Virtual Memory

October 27, 1998

Topics

• Motivation
• Address Translation
• Accelerating with TLBs
• Alpha 21X64 memory system
Levels in Memory Hierarchy

<table>
<thead>
<tr>
<th></th>
<th>Register</th>
<th>Cache</th>
<th>Memory</th>
<th>Disk Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>size:</td>
<td>200 B</td>
<td>32 KB / 4MB</td>
<td>128 MB</td>
<td>20 GB</td>
</tr>
<tr>
<td>speed:</td>
<td>3 ns</td>
<td>6 ns</td>
<td>100 ns</td>
<td>10 ms</td>
</tr>
<tr>
<td>$/Mbyte:</td>
<td>$100/MB</td>
<td>$1.50/MB</td>
<td>$0.06/MB</td>
<td></td>
</tr>
<tr>
<td>block size:</td>
<td>4 B</td>
<td>8 B</td>
<td>4 KB</td>
<td></td>
</tr>
</tbody>
</table>

larger, slower, cheaper
Virtual Memory

Classically

- Main memory acts as a cache for the secondary storage (disk)

Increases Program-Accessible Memory

- address space of each job larger than physical memory
- sum of the memory of many jobs greater than physical memory
Address Spaces

- Virtual and physical address spaces divided into equal-sized blocks
  - “Pages” (both virtual and physical)
- Virtual address space typically larger than physical
- Each process has separate virtual address space

### Virtual addresses (VA)

```
Process 1:
0
VP 1
VP 2
N-1
```

### Physical addresses (PA)

```
Process 2:
0
VP 1
VP 2
N-1
address translation
```

0
PP2
PP7
(Read-only library code)

0
PP10
M-1

- Process 1:
  - VP 1
  - VP 2
- Process 2:
  - VP 1
  - VP 2
Other Motivations

Simplifies memory management
  • Main reason today
  • Can have multiple processes resident in physical memory
  • Their program addresses mapped dynamically
    – Address 0x100 for process P1 doesn’t collide with address 0x100 for process P2
  • Allocate more memory to process as its needs grow

Provides Protection
  • One process can’t interfere with another
    – Since operate in different address spaces
  • Process cannot access privileged information
    – Different sections of address space have different access permissions
Contrast: Macintosh Memory Model

Does not Use Traditional Virtual Memory

All program objects accessed through “handles”
- Indirect reference through pointer table
- Objects stored in shared global address space
Macintosh Memory Management

Allocation / Deallocation

- Similar to free-list management of malloc/free

Compaction

- Can move any object and just update the (unique) pointer in pointer table

"Handles"

P1 Pointer Table

P2 Pointer Table

Shared Address Space

A

B

C

D

E
Macintosh vs. VM-based Mem. Mgmt

Both
- Can allocate, deallocate, and move memory blocks

Macintosh
- Block is variable-sized
  - May be very large or very small
- Requires contiguous allocation
- No protection
  - “Wild write” by one process can corrupt another’s data

VM-Based
- Block is fixed size
  - Single page
  - Can map contiguous range of virtual addresses to disjoint ranges of physical addresses
- Provides protection
  - Between processes
  - So that process cannot corrupt OS information
VM Address Translation

\[ V = \{0, 1, \ldots, N-1\} \] virtual address space \hspace{1cm} N > M
\[ P = \{0, 1, \ldots, M-1\} \] physical address space

MAP: \( V \rightarrow P \cup \{\emptyset\} \) address mapping function

- \( \text{MAP}(a) = a' \) if data at virtual address \( a \) is present at physical address \( a' \) in \( P \)
- \( = \emptyset \) if data at virtual address \( a \) is not present in \( P \)

Diagram:

- Processor
- Addr Trans Mechanism
- \( a \)
- \( a' \)
- Physical address
- Fault handler
- Missing item fault
- OS performs this transfer (only if miss)
VM Address Translation

