PCI Device Passthrough for KVM

Amit Shah, Qumranet

Allen M. Kay, Intel

Muli Ben-Yehuda, IBM

Ben-Ami Yassour, IBM

Motivation

- Performance
- Functionality is to be provided to the guest
  - Graphics cards
  - Dongle
  - TV adapters
  - Ethernet adapters
  - Audio cards
- Supporting odd-ball devices that don't have emulation support or equivalent PV drivers

Challenges

- PCI Config
- MMIO
- PIO
- Interrupts
- Address Translation (DMA)
- Security
- Hardware ROMs

PCI Config

- Virtualizing the PCI BAR
  - Guest shouldn't change the BAR address on the host
- Virtualizing registers

Passthrough PIO & MMIO

- Trapped
  - Trapped by the host
  - Emulated by userspace

Passthrough PIO & MMIO

- Direct
  - MMIO
    - Host traps guest changes to the mmio BAR
    - Host maps mmio BAR in KVM userspace via sysfs
    - Host creates a new memory slot for the mmio BAR of the passthrough device
    - When the guest accesses mmio region, pagefault is resolved according to the new mmio memory slot
  - PIO
    - Use VMCS / VMCB IO bitmaps to allow the guest to perform PIO directly without causing a trap

Passthrough Interrupts

- Host registers a passthrough interrupt handler for IRQ on behalf of the guest
- Interrupt recieved by the host is injected into the guest
- Guest acks virtual APIC
- Guests can dynamically change IRQs of a device
  - Handle with trapping writes to PCI config space

Passthrough Interrupts

- Sharing
  - Current support only for non-shared IRQ devices
  - Main concern: guest interrupt ack will be slow
  - Host keeps getting a lot of interrupts till we ack
  - Too much overhead on the host to allow this configuration
  - However, host Linux already supports MSI
    - Hence, not too big an issue.

Passthrough Interrupts: Userspace

- Alternate method for interrupt injection
  - Requested by some users
  - Mainly to support certain non-x86 systems
- Userspace IRQ delivery (-no-kvm-irqchip): irqhook
- Initially written for testing passthrough
- Also usable for -no-kvm case

