Review

- Virtual Memory, Paging really used for Protection, Translation, Some OS optimizations
- Not really routinely paging to disk
- Can think of as another level of memory hierarchy, but not really used like caches today

Agenda

- Review
- Performance Impact of Virtual memory
- Exceptions
- Administrivia
- Virtual Machines (if time permits)
- Summary

Impact of Paging

L1 cache hit = 1 clock cycles, hit 95% of accesses
L2 cache hit = 10 clocks, hit 60% of L1 misses
DRAM = 200 clock cycles (~100 nanoseconds)
Disk = 20,000,000 clocks (~10 milliseconds)
- Average Memory Access Time (no paging) = 
  95%*1 + 5%*60%*10 + 5%*50%*200 = 5.25 clocks
- Average Memory Access Time (paging) = 
  95%*1 + 5%*50%*10 + 5%*50%*Hit\_memory*200 +
  5%*50%*Hit\_memory*20,000,000
- How much slower if Hit\_memory = 99.9%?
- What Hit\_memory leads to 100X slowdown?

Memory Management Today

- Slowdown too great to run much bigger programs than memory
  - Called Thrashing
  - Buy more memory or run program on bigger computer or reduce size of problem
- Paging system today still used for
  - Translation (mapping of virtual address to physical address)
  - Protection (permission to access word in memory)
  - Sharing of memory between independent tasks

Impact of TLBs on Performance

- Each TLB miss to Page Table ~ L1 Cache miss
- Page sizes are 4 KB to 8 KB (4 KB on x86)
- TLB has typically 128 entries
  - Set associative or Fully associative
- TLB Reach: Size of largest virtual address space that can be simultaneously mapped by TLB:
  - 128 * 4 KB = 512 KB = 0.5 MB!
- What can you do to have better performance?
Improving TLB Performance

- Add larger page size that operating system can use in situations when OS knows that object is big and protection of whole object OK
  - X86 has 4 KB pages + 2 MB and 4 MB "superpages"
- Have 2 Levels of TLB just like 2 levels of cache instead of going directly to Page Table on L1 TLB miss

Nehalem Virtual Memory Details

- 48-bit virtual address space, 40-bit physical address space
- Two-level TLB
  - I-TLB (L1) has shared 128 entries 4-way associative for 4K pages, plus 7 dedicated fully-associative entries per SMT thread for large page (2/4MB) entries
  - D-TLB (L1) has 64 entries for 4K pages and 32 entries for 2/4MB pages, both 4-way associative, dynamically shared between SMT threads
  - Unified L2 TLB has 512 entries for 4K pages only, also 4-way associative
- Data TLB Reach (4 KB only): L1: 64*4 KB = 0.25 MB, L2: 512*4 KB = 2 MB (superpages) L1: 32*2-4 MB = 64-128 MB

Using Large Pages from Application?

- Difficulty is communicating from application to operating system that want to use large pages
- Linux: “Huge pages” via a library file system and memory mapping; beyond 61C
  - See http://lwn.net/Articles/375096/
- Max OS X: no support for applications to do this (OS decides if should use or not)

Address Translation & Protection

- Every instruction and data access needs address translation and protection checks
  - A good VM design needs to be fast (~ one cycle) and space efficient

Exceptions and Interrupts

- “Unexpected” events requiring change in flow of control
  - Different ISAs use the terms differently
- Exception
  - Arises within the CPU
    - e.g., undefined opcode, overflow, syscall, ...
- Interrupt
  - From an external I/O controller
  - Dealing with them without sacrificing performance is hard

Handling Exceptions

- In MIPS, exceptions managed by a System Control Coprocessor (CP0)
- Save PC of offending (or interrupted) instruction
  - In MIPS: save in special register called Exception Program Counter (EPC)
- Save indication of the problem
  - In MIPS: saved in special register called Cause register
  - We’ll assume 1-bit
    - 0 for undefined opcode, 1 for overflow
  - Jump to exception handler code at address 8000 0180_hex
### Exception Properties
- Restartable exceptions
  - Pipeline can flush the instruction
  - Handler executes, then returns to the instruction
    - Retracted and executed from scratch
- PC saved in EPC register
  - Identifies causing instruction
  - Actually PC + 4 is saved because of pipelined implementation
    - Handler must adjust PC to get right address

### Handler Actions
- Read Cause register, and transfer to relevant handler
- Determine action required
- If restartable exception
  - Take corrective action
  - Use EPC to return to program
- Otherwise
  - Terminate program
  - Report error using EPC, cause, ...

