Share Virtual Address

Yisheng Xie/ xieyisheng1@huawei.com
Bob Liu / liubo95@Huawei.com

HUAWEI TECHNOLOGIES CO., LTD.
Agenda

- What is SVA?
- Why SVA?
- How SVA works?
- Our works
- Upstream status
What is SVA?(cont.)

- SVA (Share Virtual Address) means device use the same virtual address with CPU which get the same thing. But they may have different achievement in HW and SW.
  - DMAR (IOMMU, SMMU) or MMU
  - Use the same page table or not
  - Same physical address or not
  - Support zero copy or not
  - Support IO-Page Fault or not
What is SVA?

- Intel - SVM (Share Virtual Memory):
  - CPU access DDR through MMU
  - Device access DDR through IOMMU
  - MMU share the same page table with IOMMU
  - (used by AMD’s Secure Virtual Machine in Linux)
What is SVA?(cont.)

- **Nvidia - UVA (Unify Virtual address):**
  - GPU has its own MMU
  - CPU can only access DDR while GPU can only access HBM
  - GPU MMU mirror the Page Table of CPU
  - When GPU access a VA not populated in HBM it will copy the data from DDR, and vice versa.
What is SVA?(cont.)

- ARM - SVA (share virtual address space):
  - Use SMMU instead of IOMMU
  - Device may also have DDR memory
    - With the help of CCIX, both CPU and Device can access DDR and HBM in Cache Coherent way
  - (Device memory management is another story)
Why SVA?

- Simplify user program
  - Share objects is hard to achieve for complex data-set (list, tree, ...)
  - Zero-copy is easier to achieve (device can access DDR with cc)
Why SVA? (cont.)

- Device side on-demand paging (based on I0 page fault)
  - Pros:
    - No need to pin memory - decrease pin memory overhead
    - Allow memory overcommit - usefully for low-end production
  - Cons:
    - Performance overhead of page fault for high speed device

1. `Malloc()` buffer, syscall or ioctl to kernel
2. `Kmalloc()` buffer request device initial dma
3. Device DMAs to/from kernel buffer
4. Kernel copies to user buffer
   - Device DMA to user buffer(pin....)

---

1. `Malloc()` buffer, syscall or ioctl to kernel
2. Request device to initiate DMA
3. Device DMA access process address directly. It trigger interrupt to CPU when access non-populated Virtual Address.
How SVA works?

- **ARM/SMMUv3**: Bind a mm to device to share page table
  - `arm_smmu_mm`: address space abstraction in smmu
  - `arm_smmu_cd`: smmu context descriptor instant
    - When task is bound to device, its values are got from task

![Diagram of SVA](image)
How SVA works? (cont.)

- ARM/IO Pagefault: PRI Queue vs Event Queue
  - PCI devices use PRI Queue/PASID
  - Platform devices use Event Queue/SSID (Stall-mode)
Our Works

- SVA in private mode
  - The main works add SVA support for device which do not support fault, for our device do not support IO page fault.
    - Device driver need pin the memory used by device
    - No fault mode: device use the same page table with process.
    - Private mode: device use a different page table which named io-pageable.
Our Works (cont.)

- SVA for WrapDrive
  - PASID is used for multi-queue under multi-process.
Our Works (cont.)

- A SVA implement case:
  - M-COREs and Super COREs in on-chip device
  - S-COREs use SMMUv3 whichs works in stall mode
  - M-COREs use MMU instead
Our Works (cont.)

**User Space**  |  **Kernel Space**  |  **Hardware/ Accelerator**
--- | --- | ---
USER  | MM  | SVA Driver  | IOMMU/SMMU Driver  | Task Scheduler  | S-CORE  | M-CORE

1. malloc/mmap
2. page fault: alloc PA and map Vaddr to PA
3. ioctl(fd, <pid>, TASK_BIND,...)
4. ioctl(fd, OFFLOAD_TASK, <Vaddr, PASID, ASID,TTB, TCR>)
5. free/munmap (Vaddr)

1. Get task’s mm from pid
2. Alloc sva_mm to tack the mm
3. Get ASID from task’s mm
4. Get pgd(TTB) from task’s mm
5. Get TCR from system reg

PASID; (ASID, TTB, TCR)

PASID

PASID

IO page fault from Super CORE:
Vaddr, PASID
1. get mm from PASID
2. handle_mm_fault(mm, Vaddr)

IO page fault from Normal CORE:
Vaddr, ASID
1. Get mm ASID
2. handle_mm_fault(mm, Vaddr)

reply ok
Upstream status

- SVA for SMMUv3
  - RFC1: [RFC PATCH 00/30] Add PCIe SVM support to ARM SMMUv3
    - Only works for PCIE pri
  - RFC2: [RFCv2 PATCH 00/36] Process management for IOMMU + SVM for SMMUv3
    - Add support of stall mode for platform device
  - V1/V2: [PATCH v2 00/40] Shared Virtual Addressing for the IOMMU
    - Only support devices which have IO-Page Fault ability.

- Our works is in upstream plan, which will be coming soon...
Thank you
Huawei OS Kernel Lab

Huawei Operating System R&D Department
- OS Kernel Lab
  - Linux Kernel (ARM/x86/ heterogeneous platforms) R&D and Innovation
  - R&D on a Next-generation OS kernel with Low Latency, High Security, Strong Reliability, Intelligence, etc.

Job Vacancy
Next-generation Operating System Researcher and Senior Engineer
Formal Verification Researcher and Senior Engineer
Linux Kernel Architect and Senior Engineer

Locations
Hangzhou, Beijing, Shanghai

Contact us
Tel: Mr. Wang/18658102676
Email: hr.kernel@huawei.com