To Trap or Not To Trap: The PCI Passthrough

Dongli Zhang

January 6, 2021
What is PCI passthrough?

- To assign full and direct access of PCI device to VM

The physical devices are exposed to guest VM

Device Emulation & Pavavirtualized Device

PCI Device (PF/VF) Passthrough

The NVMe developers may blame virtualization developers

The virtualization developers may blame hardware/firmware

To assign bugs related to NVMe PCI passthrough:
What is PCI passthrough?

- To assign full and direct access of PCI device to VM
- How to assign bugs related to NVMe PCI passthrough?
What is PCI passthrough?

- To assign full and direct access of PCI device to VM
- How to assign bugs related to NVMe PCI passthrough?
  - The NVMe developers may blame virtualization developers

Device Emulation & Pavavirtualized Device

PCI Device (PF/VF) Passthrough

- The physical devices are exposed to guest VM

Dongli Zhang

To Trap or Not To Trap: The PCI Passthrough

January 6, 2021 2 / 17
What is PCI passthrough?

- To assign full and direct access of PCI device to VM
- How to assign bugs related to NVMe PCI passthrough?
  - The NVMe developers may blame virtualization developers
  - The virtualization developers may blame hardware/firmware

![Diagram of PCI passthrough]

Device Emulation & Pavavirtualized Device

PCI Device (PF/VF) Passthrough

The physical devices are exposed to guest VM
**STEP 1**: To unbind 01:00.0 from NVMe driver and register to vfio-pci driver:

01:00.0 Non-Volatile memory controller: Intel Corporation Device f1a6 (rev 03)

host# echo 0000:01:00.0 > /sys/bus/pci/devices/0000\:01\:00.0/driver/unbind
host# lspci -ns 0000:01:00.0
01:00.0 0108: 8086:f1a6 (rev 03)

host# echo "8086 f1a6" > /sys/bus/pci/drivers/vfio-pci/new_id
**STEP 1**: To unbind *01:00.0* from NVMe driver and register to *vfio-pci* driver:

01:00.0 Non-Volatile memory controller: Intel Corporation Device f1a6 (rev 03)

host# echo 0000:01:00.0 > /sys/bus/pci/devices/0000:01:00.0/driver/unbind
host# lspci -ns 0000:01:00.0
01:00.0 0108: 8086:f1a6 (rev 03)
host# echo "8086 f1a6" > /sys/bus/pci/drivers/vfio-pci/new_id

**STEP 2**: To passthrough the VFIO-managed NVMe to QEMU/KVM VM:

```
# qemu-system-x86_64 -machine accel=kvm -vnc :0 -serial stdio -smp 4 -m 4096M
-net nic -net user,hostfwd=tcp::5022-:22
-hda /home/user/img/boot.qcow2
-device vfio-pci,host=0000:01:00.0
```
**STEP 1**: To unbind 02:10.0 from igb driver and register to xen-pciback driver:

02:10.0 Ethernet controller: Intel Corporation I350 Gigabit Network Connection

```bash
host# echo 0000:02:10.0 > /sys/bus/pci/devices/0000:02:10.0/driver/unbind
host# echo 0000:02:10.0 > /sys/bus/pci/drivers/pciback/new_slot
host# echo 0000:02:10.0 > /sys/bus/pci/drivers/pciback/bind
```
**STEP 1**: To unbind 02:10.0 from igb driver and register to xen-pciback driver:

02:10.0 Ethernet controller: Intel Corporation I350 Gigabit Network Connection

host# echo 0000:02:10.0 > /sys/bus/pci/devices/0000\:02\:10.0/driver/unbind
host# echo 0000:02:10.0 > /sys/bus/pci/drivers/pciback/new_slot
host# echo 0000:02:10.0 > /sys/bus/pci/drivers/pciback/bind

**STEP 2**: To append below to Xen HVM config file:

```python
pci = [ '02:10.0' ]
```
Virtualization Demo

The virtualization demo for a new instruction set

- The registers (only one): RAX
- The instructions (only one): mov $7, %rax

---

**demo 1: run instruction once**

- mov %rax, %rbx → save context
- mov $7, %rax → run instruction
- mov %rbx, %rax → restore context

**demo 2: run instruction multiple times**

- mov %rax, %rbx → save context
- mov $7, %rax → multiple times
- mov $7, %rax
- mov $7, %rax
- mov %rbx, %rax → restore context
- mov %rbx, %rax

---

Dongli Zhang

To Trap or Not To Trap: The PCI Passthrough

January 6, 2021 5 / 17
Virtualization Demo

The virtualization demo for a new instruction set
- The registers (only one): **RAX**
- The instructions (only one): `mov $7, %rax`