### Exceptions in a Pipeline
- Another form of control hazard
- Consider overflow on add in EX stage
  - Prevent $1 from being clobbered
  - Complete previous instructions
  - Flush add and subsequent instructions
  - Set Cause and EPC register values
  - Transfer control to handler
- Similar to mispredicted branch
  - Use much of the same hardware

### Exception Example

#### Time (clock cycles)

#### Exception Example

#### Exception Example

#### Exception Example

#### Exception Example
Administrivia: Remaining Lectures

- Wed 11/24: Economics of Cloud Computing
- Mon 11/29:½ Anatomy of Nehalem Microprocessor
  —½ on Cal Computing History in 1981:
  Reduced Instruction Set Computers (RISC)
- Wed 12/1:½ on Top 3 Extra Credit for SGEMM
  —½ on Cal Computing History in 1989:
  Redundant Array of Inexpensive Disks (RAID)
- Friday 12/3:
  Course summary, Cal Heritage, HKN course evaluation

Administrivia: Assignments

- Project 4: Single Cycle Processor in Logicsim
  — Due Part 2 due Saturday 11/27
  — Face-to-Face: 12/2 In Lab (signup on whiteboard)
- Extra Credit: Fastest Project 3 (due 11/29 11:59)
- Final Review: Mon Dec 6, 2-5PM (10 Evans)
- Final: Mon Dec 13 8AM-11AM (220 Hearst Gym)

Administrivia: Extra Credit

- Starting today 11/22, option to use one single instance of a Compute Cluster node (cc1.4xlarge) until 11/29 if you prefer
  — No one else on the server
- We expect that the same optimizations that work well on Amazon will also work well on instructional computers, but no guarantees
- Andrew will run on instructional computers when done to decide who is fastest

Cluster Compute Node vs. 61C Node

- 61C Servers: 2 x Intel E5520 quad-core “Nehalem”
  — Mac OS, Apple-patched GCC 4.2.1
  — 2.26 GHz clock rate
  — 12 GB 1066 MHz DDR3 memory
  — 8 MB L3, 2 HW threads/core, 1 Gbit/s Ethernet
- CCN: 2 x Intel Xeon X5570, quad-core “Nehalem”
  — Linux OS, GCC 4.3.5
  — 2.93 GHz (1.30X 5520)
  — 23 GB, 1333 MHz DDR3 (1.25X 5520) memory
  — 8 MB L3, 2 HW threads/core, 10 Gbit/s Ethernet
  — API name: cc1.4xlarge

Multiple Exceptions

- Pipelining overlaps multiple instructions
  — Could have multiple exceptions at once
  — E.g., Page fault in LW same clock cycle as Overflow of following instruction ADD
- Simple approach: deal with exception from earliest instruction e.g., LW exception serviced 1st
  — Flush subsequent instructions
- Called Precise exceptions
- In complex pipelines
  — Multiple instructions issued per cycle
  — Out-of-order completion
  — Maintaining precise exceptions is difficult!

Imprecise Exceptions

- Just stop pipeline and save state
  — Including exception cause(s)
- Let the software handler work out
  — Which instruction(s) had exceptions
  — Which to complete or flush
  — May require “manual” completion
- Simplifies hardware, but more complex handler software
- Not feasible for complex multiple-issue out-of-order pipelines to always get exact instruction
- All computers today offer precise exceptions(?)
Beyond Virtual Memory

- Desire for even greater protection than virtual memory
  - E.g., Amazon Web Services allows independent tasks run on same computer
- Can a "small" operating system simulate the hardware of some machine so that
  - Another operating system can run in that simulated hardware?
  - More than one instance of that operating system run on the same hardware at the same time?
  - More than one different operating system can share the same hardware at the same time?
- Answer: Yes

Solution – Virtual Machine

- A virtual machine provides interface identical to underlying bare hardware
  - i.e., all devices, interrupts, memory, page tables, etc.
- Virtualization has some performance impact
  - Feasible with modern high-performance computers
- Examples
  - IBM VM/370 (1970s technology!)
  - VMWare
  - Xen (used by AWS)
  - Microsoft Virtual PC

Virtual Machines

- Host Operating System:
  - OS actually running on the hardware
  - Together with virtualization layer, it simulates environment for ...
- Guest Operating System:
  - OS running in the simulated environment
- The resources of the physical computer are shared to create the virtual machines
  - Processor scheduling by OS can create the appearance that each user has own processor
  - Disk partitioned to provide virtual disks

