vIOMMU/ARM: full emulation and virtio-iommu approaches

Eric Auger
KVM Forum 2017
Overview

- Goals & Terminology
- ARM IOMMU Emulation
  - QEMU Device
  - VHOST Integration
  - VFIO Integration Challenges
- VIRTIO-IOMMU
  - Overview
  - QEMU Device
  - x86 Prototype
- Epilogue
  - Performance
  - Pros/Cons
  - Next
Main Goals

• Instantiate a virtual IOMMU in ARM virt machine
  • Isolate PCIe end-points
    1) VIRTIO devices
    2) VHOST devices
    3) VFIO-PCI assigned devices
  • DPDK on guest
  • Nested virtualization
• Explore Modeling strategies
  • full emulation
  • para-virtualization
Some Terminology

- Configuration Lookup
- TLB / Page Table Walk
- Input @ + prot flags
- Translated @

**Stage 1 - guest**
- IOVA

**Stage 2 - hyp**
- GPA
- HPA
ARM IOMMU Emulation
## ARM System MMU Family Tree

<table>
<thead>
<tr>
<th>SMMU Spec</th>
<th>Highlights</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>V7 VMSA*</td>
</tr>
<tr>
<td></td>
<td>stage 2 (hyp), Register based configuration structures</td>
</tr>
<tr>
<td></td>
<td>ARMv7 4kB, 2MB, 1GB granules</td>
</tr>
<tr>
<td>v2</td>
<td>+ V8 VMSA</td>
</tr>
<tr>
<td></td>
<td>+ dual stage capable</td>
</tr>
<tr>
<td></td>
<td>+ distributed design</td>
</tr>
<tr>
<td></td>
<td>+ enhanced TLBs</td>
</tr>
<tr>
<td>v3</td>
<td>+ V8.1 VMSA</td>
</tr>
<tr>
<td></td>
<td>+ memory based configuration structures</td>
</tr>
<tr>
<td></td>
<td>+ In-memory command and event queues</td>
</tr>
<tr>
<td></td>
<td>+ PCIe ATS, PRI &amp; PASID</td>
</tr>
<tr>
<td></td>
<td>not backward-compatible with v2</td>
</tr>
</tbody>
</table>

*VMSA = Virtual Memory System Architecture*
Origin, Destination, Choice

SMMUv2
maintained
out-of-tree by Xilinx

SMMUv3
initiated by Broadcom
Interrupted Contribution
Scalability
Memory-based Cfg
Memory-based Queues
PRI & ATS

UPSTREAM

ENABLE VHOST and VFIO USE CASES
SMMUv3 Emulation Code

• Stage 1 or stage 2
• AArch64 State translation table format only
• DT & ACPI probing
• limited set of features (no PCIe ATS PASIDS PRI, no MSI, no TZ...)

<table>
<thead>
<tr>
<th>LOC</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>600</td>
<td>IOMMU memory region infra, page table walk</td>
</tr>
<tr>
<td>1600</td>
<td>MMIO, config decoding (STE, CD), IRQ, cmd/event queue</td>
</tr>
<tr>
<td>200</td>
<td>sysbus-fdt, virt, virt-acpi-build</td>
</tr>
<tr>
<td>2400</td>
<td>Total</td>
</tr>
</tbody>
</table>
Vhost Enablement

Full Details in 2016 “Vhost and VIOMMU” KVM Forum Presentation
Jason Wang (Wei Xu), Peter Xu

- Call IOMMU Notifiers on invalidation commands
- + 150 LOC
VFIO Integration : No viommu

PCIe End Point
- GPA
- PCIe Guest Topology

Guest PoV
- GPA
- Host RAM

PCIe Host Topology
- GPA
- SID#j

Physical IOMMU
- GPA
- HPA
- Host Interconnect
- Host RAM

vfio
VFIO Integration: viommu

- Userspace combines the 2 stages in 1
- VFIO needs to be notified on each cfg/translation structure update
SMMU VFIO Integration Challenges

