AMD

Analysis of AMD HW-Assisted vIOMMU Implementation and Performance

Wei Huang
Suravee Suthikulpanit

KVM Forum 2021
Sept 15th 2021

Agenda

- AMD HW-vIOMMU Overview
- Software Prototype
- Benchmarks & Performance Analysis
- Summary

AMD HW-vIOMMU

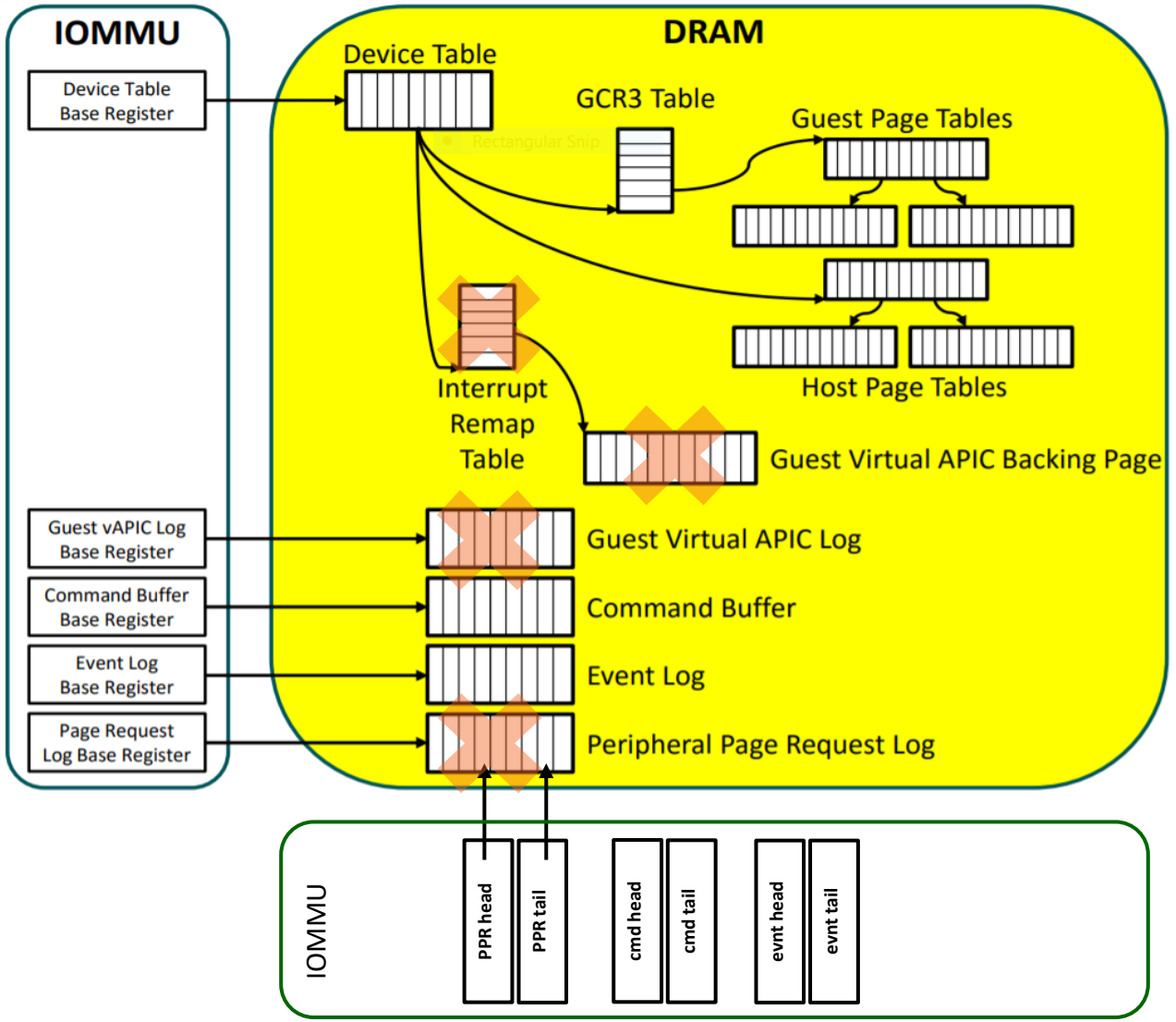
Overview

AMD

AMD IOMMU DMA-Remap

- Data structures
  - Device table
  - Command buffer
  - Event log

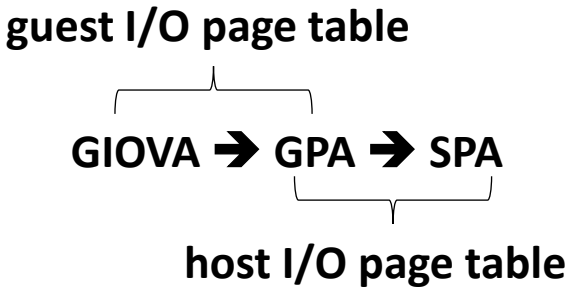
- Two types of IOMMU page table
  - Host IO Page-table (v1)
    - IOMMU-specific
    - E.g. VFIO (GPA -> SPA)
  - Guest IO Page-table (v2)
    - x86-compatible
    - E.g. DMA APIs (GVA -> GPA)
  - 3 modes of operation
    - V1 only
    - V2 only
    - V1 + V2 (nested)

