VGPU ON KVM
VFIO BASED MEDIATED DEVICE FRAMEWORK
Neo Jia & Kirti Wankhede, 08/25/2016
AGENDA

Background / Motivation
Mediated Device Framework - Overview
Mediated Device Framework - Deep-Dive
Current Status
Demo
Future work
TODAY, HOW GPU PRESENTED INSIDE KVM VM

VFIO device pass-through [1]

Great performance

Full API compatibility - GPU vendor driver inside the virtual machine

Poor density - limited by PCI-E resource

Minimal visibility of the device on the host - generic vfio_pci owns this device, and only perform enable/disable/route interrupts, reset the device

Difficult to cover all graphics workload - either underutilized or too small to scale
WHAT IS VGPU?

High level overview

- Physical GPU shared among multiple virtual machines
- Great performance and suitable for different workload
- Full API compatibility - GPU vendor driver inside the virtual machine
- Full device visibility to the hypervisor/host - allows for device-specific features such as dynamically monitoring and tuning performance, detailed error reporting, etc.
I/O VIRTUALIZATION
SR-IOV and mediated solutions

SR-IOV devices - supported by standard VFIO PCI (Direct Assignment) today

- Established QEMU VFIO/PCI driver, KVM agnostic and well-defined UAPI
- Virtualized PCI config /MMIO space access, interrupt delivery
- Modular IOMMU, pin and map memory for DMA

Mediated devices - non SR-IOV, require vendor-specific drivers to mediate sharing

- Leveraging existing VFIO framework, UAPI
- Vendor driver - Mediated Device - managing device’s internal I/O resource
MEDIATED DEVICE FRAMEWORK
A common framework for mediated I/O devices

Mediated core module (new)

Mediated bus driver, create mediated device

Physical device interface for vendor driver callbacks

Generic mediate device management user interface (sysfs)

Mediated device module (new)

Manage created mediated device, fully compatible with VFIO user API

VFIO IOMMU driver (enhancement)

VFIO IOMMU API TYPE1 compatible, easy to extend to non-TYPE1
MEDIATED DEVICE FRAMEWORK
MEDIATED DEVICE FRAMEWORK - INITIALIZATION
MEDIATED DEVICE FRAMEWORK

With QEMU - Driver Initialization

Registers VFIO MDEV as driver
MEDIATED DEVICE FRAMEWORK

With QEMU - Driver Initialization

Registers VFIO MDEV as driver

Vendor driver registers devices
MEDIATED DEVICE FRAMEWORK

With QEMU - Driver Initialization

Registers VFIO MDEV as driver

Vendor driver registers devices

Vendor driver registers Mediated CBs
MEDIATED DEVICE FRAMEWORK

Mediated Device sysfs

After vendor driver device registration, under physical device sysfs:

- `mdev_create`: create a virtual device (aka mdev device)
- `mdev_destroy`: destroy a mdev device
- `mdev_supported_types`: supported mdev and configuration of this device

Mdev node: /sys/bus/mdev/devices/$mdev_UUID/

online: start and stop virtual device
MEDIATED DEVICE FRAMEWORK

With QEMU - Device Initialization

1. Registers VFIO MDEV as driver
2. Vendor driver registers devices
3. Vendor driver registers Mediated CBs
4. User writes mdev sysfs to create mdev device

- Guest OS
- Guest RAM
- QEMU
- VM
- Mdev SysFS
- VFIO UAPI
- TYPE1 IOMMU UAPI
- Mediated Core
- Device Register Interface
- Mdev Driver Register Interface
- Mediated Bus Driver
- Mediated CBs
- VFIO MDEV
- VFIO
- TYPE1 IOMMU
- PIN/UNPIN
- Vendor driver
- PCIE Device
- MMU
- RAM
- IOMMU
**MEDIATED DEVICE FRAMEWORK**

With QEMU - Device Initialization

- Registers VFIO MDEV as driver
- Vendor driver registers devices
- Vendor driver registers Mediated CBs
- User writes mdev sysfs to create mdev device
- QEMU calls VFIO API to add VFIO dev to IOMMU container, group, get fd back
MEDIATED DEVICE FRAMEWORK

With QEMU - Device Initialization

Registers VFIO MDEV as driver
Vendor driver registers devices
Vendor driver registers Mediated CBs
User writes mdev sysfs to create mdev device
QEMU calls VFIO API to add VFIO dev to IOMMU container, group, get fd back
QEMU access device fd and present it into VM
MEDIATED DEVICE ACCESS - EMULATED
MEDIATED DEVICE ACCESS

Emulated vs Passthrough

Virtual device memory region are presented inside guest for consistent view of vendor driver.

Access to emulated regions are redirected to mediated vendor driver for virtualization support.

Access to passthrough region are directly sent to device corresponding region for max performance.

