VFIO on sPAPR/POWER

- IOMMU, Groups, containers
- VFIO driver stack
- Bind PCI device to VFIO
- Running QEMU with VFIO
- PCI support in QEMU
- DMA handling of KVM guest
- Dynamic DMA windows (POWER7/8)
- DMA handling of sPAPR KVM guest
- Performance under PowerKVM

Why

Provide isolated access to real PCI devices for KVM guests on POWER8 box

SR-IOV is the target (even though VFIO does not do anything special about it)

Compared to virtio/vhost/macvtap:

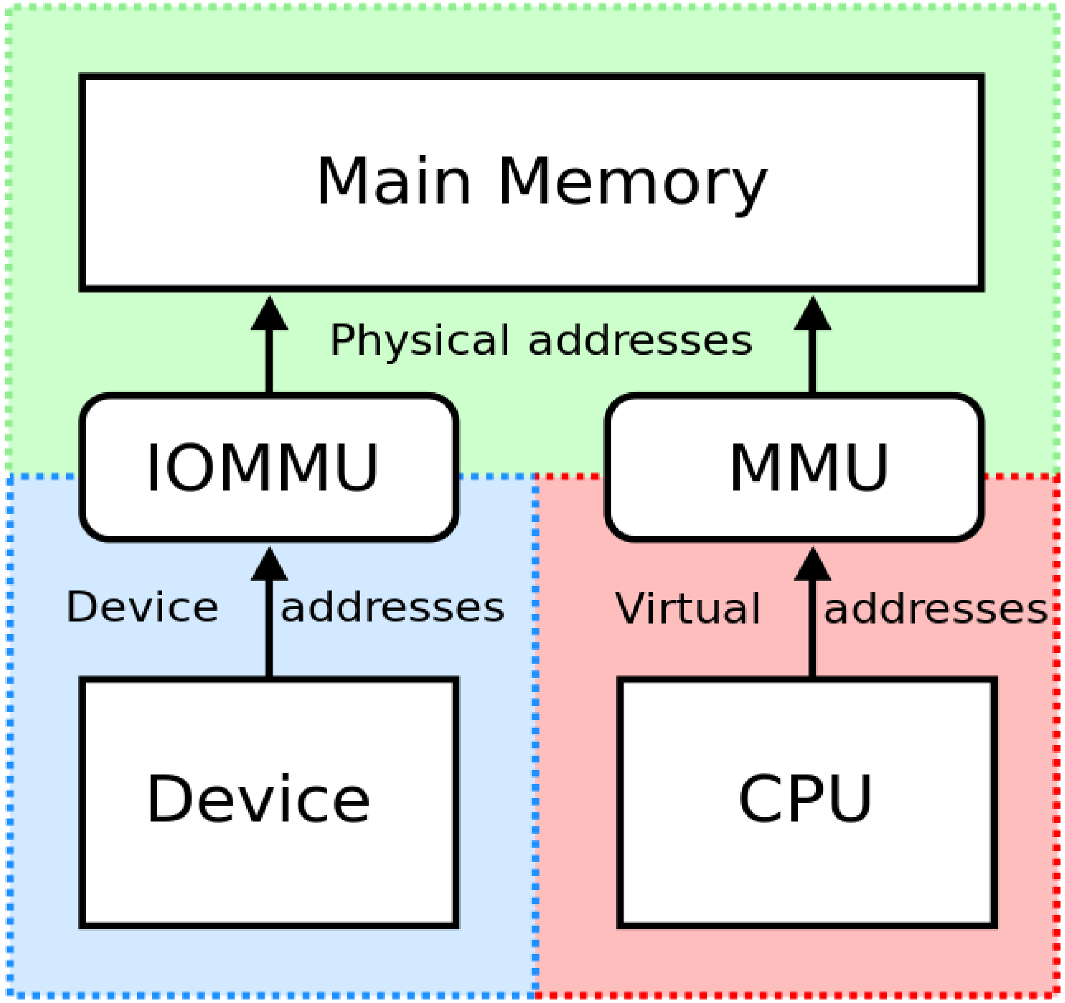
- low latency
- CPU offload

Terminology

- QEMU: Quick EMUlator
  - fully emulates any guest CPU on any host CPU
  - 99% users use it in accelerated mode (KVM) but not 100%
- KVM: Kernel-based Virtual Machine
  - requires support from host CPU
  - supports many architectures
- VFIO: provides userspace secure access to PCI hardware
  - >99.9% users QEMU-KVM but **<u>not</u>** 100%
  - It has <u>nothing</u> to do with VIRTIO or IBM VIO
  - provides PCI function isolation
- sPAPR (**S**erver **P**OWER **A**rchitecture **P**latform **R**equirements a.k.a. PAPR+): defines para-virtual interface
  - implemented by IBM PowerVM
  - device tree
  - hypercalls API

