Shared Virtual Memory in KVM

Yi Liu (yi.l.liu@intel.com)
Senior Software Engineer
Intel Corporation
Legal Disclaimer

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting www.intel.com/design/literature.htm.

Intel and the Intel logo are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others

© Intel Corporation.
Shared Virtual Memory (SVM)

- CPUs
  - MMU
  - Memory
- Root Complex
  - IOMMU
- Non-SVM capable devices (E.g.: Legacy Devices)
- Discrete Devices (E.g.: PCI-E attached devices)
- Integrated Devices (E.g.: Processor Graphics)
- OS Managed
- CPU page tables
- IOMMU page tables
- CPU
- Device
- VA
- IOVA
- PA
- Host/Physical Memory

Non-SVM capable devices
Discrete Devices
Integrated Devices
Shared Virtual Memory (SVM)

- CPUs
- MMU
- Root Complex
- IOMMU
  - Non-SVM capable devices (E.g.: Legacy Devices)
  - Discrete Devices (E.g.: PCI-E attached devices)
  - Integrated Devices (E.g.: Processor Graphics)

- CPU
- Device
  - VA
  - PA
  - CPU page tables

Host/Physical Memory

Called Shared Virtual Addressing in Linux Community
Shared Virtual Memory (SVM)

CPUs

Root Complex

IOMMU

Non-SVM capable devices
(E.g.: Legacy Devices)

Discrete Devices
(E.g.: PCI-E attached devices)

Integrated Devices
(E.g.: Processor Graphics)

memory

CPU

Device

CPU page tables

OS Managed

CPU page tables

VMM Managed

EPT tables

VT-d tables

Host/Physical Memory

GVA

GPA

HPA

GVA

GPA

HPA
SVM on Intel® VT-d

• Process Address Space ID (PASID)
  – Identify process address space

• First-level translation
  – DMA requests with PASID
  – For SVM transaction from endpoint device

• Second-level translation
  – DMA requests without PASID
  – For normal DMA transaction from endpoint device

• Translation Types
  – First-Level translation
  – Second-Level translation
  – Nested translation
  – Pass-Through (address translation bypassed)

• Intel® VT-d 3.0 introduced Scalable Mode
  – SVM can be used together with Intel® Scalable I/O Virtualization
SVM on Intel® VT-d (Cont.)

• Nested Translation
  – Use both first-level and second-level for address translation
  – Enable SVM in virtualization environment
    • First-level: GVA->GPA
    • Second-level: GPA->HPA

Most vendor supports nested translation for SVM usage in Virtual Machine
Enable SVM in VM

• Need a virtual IOMMU with SVM capability
  – Proper emulation according to IOMMU spec (e.g. Intel® VT-d specification)
    • either fully-emulated or virtio-based IOMMU

• Notification for guest translation structure changes
  – Notification mechanism is vendor specific
  – For Intel® VT-d
    • “caching-mode”: explicit cache invalidation is required for any translation structure change in software

• Enable nested translation on physical IOMMU for given PASID
Enable SVM in VM (Cont.)

SVM in VM based on Intel® VT-d
Enable SVM in VM (Cont.)

SVM in VM based on Intel® VT-d
Enable SVM in VM (Cont.)

**Guest VT-d**
- Translation structure
- PASID Table
- 1st level translation (i.e. CPU page tables)
- RTAR

**Host VT-d**
- Translation structure
- PASID Table
- 2nd level translation (i.e. VT-d tables)
- RTAR

Save guest cpu page table pointer to host

In nested translation, hardware treats 1st-level page table pointer as GPA

SVM in VM based on Intel® VT-d
Enable SVM in VM (Cont.)

SVM in VM based on Intel® VT-d
Shared Virtual Memory in KVM

- **Qemu**
  - vlOMMU emulation is in Qemu

- **VFIO: Virtual Function I/O**
  - Program host IOMMU via VFIO

- **IOMMU driver**
  - New APIs exposed to VFIO for guest SVM
Shared Virtual Memory in KVM

- Bind PASID to guest CPU page table
Shared Virtual Memory in KVM

- Bind PASID to guest CPU page table
- Forward guest CPU page table cache invalidation to host
Shared Virtual Memory in KVM

- Bind PASID to guest CPU page table
- Forward guest CPU page table cache invalidation to host
- Page fault reporting and servicing
Shared Virtual Memory in KVM

- Bind PASID to guest CPU page table
- Forward guest CPU page table cache invalidation to host
- Page fault reporting and servicing
Shared Virtual Memory in KVM