How about privileged instructions?

```plaintext
mov %rax, %rbx  →  save context
mov $7, %rax    →  run instruction
mov %rbx, %rax  →  restore context
mov %rax, %rbx  →  save context
mov $7, %rax    →  multiple times
mov $7, %rax    →  save context
mov $7, %rax    →  restore context
```

demo 1: run instruction once  demo 2: run instruction multiple times
The CPU is divided into **guest mode** and regular **host mode**

- Some privileged instructions are (trapped to and) emulated by **CPU host mode**
- Some privileged instructions are trapped to and emulated by **hypervisor software**
CPU Virtualization

- The CPU is divided into **guest mode** and regular **host mode**
- Some privileged instructions are (trapped to and) emulated by **CPU host mode**
- Some privileged instructions are trapped to and emulated by **hypervisor software**
- How about memory virtualization?

![Diagram showing CPU Guest Mode and CPU Host (regular) Mode with the process of trapping and emulating privileged instructions and an io instruction.]
Memory Virtualization

- **Guest Page Table (CR3):** Guest Virtual Address to Guest Physical Address
- **Host Page Table (EPTP):** Guest Physical Address to Host Physical Address
- To set EPT entry as non-read, non-write or non-exec can trap guest VM memory access
PCI Device Emulation 1/2

- PCI: config space, capabilities and bar
- The PCI Bar is usually to register DMA ring buffer, configure MSI-X and kick doorbell

![Diagram showing PCI Config Space and BARs](image)

E.g., the offset 0x200 in BAR 1 is the doorbell register to notify hardware to process ring buffer
EPT is configured on purpose to trap **MMIO** bar access

PCI (config/bar) access by **IO** instruction is trapped to hypervisor

Both **IO** and **MMIO** are emulated by hypervisor
PCI Passthrough 1/4: Plan A

- Hypervisor as intermediate layer:
  - Access to PCI config/bar is always trapped to hypervisor
  - Hypervisor access PCI device and propagate result to VM

---

1. IO/MMIO

2. Trap IO/MMIO

3. Redirect IO/MMIO to hardware

4. Result from hardware

5. Propagate result to virtual device and restore VM context

---

Dongli Zhang

To Trap or Not To Trap: The PCI Passthrough

January 6, 2021 10 / 17
PCI Passthrough 1/4: Plan A

- Hypervisor as intermediate layer:
  - Access to PCI config/bar is always trapped to hypervisor
  - Hypervisor access PCI device and propagate result to VM
- Cons: too many traps to hypervisor!

1. IO/MMIO
2. Trap IO/MMIO
3. Redirect IO/MMIO to hardware
4. Result from hardware
5. Propagate result to virtual device and restore VM context
Map all MMIO PCI BARs to VM address space via EPT
Access to PCI config (and IO-based BAR) is trapped to hypervisor
The VM has direct MMIO access to PCI BARs to avoid trap

Q1: DMA is based on HPA, NOT GPA!
Q2: MSI-X IRQ is based on host CPU ID, NOT guest CPU ID!
PCI Passthrough 2/4: Plan B

- Map all MMIO PCI BARs to VM address space via EPT
- Access to PCI config (and IO-based BAR) is trapped to hypervisor
- The VM has direct MMIO access to PCI BARs to avoid trap
- Q1: DMA is based on HPA, NOT GPA!

Q2: MSI-X IRQ is based on host CPU ID, NOT guest CPU ID!

Guest VM

- Map hardware PCI Bars to VM address space via EPT
- PCI Config is IO-based trap/emulation
- BAR 2 and BAR 4 (IRQ) are MMIO-based direct access

Hypervisor

- PCI BAR 2 (Config/MMIO)
- PCI BAR 4 (MSI-X/MMIO)

Q1: DMA is based on HPA, not GPA
Q2: IRQ dest is host CPU id, not guest CPU id
PCI Passthrough 2/4: Plan B

- Map all MMIO PCI BARs to VM address space via EPT
- Access to PCI config (and IO-based BAR) is trapped to hypervisor
- The VM has direct MMIO access to PCI BARs to avoid trap
- Q1: DMA is based on HPA, NOT GPA!
- Q2: MSI-X IRQ is based on host CPU ID, NOT guest CPU ID!