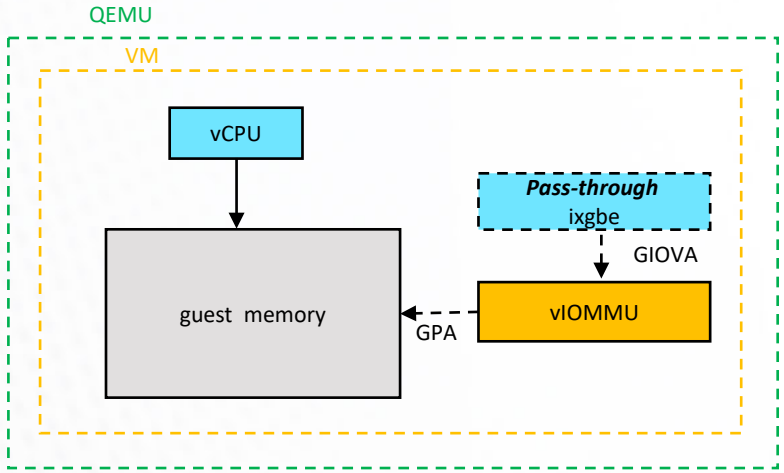


Guest vIOMMU Support

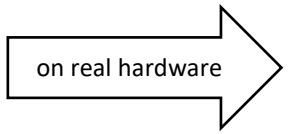
- Usage cases
  - Guest IO-protection
  - Shared Virtual Memory (SVM)
  - DPDK support inside VMs

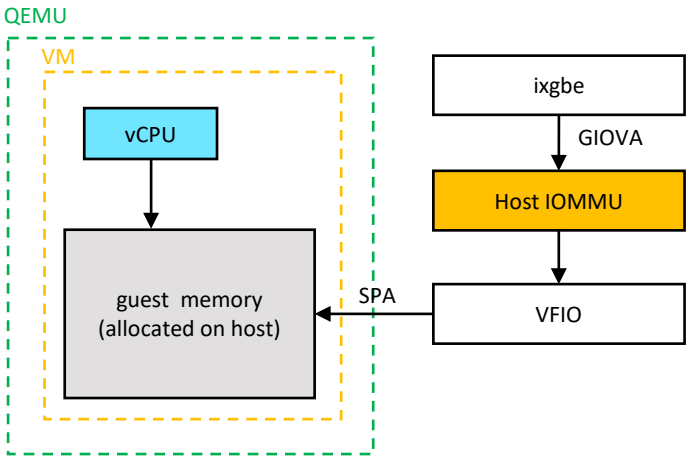
- SW-based vIOMMU
  - Need to intercept and emulate MMIO registers behavior
  - Handling guest I/O page table map/unmap/invalidation