- Bind PASID to guest CPU page table
- Forward guest CPU page table cache invalidation to host
- Page fault reporting and servicing

Neutral Kernel APIs for both emulated and virtio-based vIOMMUs
Bind PASID

- New VFIO IOCTL supporting multiple binding types:
  - VFIO_IOMMU_BIND_PROCESS
    - Binding to host CPU page table
  - VFIO_IOMMU_BIND_PGTBL
    - Binding to guest CPU page table
  - VFIO_IOMMU_BIND_PASID_TBL
    - Binding to guest PASID Table

- New IOMMU API to configure physical IOMMU
  - Need compatibility check of the table format
Forward Cache Invalidation to Host

- **Invalidation types**
  - IOMMU_INV_TYPE_TLB
    - IOTLB and paging structure-caches

- **Granularity conversion**
  - Supported granularities
    - Domain selective flush
    - PASID selective flush
    - Page selective flush
  - Avoid unnecessary flush
    - Guest global flush -> either domain/pasid selective flush in host

- **Use host Identities**
  - RID, Domain ID, PASID

Diagram:
- vIOMMU Driver
- Qemu
- VFIO
- pIOMMU Driver

Flow:
- iommu tlb flushing
- New VFIO IOCTL (VFIO_IOMMU_INVALIDATION)
- iommu_sva_invalidate(…)
- Submit Invalidation to HW
Page Fault Handling

- **PCI Express® Address Translation Service**
  - PRI: page request interface
  - Page Response (PRS)

![Diagram of Page Fault Handling]

- **PCIe (PRS/ATS)**
  - Dev IOTLB
  - PRI
  - PCIE Dev DMA

- **Dev-IOTLB missing Translation request**
  - Translation fault
  - Page request

- **IOMMU**
  - IOTLB
  - PRQ event
  - PRQ handling
  - Page response

- **Page response**
  - Translated request
Page Fault Handling (Cont.)

- **Report PRQ to Guest**
  - Page Request Capability in vIOMMU

- **Forward guest page response to host**
IOMMU Fault Reporting Framework

• Newly defined “struct iommu_fault_param”, added to “struct device”

/**
 * struct iommu_fault_param - Per device generic IOMMU runtime data
 * @dev_fault_handler: Callback function to handle IOMMU faults at device level
 * @data: handler private data
 * @faults: holds the pending faults which needs response, e.g. page response.
 * @timer: track page request pending time limit
 * @lock: protect pending PRQ event list
 */
struct iommu_fault_param {
    iommu_dev_fault_handler_t handler;
    struct list_head faults;
    struct timer_list timer;
    struct mutex lock;
    void *data;
};
IOMMU Fault Reporting Framework

1. IOMMU fault handler registration
   - iommu_register_device_fault_handler()
   - iommu_unregister_device_fault_handler()

2. In-kernel device driver and vfio driver registers its own fault handler
   - Vfio fault handler should further notify Qemu or other user-space application

3. The original idea was brought up by David Woodhouse
   - May refer to more detail https://lwn.net/Articles/608914/
Upstream Status (Kernel)

- IOMMU/VFIO extension for virtual SVA support (Jacob Pan/Yi Liu, Intel)
  - Earliest RFC patch for vSVA support

- SVA native enabling on ARM platform (Jean-Philippe Brucker, ARM)

- Shared requirements in the two tracks
  - binding PASID, fault reporting

SVA stands for Shared Virtual Addressing in Linux community
Qemu vSVA enabling has two parts (Yi Liu, Intel)

- vIOMMU emulation
  - Earliest RFC patch for vSVA back to 2017-April

- Notification framework between vIOMMU emulator and VFIO within Qemu
  - Notifier framework in v3, with community comments addressed
• Shared Virtual Memory (SVM) enables efficient workload submission by directly programming CPU virtual addresses on the device

• Intel® VT-d 3.0 specification extends SVM usage together with Intel® Scalable I/O Virtualization

• Holistic enhancements are introduced cross multiple kernel/user space components, to enable SVM virtualization in KVM

• New kernel APIs are kept neutral to support all kinds of virtual IOMMUs (either emulated or para-virtualized)
LINUX CON

containercon

CLOUD OPEN

CHINA 中国

THINK OPEN

开放性思维
Backup
Intel® VT-d ECS (deprecated)
Key Difference: PASID is a global ID space shared by all VMs. ALL page-table pointers moved to PASID Granular table.