COSC 6385
Computer Architecture
Virtualizing Compute Resources

Edgar Gabriel
Spring 2010

References

“Computer Architecture - A Quantitative Approach”
Chapter 5.4

“Intel Virtualization Technology: Hardware Support for Efficient
Processor Virtualization”,
Motivation

- Server consolidation
  - What are you doing with hundreds of cores in a single server?
- Sandboxing
  - Have the web server and the database server on the same physical machine but run in different OS
  - If the web server is hacked, the data base server is not necessarily affected by that
- Multiple execution environments
  - Service a runs best on Windows, Service b runs best on Linux

Virtualization

- Goal:
  - Run multiple instances of different OS on the same hardware
  - Present a transparent view of one or more environments (M-to-N mapping of M “real” resources, N “virtual” resources)
- Challenges:
  - Have to split all resources (processor, memory, hard drive, graphics card, networking card etc.) among the different OS -> virtualize the resources
  - The OS can not be aware that it is using virtual resources instead of real resources
Virtualization - concepts

- Two components when using virtualization:
  - Virtual Machine Monitor (VMM)
  - Virtual Machine(s) (VM)
- Para-virtualization:
  - Operating System has been modified in order to run as a VM
- ‘Fully’ Virtualized:
  - No modification required of an OS to run as a VM
Virtual Machine Monitor

- Also called ‘hypervisor’
- Isolates the state of each guest OS from each other
- Protects itself from guest software
- Determines how to map virtual resources to physical resources
  - Access to privileged state
  - Address translation
  - I/O
  - Exceptions and interrupts
- Relatively small code (compared to an OS)
- VMM must run in a higher privilege mode than guest OS

Managing Virtual Memory (I)

- Virtual memory offers many of the features required for hardware virtualization
  - Separates the physical memory onto multiple processes
  - Each process ‘thinks’ it has a linear address space of full size
  - Processor holds a page table translating virtual addresses used by a process and the according physical memory
  - Additional information restricts processes from
    - Reading a page of on another process or
    - Allow reading but not modifying a memory page or
    - Do not allow to interpret data in the memory page as instructions and do not try to execute them
Managing Virtual Memory (II)

- Virtual Memory management thus requires
  - Mechanisms to limit memory access to protected memory
  - At least two modes of execution for instructions
    - Privileged mode: an instruction is allowed to do what it whatever it wants -> kernel mode for OS
    - Non-privileged mode: user-level processes
  - Intel x86 Architecture: processor supports four levels
    - Level 0 used by OS
    - Level 3 used by regular applications
  - Provide mechanisms to go from non-privileged mode to privileged mode -> system call

Managing Virtual Memory (III)

- Provide a portion of processor state that a user process can read but not modify
  - E.g. memory protection information
Managing Virtual Memory (IV)

- Each guest OS maintains its page tables to do the mapping from virtual address to physical address.
- Most simple solution: VMM holds an additional table which maps the physical address of a guest OS onto the ‘machine address’
  - Introduces a third level of redirection for every memory access.
- Alternative solution: VMM maintains a shadow page table of each guest OS
  - Copy of the page table of the OS
  - Page tables still works with regular physical addresses
  - Only modifications to the page table are intercepted by the VMM.

Privilege Level on Intel Processors

- A protection ring is one of two or more hierarchical levels or layers of privilege within the architecture of a computer system.
- Ring 0 is the level with the most privileges and interacts most directly with the physical hardware such as the CPU and memory.
- Windows and Linux both use only levels 0 and level 3.

Image source:
ISA related problems when running a guest OS

- **Ring Aliasing**: arises when software runs at a privilege level other than which it has been written for
  - E.g. OS wants to modify page table
- **Address Space Compression**: OS expects to have access to the processors full virtual address
  - VMM needs a portion of the virtual address space
- **Non-faulting access to privileged State**
  - E.g. read-access to certain state registers
- **Adverse Impact on Guest System Calls**
  - A system call on a guest OS should not change to the privilege level 0, but to the level of the guest OS

ISA related problems when running a guest OS (II)

- **Interrupt Virtualization**
  - IA32 architecture provides the ability to mask external interrupts if the OS is not ready to handle it
  - OS frequently uses this features
  - VMM will have to handle external interrupts (e.g. hard drive, network), since it has to determine which guest OS needs to be ‘informed’ about that
- **Access to Hidden State**
  - E.g. IA32 has hidden descriptor caches for segment registers
  - No mechanism available saving a restoring hidden components of a guest context when changing VM
Intel Virtualization Architecture for IA32