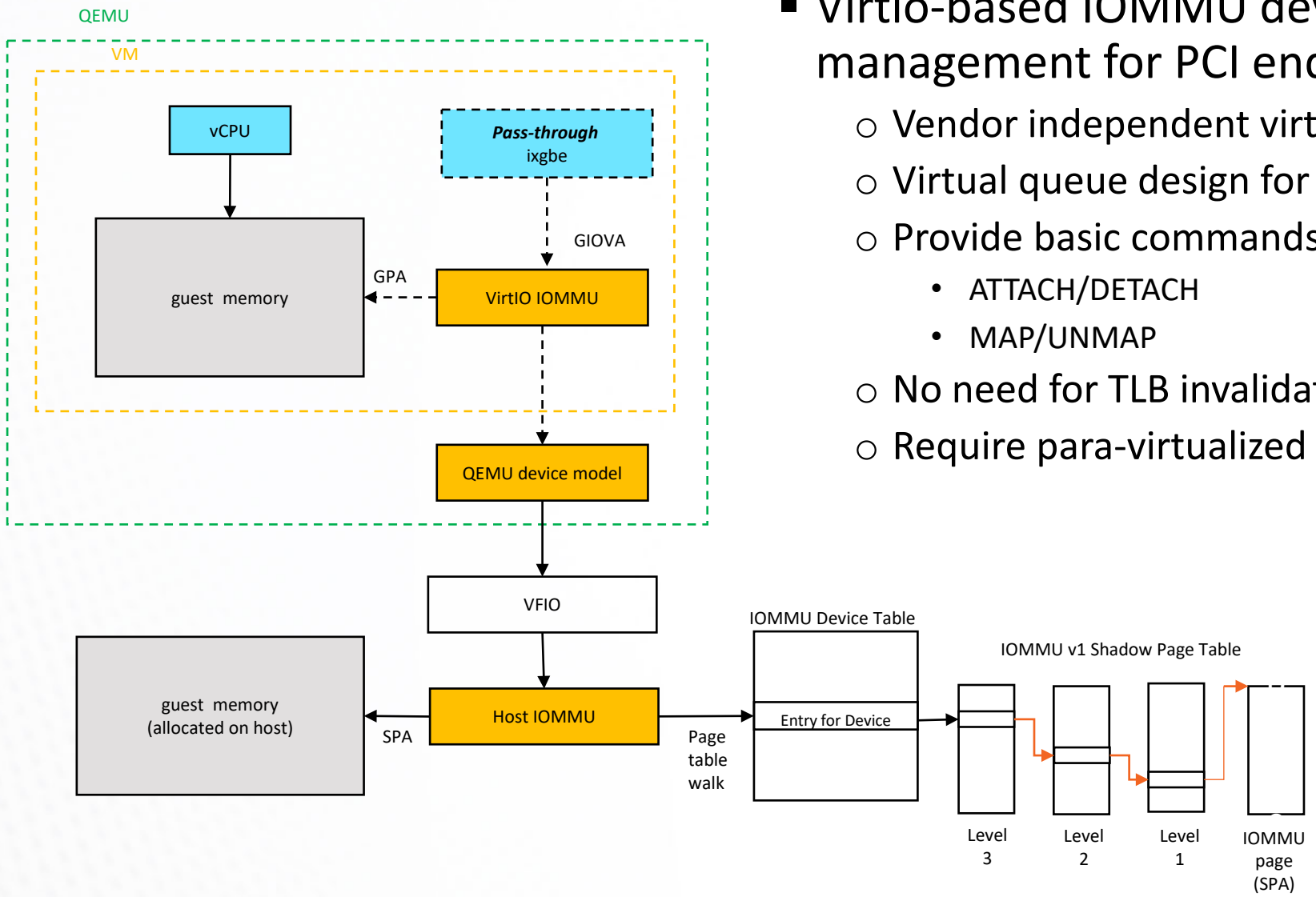
Guest View





Hardware View

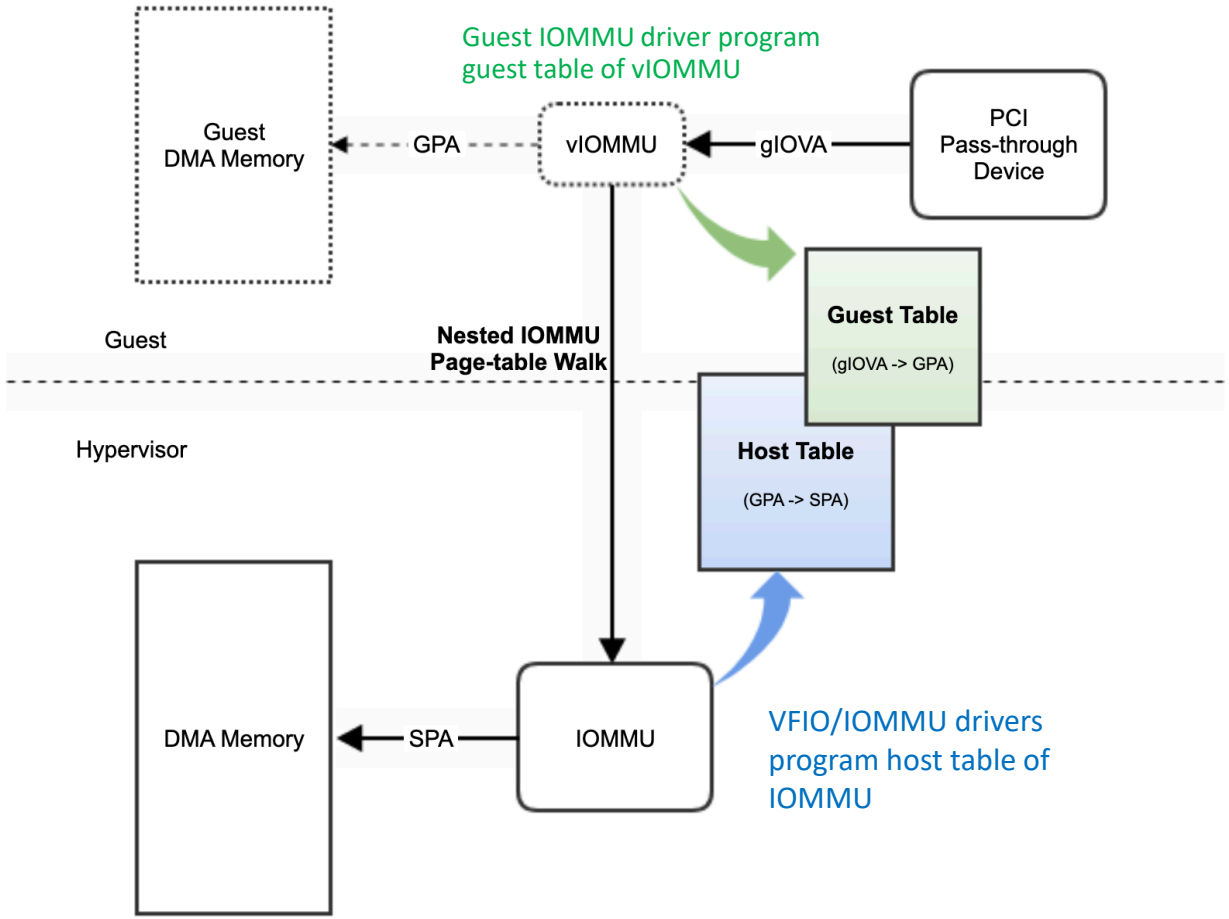
VirtIO-IOMMU



- Virtio-based IOMMU device that offers DMA management for PCI end-point devices
  - Vendor independent virtual IOMMU design
  - Virtual queue design for requests and events
  - Provide basic commands for IOMMU management
    - ATTACH/DETACH
    - MAP/UNMAP
  - No need for TLB invalidation command
  - Require para-virtualized driver in the guest VM

HW-Assisted vIOMMU (HW-vIOMMU)