**Diagram:***

1. **Guest VM**
   - Map hardware PCI Bars to VM address space via EPT
   - PCI Config is IO-based trap/emulation
   - BAR 2 and BAR 4 (IRQ) are MMIO-based direct access

2. **Hypervisor**
   - PCI Config (IO)
   - PCI BAR 2 (Config/MMIO)
   - PCI BAR 4 (MSI-X/MMIO)

Q1: DMA is based on HPA, not GPA
Q2: IRQ dest is host CPU id, not guest CPU id
- Map only some PCI BARs (e.g., ring buffer base or doorbell) to VM address space
- Access to PCI config (and IO-based BAR) and MSI-X table is trapped to hypervisor
- Redirect IRQ from PCI device to VM via hypervisor

Q1: DMA is based on HPA, not GPA! (Device Isolation across VMs)
PCI Passthrough 3/4: Plan C

- Map only some PCI BARs (e.g., ring buffer base or doorbell) to VM address space
- Access to PCI config (and IO-based BAR) and MSI-X table is trapped to hypervisor
- Redirect IRQ from PCI device to VM via hypervisor
- Q1: DMA is based on HPA, NOT GPA! (Device Isolation across VMs)

1. Map hardware PCI config/bars to VM address space via EPT
2. BAR 2 (e.g., ring buffer base or doorbell) is MMIO-based direct access
3. The MSI-X IRQ from hardware is mediated by the hypervisor
4. Access to PCI config and MSI-X bar is trapped to and emulated by the hypervisor

Q1: DMA is based on HPA, not GPA.
Map only some PCI BARs (e.g., ring buffer base or doorbell) to VM address space

- PCI Bar 2 (Config/MMIO)
- PCI Config (IO)
- PCI Bar 4 (MSI-X/MMIO)
- IOMMU
- Guest VM
- Hypervisor

Trap to hypervisor for access to PCI config and MSI-X

Direct MMIO

IOMMU-based DMA is based on GPA

IRQ is injected via IOMMU or hypervisor

To Trap or Not To Trap: The PCI Passthrough
January 6, 2021 17 / 17
PCI Passthrough 4/4: Plan D

- Map only some PCI BARs (e.g., ring buffer base or doorbell) to VM address space
- Access to PCI config (and IO-based BAR) and MSI-X table is trapped to hypervisor

**Diagram: PCI Passthrough Plan D**

- Guest VM
  - Trap to hypervisor for access to PCI config and MSI-X
- Hypervisor
  - PCI Config (IO)
  - PCI Bar 4 (MSI-X/MMIO)

**Key Points**

- Direct MMIO
- PCI Bar 2 (Config/MMIO)
- IOMMU-based DMA is based on GPA
- IRQ is injected via IOMMU or hypervisor
- Hardware
PCI Passthrough 4/4: Plan D

- Map only some PCI BARs (e.g., ring buffer base or doorbell) to VM address space
- Access to PCI config (and IO-based BAR) and MSI-X table is trapped to hypervisor
- Redirect DMA GPA to HPA via IOMMU

Diagram:
- Guest VM
  - PCI Bar 2 (Config/MMIO)
  - PCI Config (IO)
  - PCI Bar 4 (MSI-X/MMIO)
  - Trap to hypervisor for access to PCI config and MSI-X
- Hypervisor
  - Direct MMIO
  - IOMMU
  - IOMMU-based DMA is based on GPA
  - IRQ is injected via IOMMU or hypervisor
- Hardware
  - To Trap or Not To Trap: The PCI Passthrough
  - Dongli Zhang
  - January 6, 2021 13 / 17
PCI Passthrough 4/4: Plan D

- Map only some PCI BARs (e.g., ring buffer base or doorbell) to VM address space
- Access to PCI config (and IO-based BAR) and MSI-X table is trapped to hypervisor
- Redirect DMA GPA to HPA via IOMMU
- Redirect IRQ from PCI device to VM via hypervisor/IOMMU

**Diagram:**

- **Guest VM**
  - Trap to hypervisor for access to PCI config and MSI-X
  - PCI Bar 2 (Config/MMIO)
  - Direct MMIO
  - PCI Config (IO)
  - PCI Bar 4 (MSI-X/MMIO)

- **Hypervisor**
  - IOMMU
  - IOMMU-based DMA is based on GPA
  - IRQ is injected via IOMMU or hypervisor
### host# lspci -vv -s 03:00.3

**03:00.3** Ethernet controller: Intel Corporation Ethernet Controller XL710 for 40GbE QSFP+ (rev 02)
- Subsystem: XXXXXX 10 Gb/40 Gb Ethernet Adapter
- Physical Slot: 3
- Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort- <MAbort- >SERR- <PERR- INTx-
- Latency: 0, Cache Line Size: 32 bytes
- Interrupt: pin A routed to IRQ 24
- NUMA node: 0
- **Region 0:** Memory at c1800000 (64-bit, prefetchable) [size=8M]
- **Region 3:** Memory at c4000000 (64-bit, prefetchable) [size=32K]

... ...

