><th>MPN</th></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td></tr>
</table>
</td>
</tr>
</table>

General

- How is Embra simulation different than Shade simulation?
  - Embra simulates a machine, Shade simulates a process (instruction simulator)
  - Embra is slower than Shade due to various things such as:
    - Increased memory access overhead
    - Chaining difficulties and checking
  - Embra supports these while shade doesn't
    - Multiple virtual address spaces
    - Privileged instructions
    - MMU Address translation
    - Exceptions and interrupts
- TLB Misses -- Why are they important?
  - We must deal with exceptions by calling into OS so this can be a major source of overhead...
  - Embra can deal with this by modeling a larger TLB.

MMU Relocation Array

- MMU simulation performance is critical. Why? Why isn't this an issue in Shade?
  - In Embra each process has its own virtual address space, so it is not as simple as adding a simple offset like in Shade. Performance is critical because it is done frequently -- on every memory access the generated virtual address must be translated to its corresponding physical address. This means every load/store

and every instruction fetch will need to access the MMU.

- TLBs are fully associative, yet Embra doesn't implement a fully-associative TLB; why?
    - The short answer is that it is slower. In the fully-associative TLB case we would need to verify that the entry matches both the current ASID and VA and then perform address translation, whereas when we use the MMU Relocation array we simply lookup the address by the VPN. Since we are performing address translation so frequently we need something that is fast.
- MMU Relocation Array
    - What is the MMU relocation array, and how is it used?
        - The MMU relocation array is an array indexed by the Virtual Page Number (VPN). Each entry contains the Physical Page Address and the Protection Bits for the page. Address translation is done through here instead of Embra's simulated TLB. The MMU relocation array is synced with the TLB and thus valid entries in the TLB correspond to valid entries in the MMU relocation array.
    - MMU Lookup Protocol:
        - Use the virtual page number from the virtual address to perform lookup into the MMU relocation array to retrieve the physical page address
        - If in MMU, hit. Perform permission check and get the physical address using physical page address + offset
        - If not, miss. Raise an exception and call into OS to install entry into TLB.
            - Eviction: not addressed directly by authors
    - Context Switching / ASID
        - ASIDs are not modeled for the purposes of speed. The MMU relocation array only contains entries for the current ASID, making address translation faster. On context switch the MMU relocation array is flushed and all relevant entries in the Embra TLB is loaded into the MMU relocation array, eliminating the need to perform the normal TLB lookup on every memory access.
        - Why do we flush the relocation array?
            - This saves space by not having to have a MMU Relocation Array for each ASID.
            - Also, Embra does not check the ASID on MMU lookup for performance purposes.
    - How would this approach scale as virtual address spaces grow?
        - The MMU relocation array will grow as the address space grows since the number of entries is proportional to the number of virtual pages (virtual address space size / page size). This means that it could take a large amount of VM in the simulator!
- Why is instruction fetch less of a performance concern?
    - Address lookup is only done once, when it is translated into the translation cache. Compare this to ld/st which does it on each access. The assumption of course is that the translated blocks may not span pages!

- Uncached Load/Stores
  - Why do we have these?
    - We need these for accesses to untranslated instructions, I/O devices, and DMA, where we need uncached values.
  - How is this done?
    - Embra cannot distinguish between cached and uncached load/stores, so it sets uncached pages to invalid entries to cause a TLB exception, which forwards the uncached access to the appropriate device handler. This is not the most efficient (slow) but it doesn't occur often so it's okay.
- KSEG0
  - Why is this a concern?
    - Recall that in MIPS the OS has access to all physical memory using KSEG0. Access to this region is done without using the TLB to avoid TLB misses. However, in our case this may be problematic since the KSEG0 virtual address space is different from its physical address space.
  - How do we fix this?
    - We fill the MMU relocation array with KSEG0 entries mapping its virtual address range to their physical address range. Access to KSEG0 is thus handled as usual.

Chaining

- Why are chains in Embra harder to implement that chains in Shade?
  - Difficulties arise from having separate virtual address spaces per process. This means that processes could have different code mapped at the same virtual address, so Embra uses physical addresses in the translation cache.
  - However, we could also have the case where two processes map into the same code page at the same virtual address but jump to different pages at different virtual addresses. This case requires a check before execution of the translated code that verifies that the physical address of the current PC is the same as the physical address of the executing translation. If they are, we execute the rest of the translated block. If they are not, the full lookup is performed and the old chain value is overwritten. (But this is slow since it goes back to the dispatch loop!)
  - This all results in a high dispatch lookup overhead.
- How does speculative chaining help overcome this?
  - Speculative chaining chains register indirect jumps to code that verifies if the destination code is at the correct virtual and physical address. If the current PC virtual address matches the virtual address that generated the destination code, the chain is valid. Otherwise, the full lookup is performed.

Customized Translation

- What is the vQC and how is it used?
  - It is an array that tracks the status of each cache line in terms of its presence in the TLB/cache. Before each MMU lookup, the vQC is used to lookup the address. On success, it proceeds to the MMU lookup stage, otherwise the appropriate exception is raised and handled.

        - Its size is determined by (virt address space size / cache line size), following a similar theme here of sparse, large arrays for faster accesses. Thus memory overhead becomes much larger as our address space grows!

Evaluation

- Table 5.1 Observations
    - Embra is fast and depends on the code / workload
- How much does cache simulation impact Embra's performance?
    - We see slowdown of 3-9x without cache simulation and 9-20x slowdown with cache simulation