I/O Virtualization using Intel VT-d

I/O Virtualization

- Simple definition
  - Enabling VMs to perform I/O operations
- Key requirements
  - Protected access to I/O resources
  - Ability to share I/O resources

Differing requirements

- Server consolidation in Data Centers
  - Scalability
  - Performance
  - Reliability
  - Availability
- Client platforms
  - Cost and trust
  - Virtual-appliance Model: management, security, content protection
    - Applications: Intrusion monitoring (enterprise), DRM (home computers)

Three VMM architectures

- OS Hosted VMMs
  - KVM
  - Reuse Host drivers and other code
  - Only as secure as host OS
- Standalone Hypervisors
  - VMWare ESX Server
  - Smaller code path from Guest OS to I/O device
  - Limited portability
- Hybrid VMMs
  - Xen
  - Small Hypervisor with its own CPU scheduler
  - Service OS (Domain 0) to handle I/O
  - Hopefully service OS is more secure than general purpose host-OS
  - Too many privilege transitions between guest OS, VMM, and Service OS

Virtualizing I/O

- Emulation
- Para-virtualized Devices
- Direct Assignment

IOVM Architecture

- Dedicated VM that manages and interacts with a specific I/O device on behalf of rest of the system.
- I/O device directly assigned to IOVM
- IOVM can then perform either emulation of para-virtualization of the device for other “normal” VMs.
- Improves isolation of the system from driver crashes.
  - Simply restart the IOVM if driver misbehaves

VT-d support for direct assignment

Key problem

- Isolation
- How to prevent one VM from issuing DMA request to the memory of another VM?
- How to prevent an I/O device from issuing DMA to an unrelated VM?
- How to prevent interrupts from an I/O device from interfering with operations of an unrelated VM?

Solution 1: DMA remapping

- System software (VMM) can create multiple *protection domains.*
- *Protection Domain* is an isolated environment to which a subset of the host physical memory is allocated.
  - For example, memory allocated to a VM or to an IOVM
- Address specified in DMA request from an I/O device is treated as a DMA virtual address (DVA)
  - For example, DVA could be the guest physical address (GPA), also called pseudo-physical address.
- DVA is then translated to Host Physical Address (HPA) by the DMA remapping hardware.

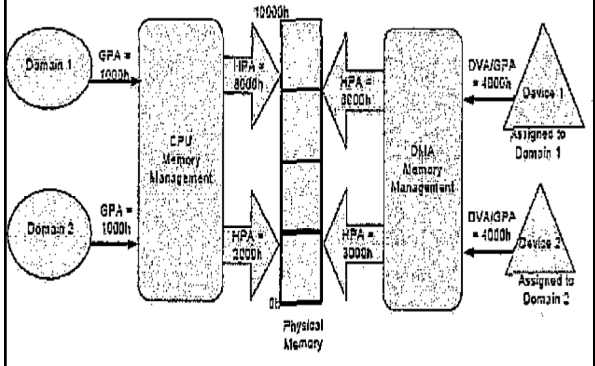


Figure 5: DMA remapping

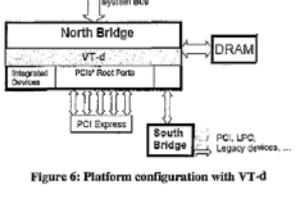
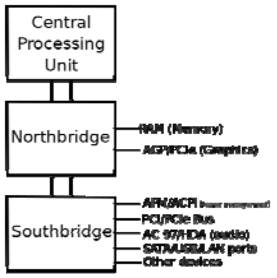


Figure 6: Platform configuration with VT-d



Mapping devices to protection domains

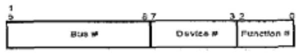


Figure 7: PCI requester identifier format

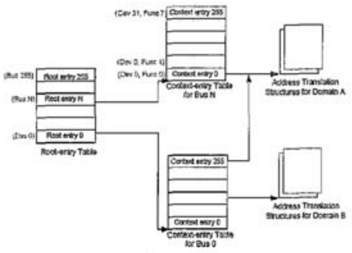


Figure 8: Device mapping structures

Translating from DVA to HPA

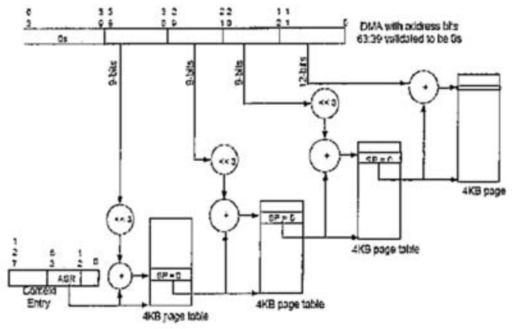


Figure 9: Example 3-level page table

Observations

- Page tables always in main memory
- But where are the root-entry table and context-entry tables located?
  - In the Northbridge?
  - Or in main memory?
- VMM programs the DMA (and interrupt) remapping structures to perform DVA/GPA to HPA mappings.

Traditional interrupt architecture

- Legacy interrupts : routed through I/O interrupt controllers
- MSI: Message Signaled interrupts:
  - Issued as DMA write transactions to pre-defined architectural addresses
  - Interrupt attributes, including destination processor, are encoded in the address and data of the DMA write request.
- This means target processor is specified by the device.

Interrupt remapping in VT-d

- Redefine the interrupt message format
- Still a DMA write request, as with MSI.
- But the write request is only a message identifier, not the actual interrupt attributes.
- The DMA write requests that correspond to interrupts are remapped through *interrupt-remapping table.*
- Each entry in the interrupt remapping table contains interrupt attributes, including the destination processor ID.
- This table can be programmed from the system software (VMM).
- So, if the guest VCPU responsible for target protection domain moves from one physical CPU (PCPU) to another, then the VMM can reprogram the interrupt remapping table to reflect the new destination processor.
- This eliminates the likelihood that incorrect PCPU might be interrupted.

Hardware caching and invalidation

- Cache frequently used remapping structures and entries
  - Context cache
    - indicates context entry table is in main memory
  - Page directory entry cache
  - IOTLB
  - Interrupt entry cache
- Software responsible for cache consistency by invalidating stale entries
  - Synchronous versus queued (batched) invalidation
- Device-IOTLB
  - Problem: IOTLB performance may be worse than CPU TLBs because of multiple concurrent DMA operations
  - Solution: Allow devices to individually cache their own translations in a Device-IOTLB.
  - Device queries the DMA remapping hardware and caches important mapping.
  - DMA request could specify either DVA or HPA.
  - Issues: Security – which devices can be trusted to not use malicious HPAs?

Future VT-d support

- Allow devices to be shared securely among multiple protection domains
- Requires fundamental changes in how devices represent themselves to rest of the system
- Implies multiple virtual functional units exposed by device, each of which can be discovered, configured, and managed.