1) Mean to force the driver to send invalidation commands for all cfg/translation structure update
   - INTEL DMAR: ✔️
   - ARM SMMU: ✗

2) Mean to invalidate more than 1 granule at a time
   - INTEL DMAR: ✔️
   - ARM SMMU: ✗

1) “Caching Mode” SMMUv3 driver option set by a FW quirk
2) Implementation defined invalidation command with addr_mask

- Shadow page tables
- Use 2 physical stages
- Use VIRTIO-IOMMU
Use 2 physical stages

- Guest owns stage 1 tables and context descriptors
  - Host does not need to be notified on each change anymore
  - Removes the need for the FW quirk
- Need to teach VFIO to use stage 2
- Still a lot to SW virtualize: Stream tables, registers, queues
- Miss an API to pass STE info
- Miss an Error Reporting API
- Related to SVM discussions ...

![Diagram of two stages](image)
VIRTIO-IOMMU
Overview

- rev 0.1 draft, April 2017, ARM
  + FW notes
  + kvm-tool example device
  + longer term vision
- rev 0.4 draft, Aug 2017
- QEMU virtio-iommu device
Device Operations

- Device is an identifier unique to the IOMMU
- An address space is a collection of mappings
- Devices attached to the same address space share mappings
- If the device exposes the feature, the driver sends probe requests on all devices attached to the IOMMU
QEMU VIRTIO-IOMMU Device

- Dynamic instantiation in ARM virt (dt mode)
- VIRTIO, VHOST, VFIO, DPDK use cases

<table>
<thead>
<tr>
<th>Component</th>
<th>LOC</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtio-iommu device</td>
<td>980</td>
<td>infra + request decoding + mapping data structures</td>
</tr>
<tr>
<td>vhost/vfio integration</td>
<td>220</td>
<td>IOMMU notifiers</td>
</tr>
<tr>
<td>machvirt dynamic instantiation</td>
<td>100</td>
<td>dt only</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>1300</strong></td>
<td></td>
</tr>
</tbody>
</table>

virtio-iommu driver: 1350 LOC
x86 Prototype

• Hacky Integration (Red Hat Virt Team, Peter Xu)
  • QEMU
    • Instantiate 1 virtio MMIO bus
    • Bypass MSI region in virtio-iommu device
  • Guest Kernel
    • Pass device mmio window via boot param (no FW handling)
    • Limited to a single virtio-iommu
    • Implement dma_map_ops in virtio-iommu driver
    • Use PCI BDF as device id
    • Remove virtio-iommu platform bus related code
Epilogue
First Performance Figures

- Netperf/iperf TCP throughput measurements between 2 machines
- Dynamic mappings only (guest feat. a single virtio-net-pci device)
- No tuning

---

ARM

Gigabyte R120, T34 (1U Server), Cavium CN88xx, 1.8 Ghz, 32 procs, 32 cores, 64 GB RAM

x86

Dell R430, Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz, 32 proc, 16 cores, 32 GB RAM
## Performance: ARM benchmarks

<table>
<thead>
<tr>
<th>Guest Config</th>
<th>netperf Rx (Mbps) vhost off / on</th>
<th>netperf Tx (Mbps) vhost off / on</th>
<th>iperf3 Rx (Mbps) vhost off / on</th>
<th>iperf3 Tx (Mbps) vhost off / on</th>
</tr>
</thead>
<tbody>
<tr>
<td>noiommu</td>
<td>4126 / 3924</td>
<td>5070 / 5011</td>
<td>4290 / 3950</td>
<td>5120 / 5160</td>
</tr>
<tr>
<td>smmuv3</td>
<td>1000 / 1410</td>
<td>238 / 232</td>
<td>955 / 1390</td>
<td>706 / 692</td>
</tr>
<tr>
<td>smmuv3,cm</td>
<td>560 / 734</td>
<td>85 / 86</td>
<td>631 / 740</td>
<td>352 / 353</td>
</tr>
<tr>
<td>virtio-iommu</td>
<td>970 / 738</td>
<td>102 / 97</td>
<td>993 / 693</td>
<td>420 / 464</td>
</tr>
</tbody>
</table>