**Capabilities:** [70] MSI-X: Enable+ Count=129 Masked-
  - Vector table: BAR=3 offset=00000000
  - PBA: BAR=3 offset=00001000

... ...

Bar 0 is mapped

BAR 3 MSI-X vector table is trapped

BAR 3 MSI-X PBA table is mapped
(qemu) info mtree -f
FlatView #2
...
...
00000000fe000000-00000000fe7fffff (prio 0, ramd): 0000:03:00.3 BAR 0 mmaps[0]
00000000fe800000-00000000fe80080f (prio 0, i/o): msix-table
00000000fe800810-00000000fe800fff (prio 0, i/o): 0000:03:00.3 BAR 3 @00000000000000810
00000000fe801000-00000000fe807fff (prio 0, ramd): 0000:03:00.3 BAR 3 mmaps[0]
...
...
MSI-X Pending Bit Array (PBA)

- **ramd**: PCI bar/region is mapped to VM address space
- **i/o**: access is trapped to and emulated by hypervisor
NVMe Developers

To kick the doorbell register for IO queue 3 does not work

Virtualization Developers

We do not understand NVMe spec/manual!

- Both developers understand PCI specification
- Write to BAR 0 offset 0x120 does not take effect
Virtualization is \textit{trap and emulation} (explicitly by software or implicitly by hardware).
Virtualization is **trap and emulation** (explicitly by software or implicitly by hardware).

PCI Passthrough is to **minimize the cost that VM accesses PCI device**, via:

- Map only some PCI BARs (e.g., ring buffer base or doorbell) to VM address space via EPT.

Congratulations! You have learned about to implement virtualization demo from scratch!
Virtualization is **trap and emulation** (explicitly by software or implicitly by hardware)

PCI Passthrough is **to minimize the cost that VM accesses PCI device**, via:
- Map only some PCI BARs (e.g., ring buffer base or doorbell) to VM address space via EPT
- Access to PCI config (and IO-based BAR) and MSI-X table is trapped to and emulated by hypervisor
Virtualization is **trap and emulation** (explicitly by software or implicitly by hardware).

PCI Passthrough is **to minimize the cost that VM accesses PCI device**, via:

- Map only some PCI BARs (e.g., ring buffer base or doorbell) to VM address space via EPT.
- Access to PCI config (and IO-based BAR) and MSI-X table is trapped to and emulated by hypervisor.
- Redirect DMA GPA to HPA via IOMMU.

Congratulations! You have learned about to implement virtualization demo from scratch!
Virtualization is **trap and emulation** (explicitly by software or implicitly by hardware).

PCI Passthrough is **to minimize the cost that VM accesses PCI device**, via:

- Map only some PCI BARs (e.g., ring buffer base or doorbell) to VM address space via EPT.
- Access to PCI config (and IO-based BAR) and MSI-X table is trapped to and emulated by hypervisor.
- Redirect DMA GPA to HPA via IOMMU.
- Redirect IRQ from PCI device to VM via hypervisor/IOMMU.
Take-Home Message

- Virtualization is **trap and emulation** (explicitly by software or implicitly by hardware)
- PCI Passthrough is **to minimize the cost that VM accesses PCI device**, via:
  - Map only some PCI BARs (e.g., ring buffer base or doorbell) to VM address space via EPT
  - Access to PCI config (and IO-based BAR) and MSI-X table is trapped to and emulated by hypervisor
  - Redirect DMA GPA to HPA via IOMMU
  - Redirect IRQ from PCI device to VM via hypervisor/IOMMU
- To bridge the gap (between NVMe/Ethernet/IB and virtualization) via PCI spec
Virtualization is **trap and emulation** (explicitly by software or implicitly by hardware)

PCI Passthrough is **to minimize the cost that VM accesses PCI device**, via:

- Map only some PCI BARs (e.g., ring buffer base or doorbell) to VM address space via EPT
- Access to PCI config (and IO-based BAR) and MSI-X table is trapped to and emulated by hypervisor
- Redirect DMA GPA to HPA via IOMMU
- Redirect IRQ from PCI device to VM via hypervisor/IOMMU

To bridge the gap (between NVMe/Ethernet/IB and virtualization) via PCI spec

Congratulations! You have learned about to implement virtualization demo from scratch!