Operating Systems & Memory Systems: Address Translation & Caches

CPS 220
Professor Alvin R. Lebeck
Fall 2001

Outline

• Review
• TLBs
• Page Table Designs
• Interaction of VM and Caches

Admin
• HW #4 Due today
• Project Status report due Thursday
Virtual Memory: Motivation

- **Process = Address Space + thread(s) of control**
- **Address space = PA**
  - programmer controls movement from disk
  - protection?
  - relocation?
- **Linear Address space**
  - larger than physical address space
  » 32, 64 bits v.s. 28-bit physical (256MB)
- **Automatic management**

Virtual Memory

- **Process = virtual address space + thread(s) of control**
- **Translation**
  - VA → PA
  - What physical address does virtual address A map to
  - Is VA in physical memory?
- **Protection (access control)**
  - Do you have permission to access it?
Segmented Virtual Memory

- Virtual address (2^{32}, 2^{64}) to Physical Address mapping (2^{30})
- Variable size, base + offset, contiguous in both VA and PA

Paged Virtual Memory

- Virtual address (2^{32}, 2^{64}) to Physical Address mapping (2^{28})
  - virtual page to physical page frame
- Fixed Size units for access control & translation
Page Table

- Kernel data structure (per process)
- Page Table Entry (PTE)
  - VA -> PA translations (if none page fault)
  - access rights (Read, Write, Execute, User/Kernel, cached/uncached)
  - reference, dirty bits
- Many designs
  - Linear, Forward mapped, Inverted, Hashed, Clustered
- Design Issues
  - support for aliasing (multiple VA to single PA)
  - large virtual address space
  - time to obtain translation

Alpha VM Mapping (Forward Mapped)

- “64-bit” address divided into 3 segments
  - seg0 (bit 63=0) user code/heap
  - seg1 (bit 63 = 1, 62 = 1) user stack
  - kseg (bit 63 = 1, 62 = 0) kernel segment for OS
- Three level page table, each one page
  - Alpha 21064 only 43 unique bits of VA
  - (future min page size up to 64KB => 55 bits of VA)
- PTE bits; valid, kernel & user read & write enable (No reference, use, or dirty bit)
  - What do you do for replacement?
Inverted Page Table (HP, IBM)

- One PTE per page frame
  - only one VA per physical frame
- Must search for virtual address
- More difficult to support aliasing
- Force all sharing to use the same VA

Hash Anchor Table (HAT)

Virtual page number   Offset

Inverted Page Table (IPT)

VA   PAST

Intel Pentium Segmentation + Paging

Logical Address

Segment Base Address

Global Descriptor Table (GDT)

Segment Descriptor

Segment Dir

Physical Address Space

Inverted Page Table (IPT)
The Memory Management Unit (MMU)

• **Input**
  – virtual address

• **Output**
  – physical address
  – access violation (exception, interrupts the processor)

• **Access Violations**
  – not present
  – user v.s. kernel
  – write
  – read
  – execute

Translation Lookaside Buffers (TLB)

• **Need to perform address translation on every memory reference**
  – 30% of instructions are memory references
  – 4-way superscalar processor
  – at least one memory reference per cycle

• **Make Common Case Fast, others correct**
• **Throw HW at the problem**
• **Cache PTEs**
Fast Translation: Translation Buffer

- Cache of translated addresses
- Alpha 21164 TLB: 48 entry fully associative

TLB Design

- Must be fast, not increase critical path
- Must achieve high hit ratio
- Generally small highly associative
- Mapping change
  - page removed from physical memory
  - processor must invalidate the TLB entry
- PTE is per process entity
  - Multiple processes with same virtual addresses
  - Context Switches?
- Flush TLB
- Add ASID (PID)
  - part of processor state, must be set on context switch
Managing a Cache/VM

- How do we detect a miss/page fault?
- What happens on a miss/page fault?
- Processor caches are a cache over main memory
  - Hardware managed
- Virtual memory can be a cache over the file system + anonymous memory (e.g., stack, heap)
  - Software managed
- How do we manage the TLB?
- What happens on a TLB miss? How does this relate to a page fault?

Hardware Managed TLBs

- Hardware Handles TLB miss
- Dictates page table organization
- Complicated state machine to “walk page table”
  - Multiple levels for forward mapped
  - Linked list for inverted
- Exception only if access violation
Software Managed TLBs

- Software Handles TLB miss
- Flexible page table organization
- Simple Hardware to detect Hit or Miss
- Exception if TLB miss or access violation
- Should you check for access violation on TLB miss?