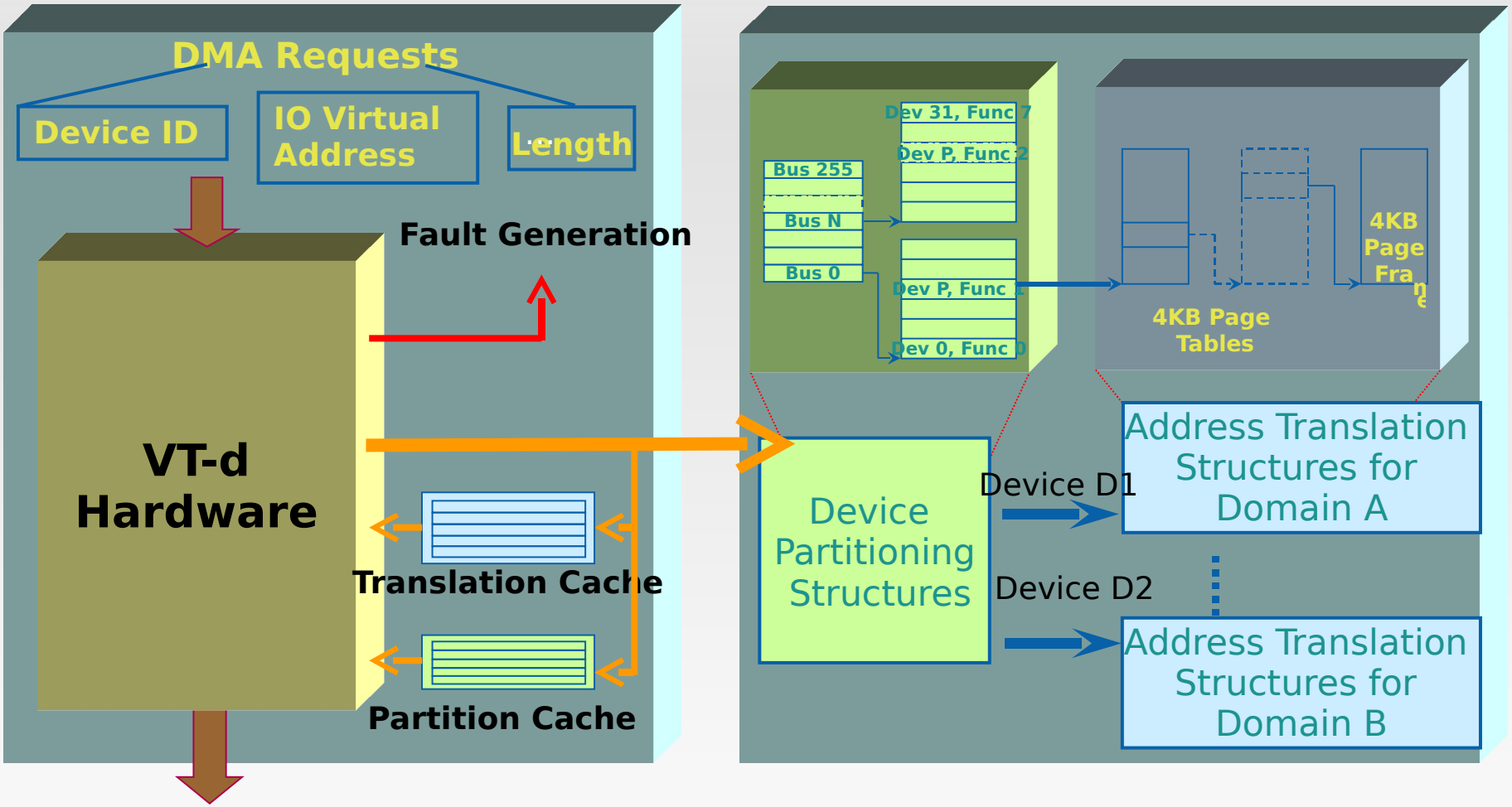
Approaches to Address Translation

- IOMMU-based
  - Intel VT-d
  - AMD IOMMU
- Software
  - Paravirtualized guests (PVDMA)
  - 1-1 mapping of guests in host address space
- Device-level
  - Devices support multiple guests and domains
- Combined
  - PVDMA with IOMMU

Address Translation: Intel's VT-d

- Provides HW translation from GPA to HPA
- Translation granularity is per PCI device
- Root entry table is indexed by PCI bus number
- Context entry table is indexed by PCI devfn
- Each context entry points to page directory table
- VT-d page table can be 3 or 4 levels
- HW caches translation in context entry cache and IOTLB

VT-d Hardware Overview



Memory Access with Host Physical Address

Memory-resident IO Partitioning & Translation Structures

Address Translation: Intel's VT-d

- Translation flow:
  - Guest driver programs IOVA to DMA register in the device
    - Direct map: IOVA = GPA
  - Device moves data to/from memory pointed to by IOVA
  - VT-d HW translates IOVA to HPA
  - Data is correctly accessed at correct physical memory pointed to by IOVA

KVM and VT-d: Direct Mapping

- Leveraging existing Linux VT-d IOMMU driver
  - Enable Linux VT-d support in the kernel
- When a PCI device is assigned
  - Calls intel_iommu_page_mapping() to build VT-d page table to map the entire guest memory
  - Top level page directory page is then programmed into the context entry corresponding to the PCI bus:dev.func
  - Guest pages are locked into memory so swapping is not allowed
- Page additions to the guest are reflected in the VT-d page table
- Each guest has one VT-d page table
  - Multiple assigned devices share the same table

Address Translation: PVDMA

- Purpose is to provide HPA to the guest driver to program the device
- Modify DMA API to translate GPA to HPA
  - Via hypercalls
- Hacky stacking of dma_ops
- Enhancements
  - Immediate: per-device dma_ops (in -mm tree)
  - Future: stackable dma_ops

Address Translation: PVDMA

- Map:
  - [G] Call original handler, get GPA of DMA location
  - [G] Make a hypercall just before writing to device
  - [H] Get the GPA->HPA mapping, pin the page(s), form a sg list and return the pointer to the HPA
  - [G] Return to original handler
- Unmap:
  - [G] Make a hypercall
  - [H] Unpin page(s), delete sg list
  - [G] Call original handler

Address Translation: PVDMA

- Stackable dma_ops needed: swiotlb may be needed within the guest
- dma_alloc_coherent:
  - If DMA area doesn't reside in high memory or force_iommu != 1, no dma_ops->alloc_coherent()

(No)Address Translation: 1:1 Mapping of Guest

- Reserve low memory for guest usage
  - GPA = HPA
- Map the entire guest – no swapping or ballooning
- Can't have more than one guest using this
- Host won't see the RAM allocated to guest
- Can have device assignment without KVM support

Address Translation: Device Support

- Devices themselves can support multiple “channels”
  - Share single device between multiple guests
  - Each device instance has own register window, etc.
- Needs guest and host drivers to program the device
- Lesser risk of device trampling on other domains
- PCI-SIG IOV SR-IOV and MR-IOV specs recently finalized
- Prototype hardware starting to become available

Comparing Passthrough Methods

	Inter-guest protection	Intra-guest protection	Memory Pinned	IOMMU remapping overhead	Unmodified guest
1:1 mapping	no	no	entire guest memory	n/a	yes – single guest only
PVDMA	no	no	active DMA buffers	n/a	no
Direct map IOMMU	yes	no	entire guest memory	none	yes
PVDMA with IOMMU	yes	yes	active DMA buffers	per dma op	no

Challenges with Address Translation

- Protection
- Need to pin all of the guest into RAM
  - Prevents memory overcommit
- Performance
  - Minimize frequency and cost of IOMMU remappings
    - IOTLB flushes
  - Current general virtualization overheads
    - All: Interrupt injection
    - PVDMA: Hypercalls

Road Ahead

- Get all this merged
- Test more devices
- Remove host device IRQ number on command line; autodetect
- Fail guest startup when a module is already loaded for the device we're assigning
  - pci_enable_device()
  - pci_request_regions()
- Direct MMIO and PIO without incurring VM exits
- Have dma_alloc_coherent() call dma_ops->alloc_coherent(); pave way for PVDMA upstream
- PVDMA with IOMMU