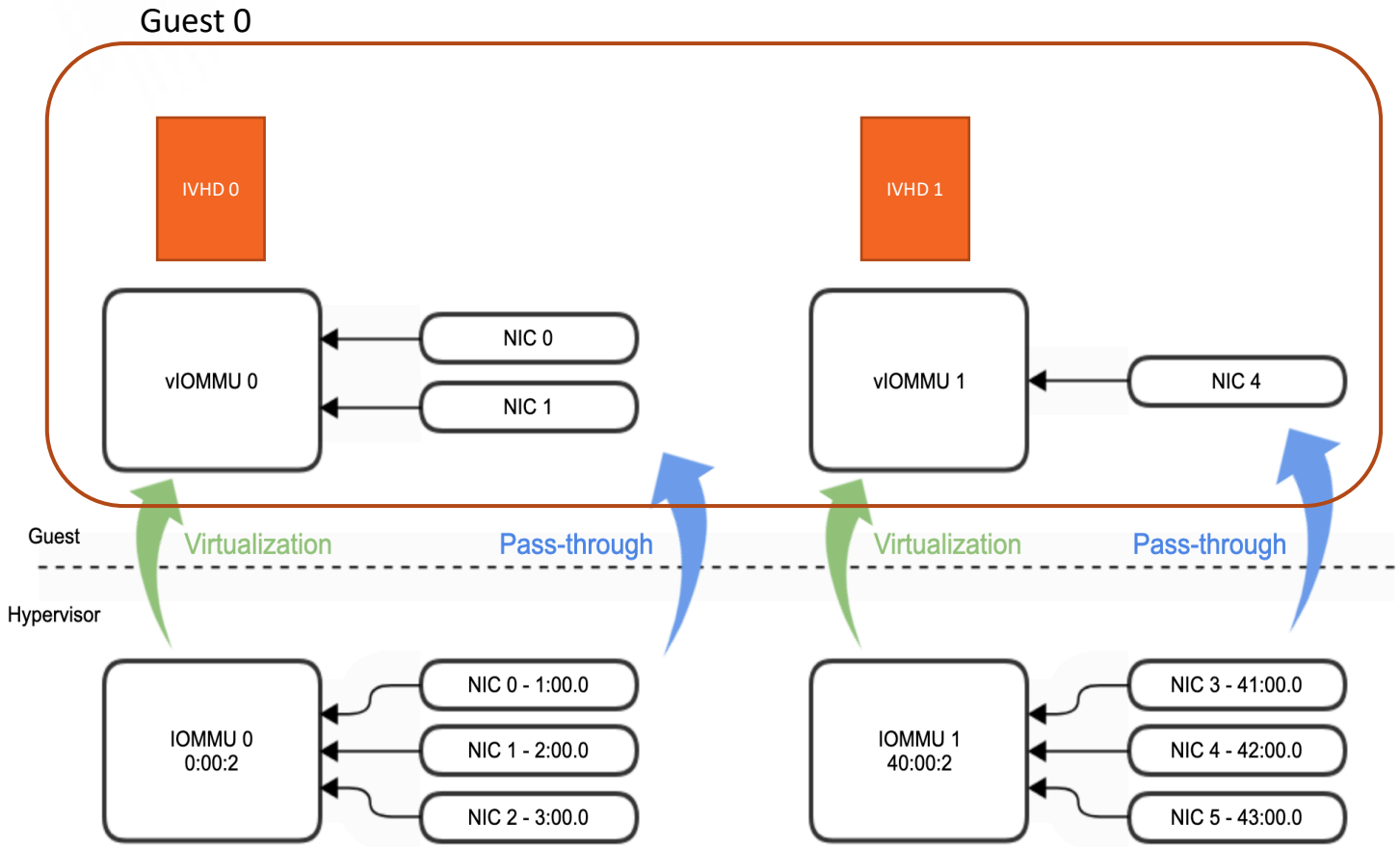
- Provides a virtual IOMMU for guest VMs, targeting at PCI pass-through devices
- Required for DMA to SEV-SNP (Secure Nest Paging) Guest
- Improve performance on guest IOMMU via
  - HW-virtualized **guest IOMMU commands**
  - HW-virtualized **MMIO registers**
    - Head / Tail pointers
  - HW-virtualized **Event and PPR logs**
  - **Nested IO page-table**
- Nested IO page-table
  - Guest (v2) table for **gIOVA -> GPA**
    - Managed by guest IOMMU driver
  - Host (v1) table for **GPA -> SPA**
    - Managed by VFIO driver