IOMMU

Hardware mapping of bus addresses to RAM.
Allows DMA for guests



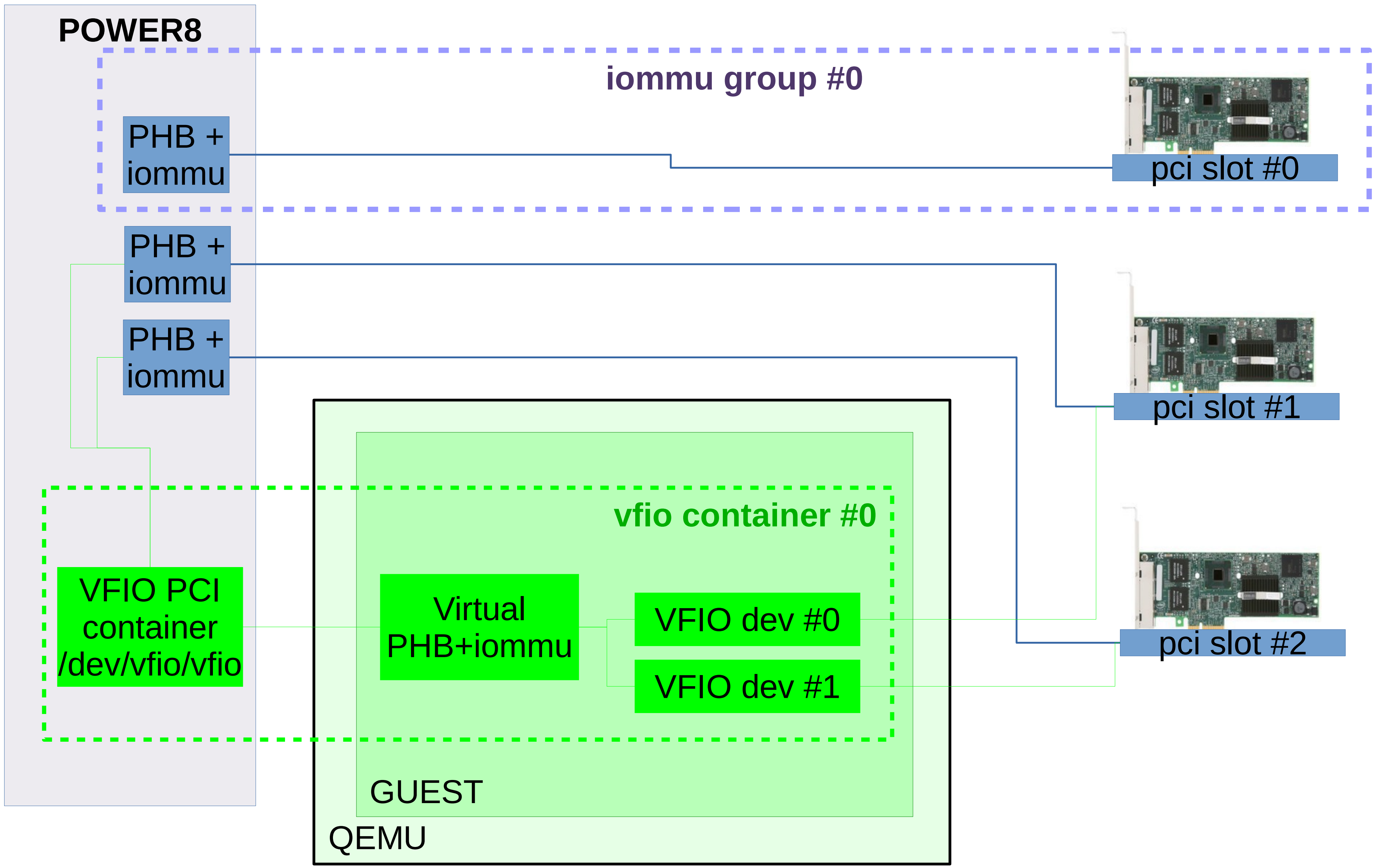
http://en.wikipedia.org/wiki/IOMMU

Groups vs. containers

- IOMMU Group: set of PCI functions which can be isolated
  Exposed to userspace as:
  ```
  /sys/kernel/iommu_groups/0/devices/*
  /sys/kernel/iommu_groups/1/devices/*
  ...
  ```
  - Platform initialization code assigns devices to groups
  - Do not trust devices and do not split functions between groups.
  - Except SR-IOV

- VFIO Container (<u>nothing</u> to do with LXC)
  Represents an IOMMU, it handles map/unmap, not a device

VFIO POWERKVM



VFIO driver stack

- VFIO PCI driver: `vfio_pci` module
- VFIO driver: `vfio` module
  - containers: `/dev/vfio/vfio`
  - groups: `/dev/vfio/0`, `/dev/vfio/1`,...
- VFIO IOMMU driver:
  - default driver (uses PCI bus `iommu_ops`)
    - `vfio_iommu_type1` module (x86/arm/embed.ppc)
  - SPAPR TCE driver (does not use `iommu_ops`)
    - `vfio_iommu_spapr_tce` module