Mapping the Kernel

- Digital Unix Kseg
  - kseg (bit 63 = 1, 62 = 0)
- Kernel has direct access to physical memory
- One VA->PA mapping for entire Kernel
- Lock (pin) TLB entry
  - or special HW detection
Considerations for Address Translation

Page Size

**Large virtual address space**
- Can map more things
  - Files, frame buffers, network interfaces, memory from another workstation
- Sparse use of address space
- Page Table Design
  - space
  - less locality => TLB misses

OS structure
- microkernel => more TLB misses

A Case for Large Pages

- Page table size is inversely proportional to the page size
  - memory saved
- Transferring larger pages to or from secondary storage, possibly over a network, is more efficient
- Number of TLB entries are restricted by clock cycle time,
  - larger page size maps more memory
  - reduces TLB misses
- Fast cache hit time easy when cache <= page size (VA caches)
  - bigger page makes it feasible as cache size grows
  - More on this later today...
A Case for Small Pages

- **Fragmentation**
  - large pages can waste storage
  - data must be contiguous within page
- **Quicker process start for small processes(??)**

Superpages

- **Hybrid solution: multiple page sizes**
  - 8KB, 16KB, 32KB, 64KB pages
  - 4KB, 64KB, 256KB, 1MB, 4MB, 16MB pages
- **Need to identify candidate superpages**
  - Kernel
  - Frame buffers
  - Database buffer pools
- **Application/compiler hints**
- **Detecting superpages**
  - static, at page fault time
  - dynamically create superpages
- **Page Table & TLB modifications**
Address Translation for Large Address Spaces

- **Forward Mapped Page Table**
  - grows with virtual address space
    - worst case 100% overhead not likely
  - TLB miss time: memory reference for each level

- **Inverted Page Table**
  - grows with physical address space
    - independent of virtual address space usage
  - TLB miss time: memory reference to HAT, IPT, list search

---

Hashed Page Table (HP)

- Combine Hash Table and IPT [Huck96]
  - can have more entries than physical page frames
- Must search for virtual address
- Easier to support aliasing than IPT
- Space
  - grows with physical space
- TLB miss
  - one less memory ref than IPT
Clustered Page Table (SUN)

- Combine benefits of HPT and Linear [Talluri95]
- Store one base VPN (TAG) and several PPN values
  - virtual page block number (VPBN)
  - block offset

Reducing TLB Miss Handling Time

- Problem
  - must walk Page Table on TLB miss
  - usually incur cache misses
  - big problem for IPC in microkernels

- Solution
  - build a small second-level cache in SW
  - on TLB miss, first check SW cache
    » use simple shift and mask index to hash table
Review: Address Translation

- Map from virtual address to physical address
- Page Tables, PTE
  - va->pa, attributes
    - forward mapped, inverted, hashed, clustered
- Translation Lookaside Buffer
  - hardware cache of most recent va->pa translation
  - misses handled in hardware or software
- Implications of larger address space
  - page table size
    - possibly more TLB misses
- OS Structure
  - microkernels -> lots of IPC -> more TLB misses

Cache Memory 102

- Block 7 placed in 4 block cache:
  - Fully associative, direct mapped, 2-way set associative
  - S.A. Mapping = Block Number Modulo Number Sets
  - DM = 1-way Set Assoc
- Cache Frame
  - location in cache
- Bit-selection

Cache Indexing

- Tag on each block
  - No need to check index or block offset
- Increasing associativity shrinks index, expands tag

<table>
<thead>
<tr>
<th>TAG</th>
<th>Index</th>
<th>Block offset</th>
</tr>
</thead>
</table>

Fully Associative: No index
Direct-Mapped: Large index

Address Translation and Caches

- Where is the TLB wrt the cache?
- What are the consequences?

- Most of today’s systems have more than 1 cache
  - Digital 21164 has 3 levels
  - 2 levels on chip (8KB-data,8KB-inst,96KB-unified)
  - one level off chip (2-4MB)
- Does the OS need to worry about this?

Definition:
page coloring = careful selection of va->pa mapping
Virtual Caches

- Send virtual address to cache. Called *Virtually Addressed Cache* or just *Virtual Cache* vs. *Physical Cache* or *Real Cache*
- Avoid address translation before accessing cache
  - faster hit time to cache
- Context Switches?
  - Just like the TLB (flush or pid)
  - Cost is time to flush + “compulsory” misses from empty cache
  - *Add process identifier tag* that identifies process as well as address within process: can’t get a hit if wrong process
- I/O must interact with cache
I/O and Virtual Caches

I/O is accomplished with physical addresses
- flush pages from cache
- need pa->va reverse translation
- coherent DMA

Aliases and Virtual Caches

- aliases (sometimes called synonyms); Two different virtual addresses map to same physical address
- But, but... the virtual address is used to index the cache
- Could have data in two different locations in the cache
If index is physical part of address, can start tag access in parallel with translation so that can compare to physical tag.

- Limits cache to page size: what if we want bigger caches and use same trick?
  - Higher associativity
  - Page coloring

Page Coloring for Aliases

- HW that guarantees that every cache frame holds unique physical address
- OS guarantee: lower n bits of virtual & physical page numbers must have same value; if direct-mapped, then aliases map to same cache frame
  - one form of page coloring