Software Prototype

QEMU Changes

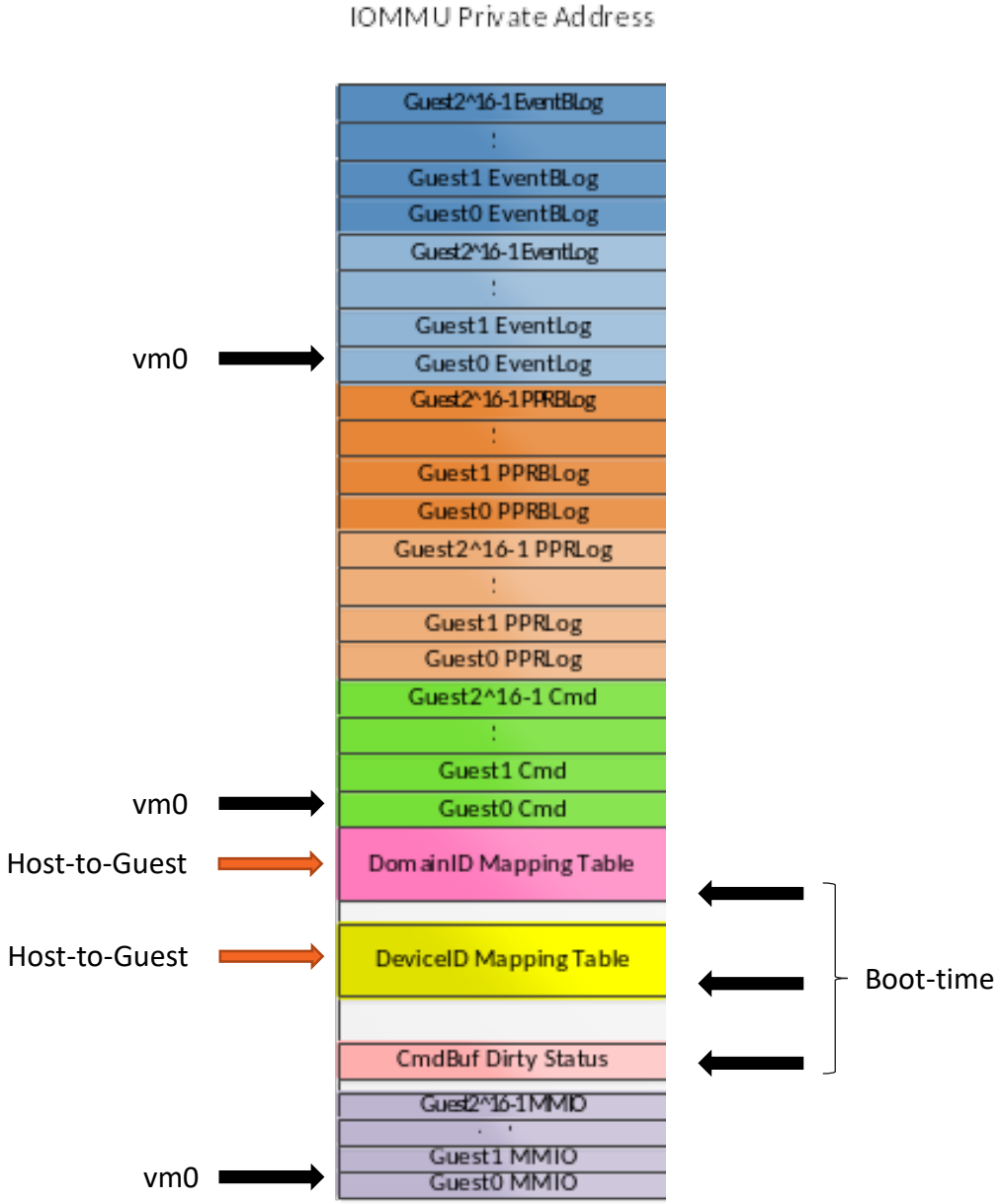
- New amd-viommu device model
- PCI topology b/w vIOMMU and PT devices
  - Each PCI pass-through device must be associated with the corresponding amd-viommu instance
- New guest ACPI IVRS table
  - To support additional IVHD blocks for the new amd-viommu device model




```
qemu-system-x86_64 -machine q35 \
    -device  ioh3420,bus=pcie.0,addr=1c.0,multifunction=on,port=1,chassis=1,id=root.1 \
    -device  amd-viommu,host=0000:00:00.2,id=0 \
    -device    vfio-pci,host=0000:01:00.0,iommu-id=0 \
    -device    vfio-pci,host=0000:02:00.0,iommu-id=0 \
    -device  amd-viommu,host=0000:40:00.2,id=1 \
    -device    vfio-pci,host=0000:42:00.0,iommu-id=1 \
```

Host IOMMU Driver Changes

- Boot-time initialization
  - Logic for detect and enable vIOMMU feature
  - Allocate commonly used data structure:
    - Domain ID table
    - Device ID table
    - CmdBuf dirty status table
- Per-VM initialization
  - Allocate per-vm data structure:
    - Guest MMIO registers
    - Guest Cmd buffer
    - Guest Evt log
  - Host-to-Guest DevID and DomID mappings
  - Trap guest access to 1st 4K of MMIO region (control)
- Support new IOMMU commands and events