Parameters

- \( P = 2^p \) = page size (bytes). Typically 1KB–16KB
- \( N = 2^n \) = Virtual address limit
- \( M = 2^m \) = Physical address limit

Notice that the page offset bits don't change as a result of translation.
Page Tables
Address Translation via Page Table

VPN acts as table index

if valid=0 then page not in memory

page table base register

virtual address

virtual page number

page offset

physical page number

valid access

physical page number

page offset

physical address

Address
Page Table Operation

Translation
  • Separate (set of) page table(s) per process
  • VPN forms index into page table

Computing Physical Address
  • Page Table Entry (PTE) provides information about page
    – Valid bit = 1 ==> page in memory.
      » Use physical page number (PPN) to construct address
    – Valid bit = 0 ==> page in secondary memory
      » Page fault
      » Must load into main memory before continuing

Checking Protection
  • Access rights field indicate allowable access
    – E.g., read-only, read-write, execute-only
    – Typically support multiple protection modes (e.g., kernel vs. user)
  • Protection violation fault if don’t have necessary permission
VM design issues

Everything Driven by Enormous Cost of Misses:

- Hundreds of thousands to millions of clocks.
  - vs units or tens of clocks for cache misses.
- Disks are high latency
  - Typically 10 ms access time
- Moderate disk to memory bandwidth
  - 10 MBytes/sec transfer rate

Large Block Sizes:

- Typically 1KB–16 KB
- Amortize high access time
- Reduce miss rate by exploiting spatial locality

Perform Context Switch While Waiting

- Memory filled from disk by direct memory access
- Meanwhile, processor can be executing other processes
Disk / System Interface

Processor Signals Controller
- Read block of length P starting at disk address X and store starting at memory address Y

Read Occurs
- Direct Memory Access
- Under control of I/O controller

I/O Controller Signals Completion
- Interrupt processor
- Can resume suspended process
VM design issues (cont)

Fully Associative Page Placement
- Eliminates conflict misses
- Every miss is a killer, so worth the lower hit time

Use Smart Replacement Algorithms
- Handle misses in software
  - Plenty of time to get job done
  - Vs. caching where time is critical
- Miss penalty is so high anyway, no reason to handle in hardware
- Dmall improvements pay big dividends

Write Back Only
- Disk access too slow to afford write through
 Integrating VM and cache

**Most Caches “Physically Addressed”**
- Accessed by physical addresses
- Allows multiple processes to have blocks in cache at same time
- Allows multiple processes to share pages
- Cache doesn’t need to be concerned with protection issues
  - Access rights checked as part of address translation

**Perform Address Translation Before Cache Lookup**
- But this could involve a memory access itself
- Of course, page table entries can also become cached
Speeding up Translation with a TLB

Translation lookaside buffer (TLB)

- Small, usually fully associative cache
- Maps virtual page numbers to physical page numbers
- Contains complete page table entries for small number of pages

![Diagram of CPU, TLB Lookup, Cache, and Main Memory with data flow]

- CPU sends virtual address (VA) to TLB Lookup.
- TLB Lookup checks for a hit or miss.
- If hit, physical address (PA) is sent to Cache, which also checks for a hit or miss.
- If hit in Cache, data is accessed and sent back to CPU.
- If hit in TLB but miss in Cache, data is accessed from Main Memory and placed in Cache.
- If miss in TLB and miss in Cache, data is accessed from Main Memory and placed in TLB and Cache.
- If miss in both TLB and Cache, data is accessed from Main Memory and placed in TLB and Cache.
Address translation with a TLB

virtual address

virtual page number

process ID

physical page number

valid dirty tag

TLB hit

cache hit

data

physical address

byte offset

index

process ID virtual page number page offset

N–1 p–1 0

TLB

Cache

data

TLB

process ID virtual page number page offset

N–1 p–1 0

TLB hit

cache hit

valid dirty tag

valid tag data

valid tag data

valid dirty tag

valid dirty tag

valid dirty tag
Alpha AXP 21064 TLB

- page size: 8KB
- hit time: 1 clock
- miss penalty: 20 clocks
- TLB size: ITLB 8 PTEs, DTLB 32 PTEs
- placement: Fully associative
TLB-Process Interactions

