Cooperative DMA in a memory oversubscribed environment

Being able to run memory oversubscribed virtual machines with PCI passthrough via VFIO.

James Gowans (jgowans@amazon.com)

Amazon / AWS / EC2

LPC (VFIO/IOMMU/PCI MC), September 2022

Agenda

1. Background: device passthrough and memory overcommit
2. IOMMU pgtable reremaping via mmu_notifier
3. Guest cooperation: page pinning device driver

Background, problem & requirements

`VFIO_IOMMU_MAP_DMA` pins all pages, populating IOMMU pgtables:

- Simplicity: no need to touch IOMMU again
- Correctness: no possibility of DMAR failure
- Prevents memory overcommit. :-(

Prior art:

- ATS + PRI for faults. Not plumbed into VFIO? Unsure how prevelant PRI is? Support on root port for arbitrary PCI or only Intel graphics? May take a while to get generally correct.
- SVA/SVM for pgtable sharing; does it work for all hardware? May not always want IOMMU and userspace strictly in sync, unless PRI/ATS in use.

Looking for solution for devices which *can't* take PFs: No PRI/ATS/SVM. Suggest software solution.

Hook VFIO into MMU notifiers

Use case of keeping IOMMU in sync with userspace. Alternative to SVA/SVM.

New ioctl: `VFIO_IOMMU_MAP_DMA_*UNPINNED*`: no allocation or IOMMU pgtable population.

Hook into mmu notifiers:

- `change_pte` when a userspace pgtable entry is (re)mapped.
- `invalidate_range_(start|end)` when entry is zapped.

Challenge: `change_pte` not extensively used. Currently only CoW? For IOMMU we *MUST* always notify. Need to increase coverage to when new page is populated (lazy alloc).

One user is KVM; additional uses may interfere...

Introduce IOMMU remap_pte callback

VFIO would invoke new IOMMU callback:

```
struct iommu_domain_ops {
...
      int (*remap_pte)(struct iommu_domain *domain,
                        dma_addr_t const iova,
                        phys_addr_t const pfn,
                        size_t const size);
}
```

That would walk page table and replace (or zap) entry.
Challenge: PTE size changes (eg: THP coalesce) may be tricky to handle. Hugetlbfs would still work; fixed huge size.

Pause for questions...

Next: make DMA robust with guest cooperation
Pause for questions/comments. Eg:

- Compare this against Shared Virtual Address/Memory?
- Is hooking into `change_pte` notifier sane?
- Is PTE size change (THP) a real problem?
- Would HMM be applicable here? (I don't think it's the right use case)
- Other challenges?

Guest cooperation: page pinning device driver

Problem: guest initiate DMA to non-resident page causing DMAR failure. Prior art:

- ATS + PRI, but not that prevalent.
- Expose vIOMMU. Expensive due to VM exits on remapping, lot of invalidation and shadow pgtables.

Light weight solution: guest kernel access page before DMA to ensure resident. Much lower cost; typically no VM exit.

Just an access is good enough for lazy alloc, a shared “pinned” bitmap can allow swap too. Carefully sequenced to avoid races.

Driver integration

Where should “page touching” functionality hook in?

- Expose as IOMMU? No: no DMAR, no IRQ remapping
- Register on `struct device` device DMA ops? This:

```
const struct dma_map_ops *dma_ops;
```

- Hook into `dma_direct_map_page` (and friends)? Eg:

```
void *dma_direct_alloc(struct device *dev, size_t size,
                       dma_addr_t *dma_handle, gfp_t gfp, ...)
        ...
        dma_pinning_pin(pfn, size, ...);
```

Device discovery

How should the host expose the device? Just a few MMIO registers... Discovery and handshake.

Must be probed early: before any DMA.

- Piggyback on existing device? (virtio something?)
- New ACPI table entry? Where?
  - Or device tree?

Guidance needed!

Next steps

Summary:

- "Page pinning" DMA hook (in place of PRI + ATS).
- Dynamic page tables via mmu_notifiers (in place of SVM)

Proof of concept; seems to work:

- Host: github:jgowans/linux/dynamic-vfio
- Guest: github:jgowans/linux/page-touching-dma-ops-v6

Need to figure out relation to IOMMUFD.

Send out RFC for dynamic VFIO via mmu_notifiers.

Start discussion on exposing page pinning device model.