AMD

QEMU & Host IOMMU Driver Interface

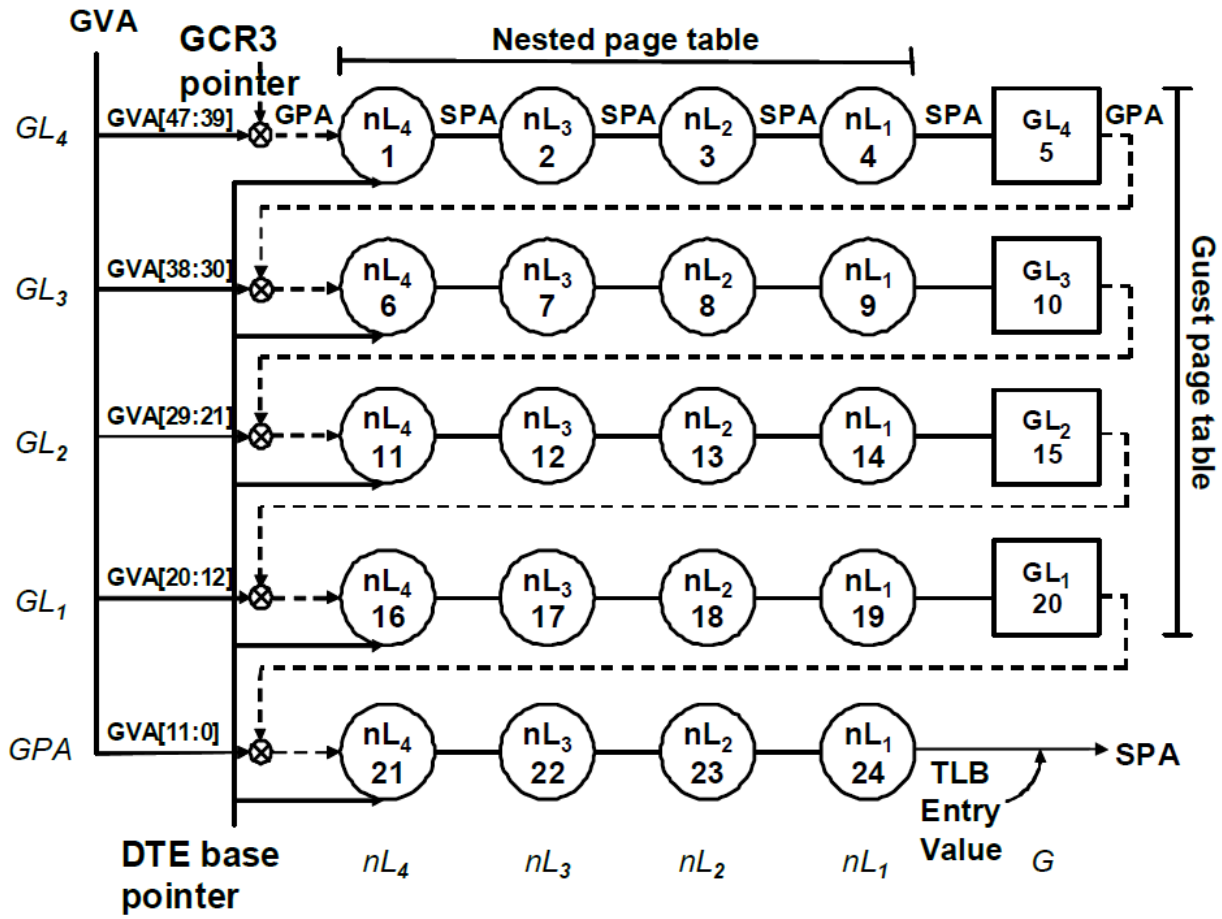
- Introduce new device FS
  - /dev/amd-viommu
- Introduce vIOMMU-specific IOCTL interfaces

Category	IOCTL Commands	Description
VM	• VIOMMU_VM_INIT • VIOMMU_VM_DESTROY	Setup/destroy per-vm vIOMMU data structure (e.g. Guest MMIO, CmdBuf Dirty Status, Guest Cmd, PPR)
Device	• VIOMMU_DEVICE_ATTACH • VIOMMU_DEVICE_DETACH	Setup/destroy DevID Mapping Table
Domain	• VIOMMU_DOMAIN_ATTACH • VIOMMU_DOMAIN_DETACH	Setup/destroy DomainID Mapping Table
MMIO	• VIOMMU_MMIO_ACCESS	Handle the trapped guest MMIO access
Command buffer	• VIOMMU_CMDBUF_MAP_UPDATE	Communicate the guest command buffer address to the host vIOMMU driver to setup GPA→SPA mapping in the host I/O page table
GCR3	• VIOMMU_GCR3_TABLE_UPDATE	Communicate when guest IOMMU driver update the GCR3 table with pointer to guest page table.

Guest AMD IOMMU Driver Changes

- For nested IO page table, guest IOMMU driver programs GIOVA -> GPA in the IOMMU guest page table.
  - VFIO programs **GPA -> SPA** in the IOMMU host page table.
  - IOMMU hardware walks through guest + host table to translate **GIOVA -> SPA** when receive DMA request.

- New IOMMUv2 page table support for DMA-API
  - Use the generic IO page table framework
  - Add command line option *"amd_iommu_iova=[v1 (default)|v2]"*
  - Use PASID=0 (for io-protection mode)
  - https://lore.kernel.org/patchwork/cover/1394015



From: 48882—AMD I/O Virtualization Technology (IOMMU) Specification

AMD

Benchmarks
&
Performance Analysis

Goals

- Compare PCI device pass-through performance of AMD HW-vIOMMU with other vIOMMU solutions.
- Analyze AMD HW-vIOMMU performance data and identify room for improvement.