TLB Translates Virtual Addresses
  • But virtual address space changes each time have context switch

Could Flush TLB
  • Every time perform context switch
  • Refill for new process by series of TLB misses
  • ~100 clock cycles each

Could Include Process ID Tag with TLB Entry
  • Identifies which address space being accessed
  • OK even when sharing physical pages
### Alpha Physical Addresses

<table>
<thead>
<tr>
<th>Model</th>
<th>Bits</th>
<th>Max. Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>21064</td>
<td>34</td>
<td>16GB</td>
</tr>
<tr>
<td>21164</td>
<td>40</td>
<td>1TB</td>
</tr>
<tr>
<td>21264</td>
<td>44</td>
<td>16TB</td>
</tr>
</tbody>
</table>

**Why a 1TB (or More) Address Space?**

- At $1.00 / MB, would cost $1 million for 1 TB
- Would require 131,072 memory chips, each with 256 Megabits
- Current uniprocessor models limited to 2 GB
Massively-Parallel Machines

Example: Cray T3E

- Up to 2048 Alpha 21164 processors
- Up to 2 GB memory / processor
- 8 TB physical address space!

Logical Structure

- Many processors sharing large global address space
- Any processor can reference any physical address
- VM system manages allocation, sharing, and protection among processors

Physical Structure

- Memory distributed over many processor modules
- Messages routed over interconnection network to perform remote reads & writes
Alpha Virtual Addresses

Page Size
• Currently 8KB

Page Tables
• Each table fits in single page
• Page Table Entry 8 bytes
  – 4 bytes: physical page number
  – Other bytes: for valid bit, access information, etc.
• 8K page can have 1024 PTEs

Alpha Virtual Address
• Based on 3-level paging structure

<table>
<thead>
<tr>
<th>level 1</th>
<th>level 2</th>
<th>level 3</th>
<th>page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>13</td>
</tr>
</tbody>
</table>

• Each level indexes into page table
• Allows 43-bit virtual address when have 8KB page size
Alpha Page Table Structure

Tree Structure
- Node degree ≤ 1024
- Depth 3

Nice Features
- No need to enforce contiguous page layout
- Dynamically grow tree as memory needs increase
Mapping an Alpha 21064 Virtual Address

PT size: 1024 PTEs

PTE size: 8 Bytes

21 bits

10 bits

13 bits

Main memory

Physical page-frame number

Page offset

Physical address

L3 page table

L2 page table

L1 page table

Page table entry

Page table entry

Page table entry

Page table base register

seg0/seg1
Selector
000 ... 0 or
111 ... 1

Level3

Level2

Level1

Page offset

Virtual address

class19.ppt
Virtual Address Ranges

<table>
<thead>
<tr>
<th>Binary Address</th>
<th>Segment</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>1...1 11 xxxx...xxx</td>
<td>seg1</td>
<td>Kernel accessible virtual addresses</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Information maintained by OS but not to be accessed by user</td>
</tr>
<tr>
<td>1...1 10 xxxx...xxx</td>
<td>kseg</td>
<td>Kernel accessible physical addresses</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– No address translation performed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Used by OS to indicate physical addresses</td>
</tr>
<tr>
<td>0...0 0x xxxx...xxx</td>
<td>seg0</td>
<td>User accessible virtual addresses</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Only part accessible by user program</td>
</tr>
</tbody>
</table>

Address Patterns

- **Must have high order bits all 0’s or all 1’s**
  - Currently 64–43 = 21 wasted bits in each virtual address
- **Prevents programmers from sticking in extra information**
  - Could lead to problems when want to expand virtual address space in future
Alpha Seg0 Memory Layout