- Low performance overall with virtual iommu, especially in Tx
  - smmuv3 performs better than virtio-iommu
    - when vhost=on
    - in Tx
  - Both perform similarly in Rx when vhost=off
- Better performance observed on next generation ARM64 server
  - Max Rx/Tx with smmuv3: 2800 Mbps/887 Mbps (42%/11% of noiommu cfg)
  - Same perf ratio between smmuv3 and virtio-iommu
## Performance: x86 benchmarks

<table>
<thead>
<tr>
<th>Guest Config (vhost=off)</th>
<th>netperf</th>
<th>iperf3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rx (Mbps)</td>
<td>Tx (Mbps)</td>
</tr>
<tr>
<td>noiommu</td>
<td>9245 (100%)</td>
<td>9404 (100%)</td>
</tr>
<tr>
<td>vt-d (deferred invalidation)</td>
<td>7473 (81%)</td>
<td>9360 (100%)</td>
</tr>
<tr>
<td>vt-d (strict)</td>
<td>3058 (33%)</td>
<td>2100 (22%)</td>
</tr>
<tr>
<td>vt-d (strict + caching mode)</td>
<td>2180 (24%)</td>
<td>1179 (13%)</td>
</tr>
<tr>
<td>virtio-iommu</td>
<td>924 (10%)</td>
<td>464 (5%)</td>
</tr>
</tbody>
</table>

- Indicative but not fair
  - virtio-iommu driver does not implement any optimization yet
  - Behaves like vtd strict + caching mode
- Looming Optimizations:
  - Deferred IOTLB invalidation
  - Page Sharing avoids explicit mappings
  - QEMU device IOTLB emulation
  - vhost-iommu
## Some Pros & Cons

<table>
<thead>
<tr>
<th>vSMMUv3</th>
<th>virtio-iommu</th>
</tr>
</thead>
<tbody>
<tr>
<td>++ unmodified guest</td>
<td>++ generic/ reusable on different archs</td>
</tr>
<tr>
<td>++ smmv3 driver reuse (good maturity)</td>
<td>++ extensible API to support high end features &amp; query host properties</td>
</tr>
<tr>
<td>++ better perf in virtio/vhost</td>
<td>++ vhost allows in-kernel emulation</td>
</tr>
<tr>
<td>+ plug &amp; play FW probing</td>
<td>+ simpler QEMU device, simpler driver</td>
</tr>
<tr>
<td>- QEMU device is more complex and incomplete</td>
<td>- virtio-mmio based</td>
</tr>
<tr>
<td>-- ARM SMMU Model specific</td>
<td>- virtio-iommu device will include some arch specific hooks</td>
</tr>
<tr>
<td>-- Some key enablers are missing in the HW spec for VFIO integration: only for virtio/vhost</td>
<td>- mapping structures duplicated in host &amp; guest</td>
</tr>
<tr>
<td></td>
<td>--para-virt (issues with non Linux OSes)</td>
</tr>
<tr>
<td></td>
<td>-- OASIS and ACPI specification efforts (IORT vs. AMD IVRS, DMAR and sub-tables)</td>
</tr>
<tr>
<td></td>
<td>-- Driver upstream effort (low maturity)</td>
</tr>
<tr>
<td></td>
<td>-- explicit map brings overhead in virtio/vhost use case</td>
</tr>
</tbody>
</table>
Next

- vSMMUv3 & virtio-iommu now support standard use cases
  - Please test & report bug/performance issues
- virtio-iommu spec/ACPI proposal review
  - Discuss new extensions
  - Follow-up SVM and guest fault injection related work
- Code Review
- Implement various optimization strategies
THANK YOU