1st access redirected to mediated vendor driver for CPU page table setup.
MEDIATED DEVICE ACCESS

Emulated

QEMU gets region info via VFIO UAPI from vendor driver thru VFIO MDEV and Mediated CBs
MEDIATED DEVICE ACCESS

QEMU gets region info via VFIO UAPI from vendor driver thru VFIO MDEV and Mediated CBs

Vendor driver accesses MDEV MMIO trapped region backed by mdev fd triggers EPT violation
QEMU gets region info via VFIO UAPI from vendor driver thru VFIO MDEV and Mediated CBs

Vendor driver accesses MDEV MMIO trapped region backed by mdev fd triggers EPT violation

KVM services EPT violation and forwards to QEMU VFIO PCI driver
QEMU gets region info via VFIO UAPI from vendor driver thru VFIO MDEV and Mediated CBs

Vendor driver accesses MDEV MMIO trapped region backed by mdev fd triggers EPT violation

KVM services EPT violation and forwards to QEMU VFIO PCI driver

QEMU convert request from KVM to R/W access to MDEV fd
MEDIATED DEVICE ACCESS

Emulated

QEMU gets region info via VFIO UAPI from vendor driver thru VFIO MDEV and Mediated CBs

Vendor driver accesses MDEV MMIO trapped region backed by mdev fd triggers EPT violation

KVM services EPT violation and forwards to QEMU VFIO PCI driver

QEMU convert request from KVM to R/W access to MDEV fd

RW handled by vendor driver via Mediated CBs and VFIO MDEV
MEDIATED DMA TRANSLATION
MEDIATED DMA TRANSLATION
With QEMU - Memory Tracking

QEMU Starts
MEDIATED DMA TRANSLATION

With QEMU - Memory Tracking

QEMU Starts

Memory regions gets added by QEMU
MEDIATED DMA TRANSLATION

With QEMU - Memory Tracking

QEMU Starts

Memory regions gets added by QEMU

QEMU calls VFIO_DMA_MAP via Memory listener
**MEDIATED DMA TRANSLATION**

**With QEMU - Memory Tracking**

1. **QEMU Starts**
2. Memory regions get added by QEMU
3. QEMU calls VFIO_DMA_MAP via Memory listener
4. TYPE1 IOMMU tracks <VA, GFN>
MEDIATED DMA TRANSLATION

With QEMU - Runtime Memory pinning

QEMU Starts

Memory regions get added by QEMU

QEMU calls VFIO_DMA_MAP via Memory listener

TYPE1 IOMMU tracks <VA, GFN>

Vendor driver pin/translate GFN by TYPE1 IOMMU to get PFN
MEDIATED DMA TRANSLATION
With QEMU - Runtime Memory pinning

QEMU Starts
Memory regions get added by QEMU
QEMU calls VFIO_DMA_MAP via Memory listener
TYPE1 IOMMU tracks <VA, GFN>
Vendor driver pin/translate GFN by TYPE1 IOMMU to get PFN
Vendor driver call pci_map_sg to map PFNs to BFN, program DMA
MEDIATED DEVICE - INTERRUPT
MEDIATED DEVICE FRAMEWORK

With QEMU - Interrupt Setup

QEMU query MDEV supported interrupt type, provided by vendor driver
MEDIATED DEVICE FRAMEWORK

With QEMU - Interrupt Setup

QEMU query MDEV supported interrupt type, provided by vendor driver

QEMU setups up KVM IRQFD
MEDIATED DEVICE FRAMEWORK

With QEMU - Interrupt Setup

QEMU query MDEV supported interrupt type, provided by vendor driver

QEMU setups up KVM IRQFD

QEMU notifies the vendor driver IRQFD via VFIO PCI UAPI
MEDIATED DEVICE FRAMEWORK

With QEMU - Interrupt injection in runtime

- QEMU query MDEV supported interrupt type, provided by vendor driver
- QEMU setups up KVM IRQFD
- QEMU notifies the vendor driver IRQFD via VFIO PCI UAPI

Vendor driver inject interrupt by signaling on eventfd, trigger guest ISR
MEDIATED DEVICE - CURRENT STATUS
CURRENT STATUS

Upstream

[PATCH v7] is sent out by Kirti Wankhede on 08/24/2016

vfio: Mediated device Core driver
vfio: VFIO driver for mediated devices
vfio iommu: Add support for mediated devices
docs: Add Documentation for Mediate devices

Tested with Linux kernel 4.7

Multiple mediated device per VM
Multiple VFIO passthru device per VM
Mixed mediated device and VFIO passthru device
DEMO: NVIDIA VGPU
MEDIATED DEVICE FRAMEWORK - FUTURE WORK
MEDIATED DEVICE FRAMEWORK

Future work

VM

Guest OS

Guest RAM

PCIE MDEV

QEMU

Mdev SysFS

VFIO UAPI

TYPE1 IOMMU UAPI

Mediated Core

Device Register Interface

Mdev Driver Register Interface

Mediated Bus Driver

Mediated CBs

Vendor driver

VFIO MDEV

VFIO

KVM

TYPE1 IOMMU

PIN/UNPIN

MMU

RAM

IOMMU

PCIE Device

GPU

GPU

PCIE Device

VFIO

MDEV

Vendor driver
MEDIATED DEVICE FRAMEWORK

Future work

POWER support - by extend pin/unpin to SPAPR TCE v2 IOMMU
MEDIATED DEVICE FRAMEWORK

Future work

POWER support - by extend pin/unpin to SPAPR TCE v2 IOMMU

Libvirt integration
REFERENCE

[1] An Introduction to PCI Device Assignment with VFIO - Alex Williamson, Red Hat

[Qemu-devel] [PATCH v7 0/4] Add Mediated device support

[libvirt] [RFC] libvirt vGPU QEMU integration
QUESTIONS?