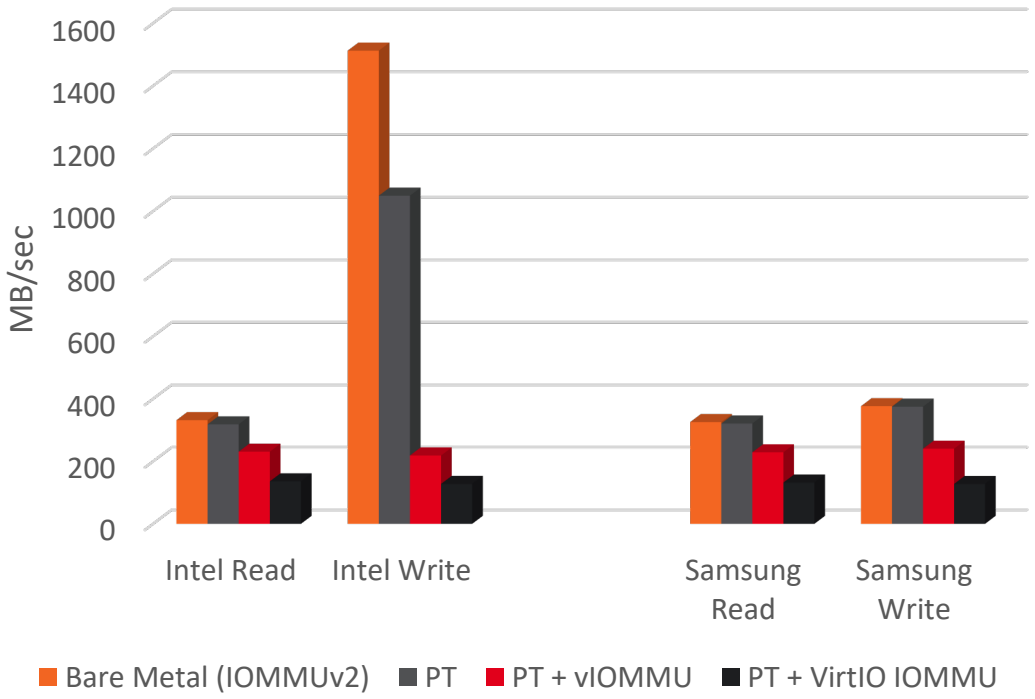
Configurations

- Bare-metal + No IOMMU
  - Host Kernel : v5.12 (iommu=pt)

- VM + PT:
  - No Guest IOMMU
    - Host Kernel : v5.12
    - Guest Kernel : v5.12
    - Qemu : v6.0
  - AMD HW-vIOMMU
    - Host Kernel : v5.12 **w/ vIOMMU support**
    - Guest Kernel : v5.12 **w/ io-pagetable support for IOMMUv2 page table**
    - Qemu : v6.0 **w/ AMD vIOMMU device model**
  - VirtIO IOMMU
    - Host Kernel : v5.12
    - Guest Kernel : v5.13-rc4 **w/ x86 virtio-iommu**
    - Qemu : v6.0 **w/ VirtIO IOMMU device model**

FIO

- Samsung SSD 960EVO (m.2)
- Intel SSD DC P3700 Series (u.2)
- Use the same Linux NVME driver
- Use default configurations for OS and benchmark

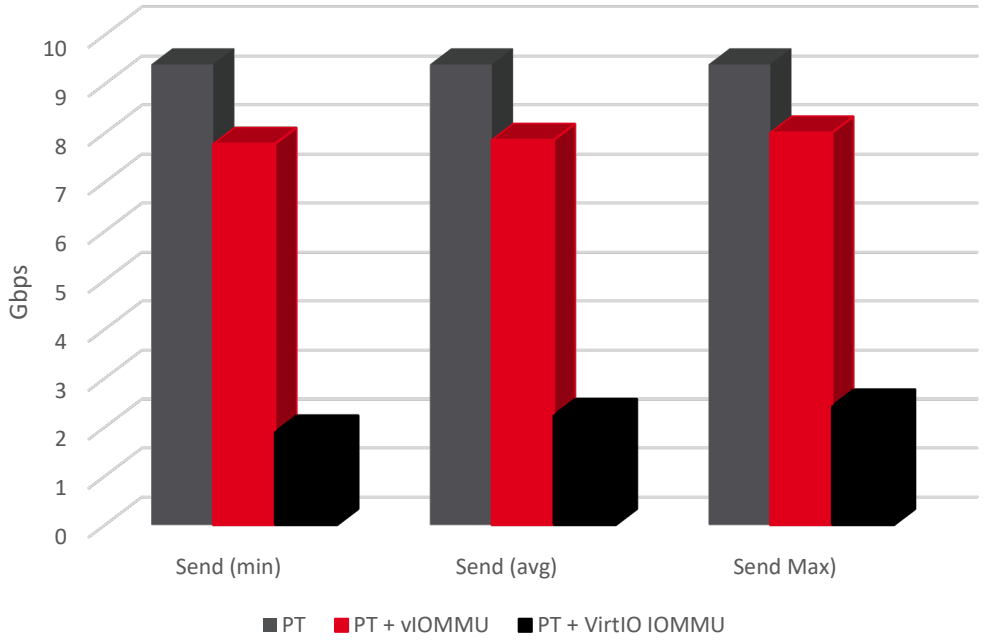


	Intel		Samsung	
Configuration	**Read (MB/sec)**	**Write (MB/sec)**	**Read (MB/sec)**	**Write (MB/sec)**
Bare Metal (no IOMMU)	331.82	1512.4057	327.60	376.48
PT (no IOMMU)	318.38	1049.42	321.04	374.89
PT + HW-vIOMMU	**230.88 (72.5%)**	**218.66 (20.8%)**	**228.82 (72.3%)**	**240.33 (64.1%)**
PT + VirtIO IOMMU	136.02 (42.7%)	127.70 (12.2%)	132.04 (41.1%)	127.56 (34.0%)

fio --thread --size=100% --direct 1 --buffered 0 --iodepth 8 --invalidate 1 --bs 4096 --ioengine libaio --time_based --norandommap --random_generator=lfsr --cpus_allowed_policy=split --exitall --ramp_time 5 --runtime 60 --allrandrepeat 0 --rw rand[read|write]

NETPERF : TCP_STREAM

- Intel 82599ES 10GbE
  - Dual ports
  - Physical loop-back mode
- Linux ixgbe driver
- Use default configuration for benchmark and OS
- Use PT (no IOMMU) case for base-line

