Full GPU virtualization in mediated pass-through way

Zhenyu Wang
So...GPU virtualization

GPU Accelerated Tasks
- Games
- Video Playback/Edit
- Web Experience
- Office Productivity
- User Interface
- Computer Aided Design
- Weather broadcast
Topics

• Intel GPU virtualization approaches
• A little history of GVT-g project
• VFIO with mediated device
• GVT-g device model
Intel GPU virtualization approaches

**API Forwarding (GVT-s)**
- **VM**: Frontend
- **Backend**: Graphics Driver
- Pros: Performance, Sharing
- Cons: No media/GPGPU, Lacks compatibility

**Direct Pass-thru (GVT-d)**
- **VM**: Graphics Driver
- **Hypervisor**: Device
- Pros: Performance, Capability
- Cons: No sharing

**Full GPU Virtualization (GVT-g)**
- **VM**: Graphics Driver
- **Hypervisor**: Device
- Pros: Performance, Capability, Sharing
GVT-g history

2011-2012  Project Start: First XenGT POC done in 2012 on Sandybridge

2013       Work with Citrix for XenGT production, Haswell support

2014       Add KVMGT support, Broadwell support

2015       GVT device model rewrite for upstream i915, Skylake support

2016       Engage in VFIO/mdev model, KVMGT upstream merged at 4.10!

2017       Kabylake and more features support, XenGT upstream ongoing
GVT-g architecture overview
VFIO

- Original “Virtual Function I/O” → “Versatile Framework for userspace I/O”
  - VFIO is a secure, userspace driver framework
  - IOMMU-based DMA mapping and isolation (iommu_group)
  - Full device access (MMIO, I/O port, PCI config)
  - Used for physical device assign to VM
    - now for virtual device assignment
- Device assignment = userspace driver
  - Access to device resources
  - Isolation and secure DMA mapping through an IOMMU
  - Interrupt signaling support
VFIO resource access

- Divided into regions with index
- Each region maps to a device resource
  - MMIO Bar, IO Bar, PCI config space
- Region count and info discovered through ioctl
  - `VFIO_DEVICE_GET_REGION_INFO`
- Fast “mmap”, slow “read/write”
- Access Path
  - Trapped by Hypervisor (QEMU/KVM)
  - MemoryRegion lookup performed
  - MemoryRegion.{read,write} accessors called
  - Read/write VFIO region offsets
Mediated device framework

- Co-work from NVidia, Redhat, Intel, IBM
- Represent virtual device to userspace via VFIO interface
- Virtual device access is handled by vendor-specific driver to mediate resource sharing

```
Mdev Bus driver

Mdev core module

Physical Device interface

mdev_register_driver()

VFIO mdev

Probe()/remove()

VFIO user api

mdev_register_device()

kvmgt.ko

callbacks

I915/gvt

GPU hw

nvidia.ko
```
Mdev create & assignment

- Vendor driver register device
  - mdev_create: create virtual device
  - mdev_destroy
  - mdev_supported_types: typed mdev configuration
    - vGPU types base on memory/fence/resolution configs
- UUID based device node:
  - /sys/bus/mdev/devices/$UUID/
- Get VFIO device file descriptor and present to VM
MDev resource access

- Get region info via VFIO ioctl from vendor driver
- Guest MMIO access trapped by KVM
- KVM forward to QEMU VFIO driver
- Convert to R/W request on VFIO device regions
- Handled by mediated vendor driver in kernel
MDev DMA

- QEMU setup guests memory
- VFIO_MAP_DMA with \{GFN, VA\}
- Vendor driver call VFIO pin pages to get PFN
  - VFIO keep reference counted pinned \(<\text{IOVA}, \text{PFN}>\)
- Vendor driver call dma map API for pIOVA
MDev interrupt

- QEMU setup KVM irqfd
- QEMU notify vendor driver with irqfd via VFIO interface
  - VFIODEVICE_SET_IRQS
- Vendor driver inject interrupt by signaling on eventfd
  - Directly inject into VM
GVT-g device model

- GVT-g device model
  - gvt
    - vGPU state
    - vGPU state
    - MMIO handler
    - GTT balloon
    - Shadow mm
    - vGPU scheduler
    - PCI config handler
    - Virtual interrupt
    - Virtual execclist
    - Virtual display
  - Resource alloc
  - Request submission
  - Request done
  - I915
  - i915
  - Request submission
  - Request done
  - Resource alloc
- VFS/KVM
  - Guest mem access/pin/unpin
  - IRQ injection
  - Write protect tracking
  - GPU HW
  - Virtual display
  - Virtual execclist
  - Virtual interrupt
  - MMIO handler
- GVT-g device model
- Intel® Open Source Technology Center
GVT-g device model

- where GPU virtualization logic actually lives
  - virtual GPU state maintenance for VM
  - MMIO handler to emulate HW access behavior for guest driver
  - GPU workload submission emulation and VM notification
vGPU memory manage

- Global graphics memory is partitioned
- Ballooned space
- Aperture access without trap

- Fully shadowed PPGTT
- Use KVM guest page write protect for page table update tracking
vGPU workload execution

- vGPU shadow context
- Virtualized executor interface
- Command parser on vGPU ring/privileged buffer
  - Emulated user interrupt
vGPU scheduling

- vGPU instance: time based scheduling
- Scheduled vGPU instance can submit requests to all engines
  - Per-engine work thread
- Scheduler policy based on vGPU weight
vGPU full virtualized display
HOWTO

• Kernel config (>= 4.10)
  - CONFIG_VFIO, CONFIG_VFIO_MDEV, CONFIG_VFIO_MDEV_DEVICE
  - CONFIG_DRM_I915_GVT, CONFIG_DRM_I915_GVT_KVMGT
  - i915.enable_gvt=1

• Create mdev (vGPU)
  - “uuid > /sys/devices/pci0000:00/0000:00:02.0/mdev_supported_types/i915-GVTg_V5_4/create”

• Start VM
  - “qemu-system-x86_64 -m 1024 -enable-kvm -device vfio-pci,sysfsdev=/sys/bus/pci/devices/0000:00:02.0/$UUID...”

• Detailed HOWTO
  - https://github.com/01org/gvt-linux/wiki
Current upstream status

- KVMGT fully support in upstream, kernel >=4.10, qemu
- Support Broadwell/Skylake/Kabylake for Linux (includes Android) guest (kernel >= 4.8) and Windows guest
- All kinds of GPU applications are supported in guest
  - Although some media features missed for GuC/HuC firmware support
- MTBF time (1 Windows VM): more than 1 week
- Performance (Media workload)
  - Peak perf 95% of native host (1VM)
  - Reach average over 85% performance of native (1 VM run)
- Links:
  - https://github.com/01org/gvt-linux.git
  - https://01.org/igvt-g