Virtual Machine Monitor

- Maps virtual resources to physical resources
  - Memory, I/O devices, CPUs
- Guest code runs on native machine in user mode
  - Traps to VMM on privileged instructions and access to protected resources
- Guest OS may be different from host OS
- VMM handles real I/O devices
  - Emulates generic virtual I/O devices for guest

Example: Timer Virtualization

- In native machine, on timer interrupt
  - OS suspends current process, handles interrupt, selects and resumes next process
- With Virtual Machine Monitor
  - VMM suspends current VM, handles interrupt, selects and resumes next VM
- If a VM requires timer interrupts
  - VMM emulates a virtual timer
  - Emulates interrupt for VM when physical timer interrupt occurs

Virtual Machine Instruction Set Support

- Similar to what need for Virtual Memory
- User and System modes
- Privileged instructions only available in system mode
  - Trap to system if executed in user mode
- All physical resources only accessible using privileged instructions
  - Including page tables, interrupt controls, I/O registers
- Renaissance of virtualization support
  - Current ISAs (e.g., x86) adapting, following IBM’s path
6 Reasons for Virtual Machines at Amazon

- AWS based on Xen VM on x86 servers
- Protect users from each other
- Simplified software distribution within AWS
  - Customers only need install an image and then AWS automatically distributes it to all the instances being used
- Control resource usage
  - Easy to reliably kill a VM

Original AWS Instances, Prices

<table>
<thead>
<tr>
<th>Instance</th>
<th>Per Hour</th>
<th>Ratio to Small</th>
<th>Compute/Unit</th>
<th>Memory (GB)</th>
<th>Disk (GB)</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Small</td>
<td>$0.100</td>
<td>1.0</td>
<td>1</td>
<td>1.00</td>
<td>1.7</td>
<td>1690 GB</td>
</tr>
<tr>
<td>Standard Large</td>
<td>$0.400</td>
<td>4.0</td>
<td>4</td>
<td>2.00</td>
<td>7.5</td>
<td>850 GB</td>
</tr>
<tr>
<td>Standard Extra Large</td>
<td>$0.800</td>
<td>8.0</td>
<td>8</td>
<td>4.00</td>
<td>15.0</td>
<td>1690 GB</td>
</tr>
</tbody>
</table>

- Cheapest was 2 VMs / real core
  - "1 Virtual Core" and 1.7 GB memory
  - "Equivalent CPU capacity of a 1.0-1.2 GHz 2007 AMD Opteron or 2007 Intel Xeon processor" = 1 EC2 "compute unit"
- Most expensive was 1 VM for 4 real cores @ 8X price
  - "4 Virtual Cores" each with 2X performance of Standard Virtual Cores and 15.0 GB memory
- One "Goldilocks" VM in between

November 2010 AWS Instances, Prices

<table>
<thead>
<tr>
<th>Instance</th>
<th>Per Hour</th>
<th>Ratio to Small</th>
<th>Compute/Unit</th>
<th>Memory (GB)</th>
<th>Disk (GB)</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Small</td>
<td>$0.085</td>
<td>1.0</td>
<td>1</td>
<td>1.00</td>
<td>1.7</td>
<td>1690 GB</td>
</tr>
<tr>
<td>Standard Large</td>
<td>$0.240</td>
<td>4.0</td>
<td>4</td>
<td>2.00</td>
<td>7.5</td>
<td>850 GB</td>
</tr>
<tr>
<td>Standard Extra Large</td>
<td>$0.680</td>
<td>8.0</td>
<td>8</td>
<td>4.00</td>
<td>15.0</td>
<td>1690 GB</td>
</tr>
</tbody>
</table>

- Performance ratio/virtual core now 2X, 2.5X, 3.25X, 4.2X
- Virtual Cores/instance now 2X, 4X, 8X

6 Reasons for Virtual Machines at Amazon (cont’d)

4. Gives AWS multiple price points
   - lowest price by packing multiple VMs ("virtual cores") on 1 server
   - highest price is exclusive access to all the machine resources
   - several intermediary points

“And In Conclusion”

- Virtual Memory, Paging really used for Protection, Translation, Some OS optimizations
  - Not really routinely paging to disk
  - Can think of as another level of memory hierarchy, but not really used like caches
- Virtual Machines as even greater level of protection to allow greater level of sharing
  - Enables fine control, allocation, pricing of Cloud Computing
Acknowledgements

- These slides contain material developed and copyright by:
  - Arvind (MIT)
  - Krste Asanovic (MIT/UCB)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)

- MIT material derived from course 6.823
- UCB material derived from course CS252