Paravirtualized DMA using IOMMU emulation

Joerg Roedel | August 9th, 2010
Overview

- State of KVM Device Passthrough
- IOMMU architectures
- The Idea: A Paravirtual IOMMU
- Guest Kernel Changes
State of KVM Device Passthrough

- KVM allows DMA for a passthrough device
- Device can access any guest physical address
- Implemented using newer hardware IOMMUs
- Problem: PCI DMA has no concept of demand paging
  - Requires that all guest memory is mapped (and pinned) in advance
  - Prevents that any guest memory is moved or swapped out
- This talk is about an idea that limits the amount of guest ram to pin
IOMMU architectures
State-of-the-Art IOMMU Architecture

- Modern IOMMUs have a large feature set
  - Can differentiate between devices
  - Support remapping of full 64-bit address space
  - Protect main memory by assigning access permissions to DMA mappings
- Support many use-cases
  - DMA remapping for legacy devices
  - Protection against misbehaving devices
  - Virtualization
- Difficult to emulate
State-of-the-Art IOMMU architecture example: AMD-Vi

Diagram showing the IOMMU architecture with DTEs connected to Device Table, and branches leading to Network, Disc, and GFX.
Legacy IOMMUs

- Support only one scenario:
  - DMA remapping for legacy devices
- Cannot differentiate between all devices
- Different flavors available
- All use the same principles
  - Provide an aperture
  - Mapping is implemented by a single-level linear page table
Legacy IOMMU Example: AMD GART

- Basic principle: Remaps physical memory
- Implements an address range, which is remapped: The Aperture
  - Remapping destination configured in a page table
  - Access outside the aperture range is not remapped
- TLB can only be flushed completely
- Can be emulated with an State-of-the-Art IOMMU
The Idea: A Paravirtual IOMMU
Basic Idea for Paravirtualized DMA

- Idea: Emulate a legacy-type IOMMU
- Emulate an IOMMU with the following properties
  - Fixed size aperture for exclusive DMA access
  - Passthrough device can only access the aperture address range
- Guest cannot pin more memory than provided in the aperture
- Could be implemented even with a legacy IOMMU on the host side (when trusting the guest)
- Two variants to evaluate
First Variant

- Use guest RAM for the aperture
- Passthrough device can DMA into the aperture
- Kernel bounce buffers the data to/from the destination buffer

Pros:
- No hypercalls needed for mapping/unmapping
- Could reuse SWIOTLB code with minor modifications

Cons:
- Guest could not use the aperture RAM otherwise
- Bounce buffering expensive
First Variant - Illustrated

- Device
- DMA
- Aperture
- Guest RAM
- Buffer
- CPU
Second Variant

- Guest provides a remapping table for the aperture
- TLB range flush hypercall to sync guest remapping table with host IOMMU page table
- Host IOMMU is used to emulate the paravirtual IOMMU

Pros:
- Offloads remapping from software to hardware (no bounce buffering)
- Can reuse most parts of the GART code
- Guest does not need to reserve aperture range in RAM

Cons:
- Requires TLB-sync hypercall for every mapping operation
- More difficult to implement than first variant
Guest Kernel Changes
Guest Kernel Changes

- Infrastructure needed in the guest kernel:
  - Per-device DMA-ops
  - Paravirtual IOMMU driver
- Per-device DMA-ops already available in Linux
- A paravirtual IOMMU driver for Linux could reuse existing code in the kernel
  - SWIOTLB for first variant
  - PCI-GART for second variant
- Guest kernel changes not expected to be very intrusive
Thanks for listening!

Trademark Attribution

AMD, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names used in this presentation are for identification purposes only and may be trademarks of their respective owners.

©2009 Advanced Micro Devices, Inc. All rights reserved.