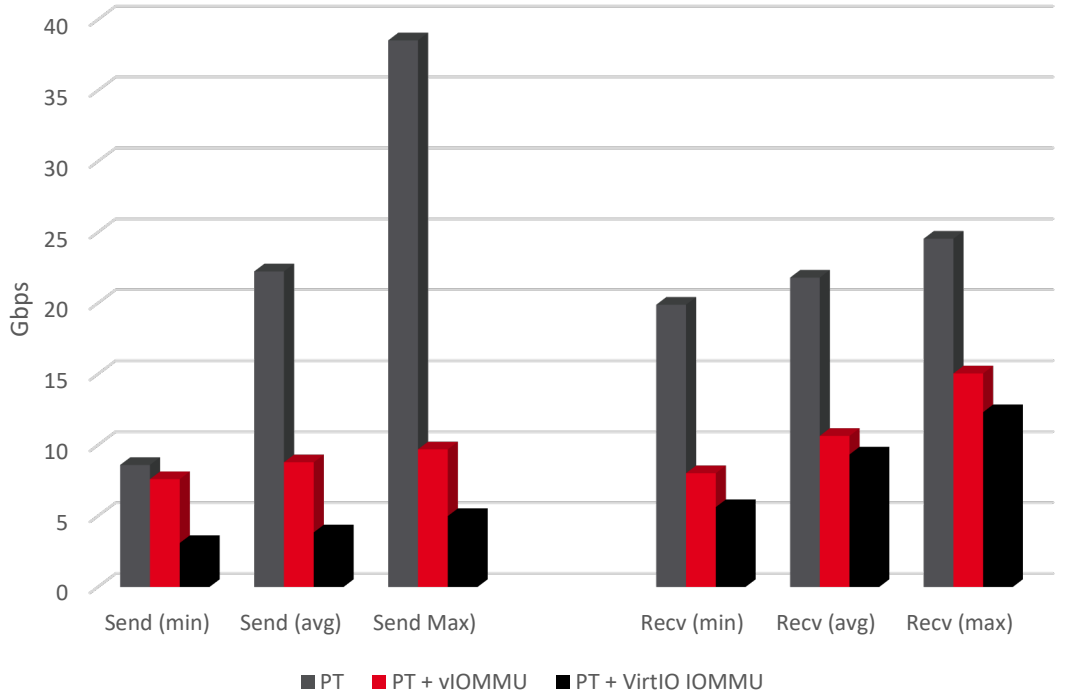


Intel 10GbE	Send (Gbps) Min / Avg / Max	Receive (Gbps) Min / Avg / Max
PT (no IOMMU)	9.4	9.4
PT + HW-vIOMMU	7.78 (84.47%) **7.86 (83.68%)** 8.01 (82.77%)	**9.4**
PT + VirtIO IOMMU	1.91 (20.31%) **2.25 (23.93%)** 2.44 (25.96%)	**9.4**

AMD

NETPERF : TCP_STREAM

- Mellanox MT28800 100GbE
  - Dual port
  - Physical loop-back mode
- Linux mlx5_core driver
- Default configuration



Mellanox 100GbE	Send (Gbps) Min / Avg / Max	Receive (Gbps) Min/ Avg / Max
PT (no IOMMU)	8.66 **22.24** 38.54	19.89 **21.80** 24.55
PT + HW-vIOMMU	7.61 (34.1%) **8.79 (30.0%)** 9.70 (25.0%)	8.02 (40.3%) **10.65 (48.8%)** 15.05 (61.3%)
PT + VirtIO IOMMU	3.10 (13.9%) **3.85 (13.8%)** 5.00 (12.9%)	5.63 (28.3%) **9.34 (42.8%)** 12.32 (50.2%)

NETPERF : TCP_STREAM (Intel vs Mellanox)

For Netperf, AMD HW-vIOMMU performance bottleneck:

- Send : 8Gbps
- Receive : 10Gbps

Intel 10GbE	Send (Gbps) Min / Avg / Max	Receive (Gbps) Min / Avg / Max
PT (no IOMMU)	9.4	9.4
PT + HW-vIOMMU	**7.78 (84.47%)** **7.86 (83.68%)** **8.01 (82.77%)**	**9.4**

Mellanox 100GbE	Send (Gbps) Min / Avg / Max	Receive (Gbps) Min/ Avg / Max
PT (no IOMMU)	8.66 22.24 38.54	19.89 21.80 24.55
PT + HW-vIOMMU	**7.61 (34.1%)** **8.79 (30.0%)** **9.70 (25.0%)**	**8.02 (40.3%)** **10.65 (48.8%)** **15.05 (61.3%)**

PT + vIOMMU Overhead Analysis

- Guest INVALIDATE_IOMMU_PAGES
  - Instruct IOMMU to invalidate range of entries in TLB

- IOMMU TLB misses
  - Perf stat w/ IOMMU performance counters (host)
  - Large amount of IOMMU TLB miss PTE/PDE in vIOMMU case
  - No IOMMU TLB miss in PT case, since only use the IOMMU host page table (pinned)

- Nested page table walk
  - Higher latency than non-nested page table walk

Summary

- AMD HW-vIOMMU shows improvement of PCI pass-through I/O performance for guest IO-protection use case, when comparing to other SW-vIOMMU solutions

- Improvements:
  - Better IOMMU TLB invalidation scheme to reduce the number of :
    - IOMMU invalidations
    - Number of page-table walks

- Next Steps
  - Upstreaming to Linux / Qemu
  - Experiment w/ other vIOMMU usecases
  - Solution for hybrid vIOMMU model (SW+HW vIOMMU)
  - Interrupt Remapping support w/ HW vIOMMU

Thank you

AMD

DISCLAIMER AND ATTRIBUTIONS

DISCLAIMER

The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD’s products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. GD-18