Regions

- **Data**
  - Static space for global variables
    - Allocation determined at compile time
    - Access via $gp$
  - Dynamic space for runtime allocation
    - E.g., using `malloc`

- **Text**
  - Stores machine code for program

- **Stack**
  - Implements runtime stack
  - Access via $sp$

- **Reserved**
  - Used by operating system
    - shared libraries, process info, etc.
**Alpha Seg0 Memory Allocation**

**Address Range**
- User code can access memory locations in range
  \[0x0000000000010000\text{ to } 0x000003FFFFFF\]
- Nearly \(2^{42} \approx 4.3980465 \times 10^{12}\) byte range
- In practice, programs access far fewer

**Dynamic Memory Allocation**
- Virtual memory system only allocates blocks of memory as needed
- As stack reaches lower addresses, add to lower allocation
- As break moves toward higher addresses, add to upper allocation
  - Due to calls to `malloc`, `calloc`, etc.
Minimal Page Table Configuration

User-Accessible Pages

• VP4: Shared Library
  – Read only to prevent undesirable interprocess interactions
  – Near top of Seg0 address space

• VP3: Data
  – Both static & dynamic
  – Grows upward from virtual address 0x140000000

• VP2: Text
  – Read only to prevent corrupting code

• VP1: Stack
  – Grows downward from virtual address 0x120000000
Partitioning Addresses

Address 0x001 2000 0000

- Level 1: 0
- Level 2: 576
- Level 3: 0

Address 0x001 4000 0000

- Level 1: 0
- Level 2: 640
- Level 3: 0

Address 0x3FF 8000 0000

- Level 1: 511
- Level 2: 768
- Level 3: 0
Mapping Minimal Configuration

VP1
- 0000 0001 2000 0000
- 0000 0001 2000 1FFF
- 0000 03FF 8000 0000
- 0000 03FF 8000 1FFF
- 0000 0001 1FFF E000
- 0000 0001 1FFF FFFF

VP2
- 0000 0001 4000 0000
- 0000 0001 4000 1FFF

VP3
- 0000 0001 4000 0000
- 0000 0001 4000 1FFF

VP4
- 0000 0001 8000 0000
- 0000 0001 8000 1FFF

511
- 0

768
- 0

640
- 0

576
- 0

575
- 0

1023
- 0

0

511
- 0

768
- 0

640
- 0

576
- 0

575
- 0

1023
- 0

0

511
- 0

768
- 0

640
- 0

576
- 0

575
- 0

1023
- 0

0
Increasing Heap Allocation

Without More Page Tables
- Could allocate 1023 additional pages
- Would give ~8MB heap space

Adding Page Tables
- Must add new page table with each additional 8MB increment

Maximum Allocation
- Our Alphas limit user to 1GB data segment
- Limit stack to 32MB
Expanding Alpha Address Space

Increase Page Size

- Increasing page size 2X increases virtual address space 16X
  - 1 bit page offset, 1 bit for each level index

<table>
<thead>
<tr>
<th>level 1</th>
<th>level 2</th>
<th>level 3</th>
<th>page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>10+k</td>
<td>10+k</td>
<td>10+k</td>
<td>13+k</td>
</tr>
</tbody>
</table>

Physical Memory Limits

- Cannot be larger than kseg
  VA bits \(-2 \geq PA\) bits
- Cannot be larger than 32 + page offset bits
  - Since PTE only has 32 bits for PPN

Configurations

- Page Size 8K 16K 32K 64K
- VA Size 43 47 51 55
- PA Size 41 45 47 48
Main Theme

Programmer’s View

- Large “flat” address space
  - Can allocate large blocks of contiguous addresses
- Processor “owns” machine
  - Has private address space
  - Unaffected by behavior of other processes

System View

- User virtual address space created by mapping to set of pages
  - Need not be contiguous
  - Allocated dynamically
  - Enforce protection during address translation
- OS manages many processes simultaneously
  - Continually switching among processes
  - Especially when one must wait for resource
    » E.g., disk I/O to handle page fault