• Two new forms of CPU operations:
  - **VMX root** operations: intended for use by a VMM
  - **VMX non-root** operations: intended for use by a VM
  - Orthogonal concept to the privileges, i.e. both forms support all four privilege levels
  - **VM entry**: operation to enter VMX non-root mode
  - **VM exit**: Operation to enter VMX root mode

• Instructions and events can cause VM exits

• **Virtual Machine Control Structure (VMCS):**
  - manages VM entries and VM exits.
  - Contains a *guest-state area* and a *host-state area*.
  - **VM entry** loads processor state from the *guest-state area*
  - **VM exit** saves processor state to the *guest-state area* and loads processor state from the *host-state area*.
Support for virtualized execution in the Intel Nehalem architecture (I)

- Nehalem’s TLB entries have also changed subtly by introducing a “Virtual Processor ID” or VPID. Every TLB entry caches a virtual to physical address translation for a page in memory, and that translation is specific to a given process and virtual machine.
- Intel’s older CPUs would flush the TLBs whenever the processor switched between the virtualized guest and the host instance, to ensure that processes only accessed memory they were allowed to touch.
- The VPID tracks which VM a given translation entry in the TLB is associated with, so that when a VM exit and re-entry occurs, the TLBs do not have to be flushed for safety.

Support for virtualized execution in the Intel Nehalem architecture (II)

- Extended Page Tables (EPT): eliminate many VM transitions (rather than just lowering the latency as the VPID does).
  - The normal page tables map guest virtual addresses to guest physical addresses;
  - for a virtualized system, there is also a translation from guest physical to host physical addresses.
  - The EPT manages those mappings from guest physical to host physical. When a page fault happens on the guest physical to host physical mapping, Nehalem will simply walk the EPTs, whereas earlier Intel designs (and AMD designs before Barcelona) would need the hypervisor to service the page fault. This eliminates a lot of unnecessary VM exits.
I/O

- All VMs have to share a single networking card, hard drive, graphics card etc.
- I/O operations have to be redirected to the VMM or a dedicated guest OS which has access to
  - Device drives used by guest OS is typically just forwarding the request to the VMM
- Typical steps
  - VM exit due to an I/O access
  - Decode the requested I/O instruction
  - Make an I/O request package describing the event
  - Send the event to the real device
  - Wait for reponse from the I/O port
  - VMRESUME back to the guest OS

Network measurements: Latency & Bandwidth

Rakhi/Kshitij
NVM Latency : 101us
VM Latency : 161us

Rakhi/Vish
NVM Latency : 90us
VM Latency : 122us
Sequential ATF

Parallel ATF with runtime optimization (2 Proc case)

<table>
<thead>
<tr>
<th>Problem size</th>
<th>ADCL Winner</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 X 32 X 32</td>
<td>sendrecv_pair</td>
</tr>
<tr>
<td>128 X 32 X 32</td>
<td>sendrecv_pair</td>
</tr>
<tr>
<td>128 X 64 X 32</td>
<td>sendrecv_pair</td>
</tr>
<tr>
<td>64 X 32 X 32</td>
<td>sendrecv_aao_pack</td>
</tr>
<tr>
<td>128 X 32 X 32</td>
<td>sendrecv_pair_pack</td>
</tr>
<tr>
<td>128 X 64 X 32</td>
<td>sendrecv_pair_pack</td>
</tr>
</tbody>
</table>
Parallel ATF with runtime optimization (4 Proc case)

NPB: FT

- A different communication pattern: MPI_Bcast and MPI_Alltoall

<table>
<thead>
<tr>
<th>Platform</th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVM2</td>
<td>44.11</td>
</tr>
<tr>
<td>VM2</td>
<td>53.69</td>
</tr>
<tr>
<td>NVM4</td>
<td>32.24</td>
</tr>
<tr>
<td>VM4</td>
<td>39.48</td>
</tr>
</tbody>
</table>

- More than 20% of performance penalty on the 2 proc test case.
Recap for the 3rd quiz

- Content:
  - chapter F (Vector processors)
  - chapter 3.1 - 3.5 (limitations of ILP, multi-threading)
  - chapter 4.1 - 4.5 (multi-processors, cache coherence and synchronization)
  - chapter 5.4 (Virtualization)
  - multi-core lecture
  - Larrabee paper (reference [1] from lecture 19 chapter 1-3)
  - (PAPI) homework
- Please be sure to be also familiar with the cache organization aspects (lecture 4 and 5)

Recap for the 3rd quiz

- You are allowed to have 9 sheets of handwritten notes
  - 3 pages from the 1st quiz
  - 3 pages from the 2nd quiz
  - 3 new pages with the new material
- Possible calculations in the 3rd quiz:
  - Parallel Speedup and Efficiency
  - N^{1/2} calculations from Vector processors lecture