Bind PCI device to VFIO

```
aik@palm:~$ echo 0000:01:00.0 > /sys/bus/pci/devices/0000:01:00.0/driver/unbind

aik@palm:~$ echo “10de 1023” > /sys/bus/pci/drivers/vfio-pci/new_id

aik@palm:~$ lspci -nnvs 0000:01:00.0

0000:01:00.0 3D controller [0302]: NVIDIA Corporation GK110BGL [Tesla K40m]
[10de:1023] (rev a1)

    Subsystem: NVIDIA Corporation Device [10de:097e]

    Flags: fast devsel, IRQ 504

    Memory at 3fe000000000 (32-bit, non-prefetchable) [disabled] [size=16M]

    Memory at 3b0400000000 (64-bit, prefetchable) [disabled] [size=16G]

    Memory at 3b0200000000 (64-bit, prefetchable) [disabled] [size=32M]

    Capabilities: <access denied>

    Kernel driver in use: vfio-pci
```

Running QEMU with VFIO

- x86:

```
-device vfio-pci,host=01:2.3
```

- sPAPR:

```
-device spapr-pci-vfio-host-bridge,id=mybus \
-device vfio-pci,host=0000:01:2.3,bus=mybus.0
```

  – Container corresponds to a PCI host (PHB)
  – VFIO gets a separate (from emulated PCI) PHB:
    - multiple PHBs are easy on sPAPR
    - VFIO's table contains host addresses, emulated PCI contains guest addresses => we want simpler cleanup

PCI support in QEMU

	x86	sPAPR
config space	emulated PIIX	hypercalls
interrupts	emulated APIC	hypercalls
BAR	mmap or emulation	mmap or emulation
DMA	?	?

Note: big/little endian on host/guest

DMA for guest

- x86: no guest visible IOMMU, IOMMU domains on the host
  - entire guest RAM is mapped (pinned)
- sPAPR: guest-visible IOMMU
  - map/unmap done via hypercalls:
    - H_PUT_TCE(LIOBN, IO_addr, GPA), 1 page per call
    - H_STUFF_TCE(LIOBN, IO_addr, tce_num)
    - H_PUT_TCE_INDIRECT(LIOBN, IO_addr, GPA, tce_num)
      - LIOBN - Logical IO Bus number
  - DMA windows advertised in device tree
    - backed by IOMMU (a.k.a. TCE) table
    - default “32bit” a.k.a. “small” DMA window, 2GB, 4K pages
    - guest maps RAM page to window on demand

Dynamic DMA windows (POWER7/8)

More windows, more tables.

Hypercalls:

- query
- create
- remove
- reset

	"**Default**" 32 bit	"**Huge**" or "**DDW**" 64 bit
Bus base address *fixed in HW*	0	0800.0000.0000.0000 (60bit) *only for 64bit dma mask*
size	2GB	any, up to guest RAM size
page size	4KB	4KB 64KB 16MB ... 16GB
hypercalls rate	**a lot**, like 1..4 per ethernet packet	few, once at the boot time

TCE tables for 64GB:

Page size	Table size
4K	128**M**B (but multi-level supported)
64K	8MB
16M	32**K**B

DMA on sPAPR KVM guest

Typical DMA map/unmap on para-virtualized sPAPR guest:

- GUEST: dma_alloc()
- GUEST: hypercall (H_PUT_TCE...)
- KVM: real mode (MMU off) handler
- KVM: virtual mode (MMU on) handler
- QEMU: GPA -> UA, LIOBN -> VFIO container
- QEMU: ioctl(VFIO_container_fd, (un)map)
- HOST-VFIO: vfio_spapr_tce driver
- HOST-ARCH: UA -> HPA + update real IOMMU table

Things to take care of: 1) pin pages, 2) update locked_vm counter

Performance under KVM (no optimization)

Chelsio CXGB3, maximum rate = **1050**MB/s

`H_PUT_TCE` -> **180**MB/s

`H_PUT_TCE_INDIRECT/H_STUFF_TCE` help -> **320**MB/s

Why:

- Device drivers call `dma_alloc()` a lot
- Real mode -> Virtual mode (enabling MMU)
- Virtual mode -> User mode
- User mode -> `ioctl()` -> Virtual mode

Performance under KVM (host virtual mode)

To do (in addition to QEMU):

- GPA -> UA
  - use KVM memslots
- LIOBN -> VFIO container
  - use VFIO KVM device

Performance

`H_PUT_TCE` **180** -> **750**MB/s

`H_PUT_TCE_INDIRECT` **320** -> **850**MB/s

Performance under KVM (host real mode)

(This is what pHyp does)

All the same as in virtual mode except memory pinning

- no `pfn_to_page()/page_to_pfn()`
- no `get_user_pages_fast()`
- locked pages accounting is problematic

Solution: pin DMA memory at guest boot time

Performance:

`H_PUT_TCE` **750** -> **1050**MB/s

`H_PUT_TCE_INDIRECT` **850** -> **1050**MB/s

Conclusion

Conclusion: none :)

Plan:

- push everything to upstream

- may be become more like x86 and put everything on a single PHB

- ...

QUESTIONS?