KVM Platform Device Passthrough

Eric Auger, Linaro
KVM Forum
Oct 14, 2014
Goal: efficiently assign platform devices to KVM guests
Agenda

- VFIO Framework
- Focus on IRQ assignment
  - Understand legacy frameworks
  - Why hardware-assisted IRQ forwarding is crucial?
- Forwarded IRQ Integration with KVM/VFIO
- Experimental Results
VFIO Platform Driver

- allows user-side to
  - mmap device MMIO regions
  - route physical IRQ to eventfd
  - Dma map buffers on iommu
QEMU VFIO device

- Setup routes between guest and assigned device
  - MMU
  - IOMMU
  - IRQ injection path

- Generate guest device device tree node
MMIO & IRQ Paths

Host

Qemu

Xgmac QEMU VFIO DEVICE

vfio driver

iommu driver

KVM/irqfd

Physical IRQ

Guest

Xgmac driver

EL0

EL1

EL2

Physical IRQ

IRQ injection

IRQ completion

direct MMIO access

IO platform device

IOMMU
ARM IRQ Handling

Level Sensitive

IRQ line request

Inactive  Pending  Active & pending  Active  Active & pending  Pending

Ack

Device IRQ status reset

ISR

Deactivate

Edge Sensitive

IRQ line request

Inactive  Pending  Active  Active & pending  Pending

Ack

ISR

Deactivate
Assigned Level Sensitive IRQ Model
Level Sensitive IRQ
Implementation Challenges

1) Physical IRQ completion
2) Virtual IRQ modeling
3) Virtual IRQ completion propagation
Basic vfio/irqfd ARM porting

- VFIO Mask/unmask
- Trap on completion
Performance Challenges on ARM

- 1 VM switch when injecting
- 1 VM switch when completing
- VM Switch really costly on ARM

Goal: Propose a new method to save completion VM switch using ARM GIC virtualization features
GIC Forwarding Feature

- GIC can automatically complete physical IRQ on virtual IRQ completion

- Host only drops the running priority of the CPU I/F to allow other physical IRQs to be signaled

- Same IRQ cannot be signaled before its deactivation by GIC HW
Forwarded IRQ Patch

- “ARM: forwarding physical interrupts to a guest VM” from M. Zyngier
  - Enable mode where priority drop and deactivate are separated, Linux wide
    - Current used mode is simultaneous prio drop & deactivate
  - Provides separate operations to program IRQ forwarding at
    - IRQCHIP
    - VGIC
vfio/irqfd/forward

- No mask/unmask anymore
- Guest completion propagated by GIC HW
- No VM switch at completion
- Natural and optimized implementation
IRQ Path with KVM (irqfd/forward)
Forwarded IRQ Integration

Allow userspace to configure forwarding of a VFIO device IRQ

VFIO_DEVICE_SET_IRQS
(irq_index, eventfd)

KVM_SET_DEVICE_ATTR
(vfio fd, irq index, gsi)

KVM_IRQFD
(eventfd, gsi)

VFIO platform driver

is_forwarded?

KVM-VFIO device

get hwirq from irq index

IRQCHIP

GIC

Set_forwarded()
Performance Measures

- Calxeda Midway
  - Communication between 2 nodes
  - 1Gb/s switch

- 2 xgmacs
  - eth0 assigned to host
  - eth1 assigned to guest if any

- Versions:
  - All kernels are 3.17rc3
  - QEMU is 2.1.0
Comparison

- **Native Performance**

  Node 1
  
  host 10.5.3.7
  
  netperf client, IRQ count

  Eth0

  TCP_STREAM (Tx)
  TCP_MAERTS (Rx)
  TCP_RR (RTT)

  Eth0

  host 10.5.3.10
  
  netserver

  Node 2

- **Guest Performance**

  host 10.5.3.7

  Eth0

  guest 10.5.3.8
  
  netperf client, IRQ count

  Eth1

  host 10.5.3.10
  
  netserver

  Eth0
Round Trip Time

TCP_RR

_native_perf_ 18350 trans/s

no irqfd 13135 13439

virtio-net
Xgmac IRQ rate on guest (IRQ/s)

![Bar chart showing IRQ rates in percentage vs native for different conditions.]

- irqfd: 19025, 24179, 26952, 36413, 26223, 27051
- forward: 41901, 115162, 36413, 36700

Legend:
- Tx
- Rx
- RR
Throughput with 3 TCP/IP patterns

<table>
<thead>
<tr>
<th>Native BW (Mb/s)</th>
<th>Tx</th>
<th>Rx</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>669</td>
<td>623</td>
</tr>
<tr>
<td>file</td>
<td>449</td>
<td>578</td>
</tr>
<tr>
<td>bulk</td>
<td>933</td>
<td>933</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Buffer Size Options</th>
<th>default</th>
<th>file</th>
<th>bulk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local Tx &amp; Rx socket buffer size (-s)</td>
<td>8kB</td>
<td>8kB</td>
<td>64kB</td>
</tr>
<tr>
<td>Remote Tx &amp; Rx socket buffer size (-S)</td>
<td>8kB</td>
<td>8kB</td>
<td>64kB</td>
</tr>
<tr>
<td>Local send size (-m)</td>
<td>8kB</td>
<td>4kB</td>
<td>8kB</td>
</tr>
<tr>
<td>Remote received size (-M)</td>
<td>8kB</td>
<td>4kB</td>
<td>8kB</td>
</tr>
</tbody>
</table>
Status & Next
# QEMU patches & dependencies

<table>
<thead>
<tr>
<th>#</th>
<th>QEMU Patches</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>KVM platform device passthrough</td>
<td>E. Auger</td>
</tr>
<tr>
<td>1</td>
<td>Dynamic sysbus device allocation support</td>
<td>A. Graf</td>
</tr>
<tr>
<td>2</td>
<td>machvirt dynamic sysbus device instantiation</td>
<td>E. Auger</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>Kernel Patches</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>VFIO support for platform devices</td>
<td>A. Motakis</td>
</tr>
<tr>
<td>1</td>
<td>ARM: KVM: add irqfd support</td>
<td>E. Auger</td>
</tr>
<tr>
<td>2</td>
<td>KVM-VFIO IRQ forward control</td>
<td>E. Auger</td>
</tr>
<tr>
<td>3</td>
<td>ARM: Forwarding physical interrupts to a guest VM</td>
<td>M. Zyngier</td>
</tr>
</tbody>
</table>
Conclusion

• Main functional bricks are available for efficient KVM platform device passthrough

• Forwarded IRQ usage shows improvements on
  - Sustained IRQ rate
  - Latency
  - Bandwidth, on some patterns

• Please test and use VFIO platform
  - Start integrating your devices
  - Share issues with complex device tree nodes
  - Work ongoing on AArch64 too