Shared Virtual Addressing (SVA) with ENQCMD

Background

Shared Virtual Addressing (SVA) allows the processor and device to use the same virtual addresses avoiding the need for software to translate virtual addresses to physical addresses. SVA is what PCIe calls Shared Virtual Memory (SVM).

In addition to the convenience of using application virtual addresses by the device, it also doesn't require pinning pages for DMA. PCIe Address Translation Services (ATS) along with Page Request Interface (PRI) allow devices to function much the same way as the CPU handling application page-faults. For more information please refer to the PCIe specification Chapter 10: ATS Specification.

Use of SVA requires IOMMU support in the platform. IOMMU is also required to support the PCIe features ATS and PRI. ATS allows devices to cache translations for virtual addresses. The IOMMU driver uses the mmu_notifier() support to keep the device TLB cache and the CPU cache in sync. When an ATS lookup fails for a virtual address, the device should use the PRI in order to request the virtual address to be paged into the CPU page tables. The device must use ATS again in order the fetch the translation before use.

Shared Hardware Workqueues

Unlike Single Root I/O Virtualization (SR-IOV), Scalable IOV (SIOV) permits the use of Shared Work Queues (SWQ) by both applications and Virtual Machines (VM's). This allows better hardware utilization vs. hard partitioning resources that could result in under utilization. In order to allow the hardware to distinguish the context for which work is being executed in the hardware by SWQ interface, SIOV uses Process Address Space ID (PASID), which is a 20-bit number defined by the PCIe SIG.

PASID value is encoded in all transactions from the device. This allows the IOMMU to track I/O on a per-PASID granularity in addition to using the PCIe Resource Identifier (RID) which is the Bus/Device/Function.

ENQCMD

ENQCMD is a new instruction on Intel platforms that atomically submits a work descriptor to a device. The descriptor includes the operation to be performed, virtual addresses of all parameters, virtual address of a completion record, and the PASID (process address space ID) of the current process.

ENQCMD works with non-posted semantics and carries a status back if the command was accepted by hardware. This allows the submitter to know if the submission needs to be retried or other device specific mechanisms to implement fairness or ensure forward progress should be provided.

ENQCMD is the glue that ensures applications can directly submit commands to the hardware and also permits hardware to be aware of application context to perform I/O operations via use of PASID.

Process Address Space Tagging

A new thread-scoped MSR (IA32_PASID) provides the connection between user processes and the rest of the hardware. When an application first accesses an SVA-capable device, this MSR is initialized with a newly allocated PASID. The driver for the device calls an IOMMU-specific API that sets up the routing for DMA and page-requests.

For example, the Intel Data Streaming Accelerator (DSA) uses iommu_sva_bind_device(), which will do the following:

- Allocate the PASID, and program the process page-table (%cr3 register) in the PASID context entries.
- Register for mmu_notifier() to track any page-table invalidations to keep the device TLB in sync. For example, when a page-table entry is invalidated, the IOMMU propagates the invalidation to the device TLB. This will force any future access by the device to this virtual address to participate in ATS. If the IOMMU responds with proper response that a page is not present, the device would request the page to be paged in via the PCIe PRI protocol before performing I/O.

This MSR is managed with the XSAVE feature set as "supervisor state" to ensure the MSR is updated during context switch.

PASID Management

The kernel must allocate a PASID on behalf of each process which will use ENQCMD and program it into the new MSR to communicate the process identity to platform hardware. ENQCMD uses the PASID stored in this MSR to tag requests from this process. When a user submits a work descriptor to a device using the ENQCMD instruction, the PASID field in the descriptor is auto-filled with the value from MSR_IA32_PASID. Requests for DMA from the device are also tagged with the same PASID. The platform IOMMU uses the PASID in the transaction to perform address translation. The IOMMU APIs setup the corresponding PASID entry in IOMMU with the process address used by the CPU (e.g. %cr3 register in x86).

The MSR must be configured on each logical CPU before any application thread can interact with a device. Threads that belong to the same process share the same page tables, thus the same MSR value.

PASID Life Cycle Management

PASID is initialized as INVALID_IOASID (-1) when a process is created.

Only processes that access SVA-capable devices need to have a PASID allocated. This allocation happens when a process opens/binds an SVA-capable device but finds no PASID for this process. Subsequent binds of the same, or other devices will share the same PASID.

Although the PASID is allocated to the process by opening a device, it is not active in any of the threads of that process. It's loaded to the IA32_PASID MSR lazily when a thread tries to submit a work descriptor to a device using the ENQCMD.

That first access will trigger a #GP fault because the IA32_PASID MSR has not been initialized with the PASID value assigned to the process when the device was opened. The Linux #GP handler notes that a PASID has been allocated for the process, and so initializes the IA32_PASID MSR and returns so that the ENQCMD instruction is re-executed.

On fork(2) or exec(2) the PASID is removed from the process as it no longer has the same address space that it had when the device was opened.

On clone(2) the new task shares the same address space, so will be able to use the PASID allocated to the process. The IA32_PASID is not preemptively initialized as the PASID value might not be allocated yet or the kernel does not know whether this thread is going to access the device and the cleared IA32_PASID MSR reduces context switch overhead by xstate init optimization. Since #GP faults have to be handled on any threads that were created before the PASID was assigned to the mm of the process, newly created threads might as well be treated in a consistent way.

Due to complexity of freeing the PASID and clearing all IA32_PASID MSRs in all threads in unbind, free the PASID lazily only on mm exit.

If a process does a close(2) of the device file descriptor and munmap(2) of the device MMIO portal, then the driver will unbind the device. The PASID is still marked VALID in the PASID_MSR for any threads in the process that accessed the device. But this is harmless as without the MMIO portal they cannot submit new work to the device.

Relationships

- Each process has many threads, but only one PASID.
- Devices have a limited number (~10's to 1000's) of hardware workqueues. The device driver manages allocating hardware workqueues.
- A single mmap() maps a single hardware workqueue as a "portal" and each portal maps down to a single workqueue.
- For each device with which a process interacts, there must be one or more mmap()'d portals.
- Many threads within a process can share a single portal to access a single device.
- Multiple processes can separately mmap() the same portal, in which case they still share one device hardware workqueue.
- The single process-wide PASID is used by all threads to interact with all devices. There is not, for instance, a PASID for each thread or each thread<->device pair.

FAQ

- What is SVA/SVM?

Shared Virtual Addressing (SVA) permits I/O hardware and the processor to work in the same address space, i.e., to share it. Some call it Shared Virtual Memory (SVM), but Linux community wanted to avoid confusing it with POSIX Shared Memory and Secure Virtual Machines which were terms already in circulation.

- What is a PASID?

A Process Address Space ID (PASID) is a PCIe-defined Transaction Layer Packet (TLP) prefix. A PASID is a 20-bit number allocated and managed by the OS. PASID is included in all transactions between the platform and the device.

- How are shared workqueues different?

Traditionally, in order for userspace applications to interact with hardware, there is a separate hardware instance required per process. For example, consider doorbells as a mechanism of informing hardware about work to process. Each doorbell is required to be spaced 4k (or page-size) apart for process isolation. This requires hardware to provision that space and reserve it in MMIO. This doesn't scale as the number of threads becomes quite large. The hardware also manages the queue depth for Shared Work Queues (SWQ), and consumers don't need to track queue depth. If there is no space to accept a command, the device will return an error indicating retry.

A user should check Deferrable Memory Write (DMWr) capability on the device and only submits ENQCMD when the device supports it. In the new DMWr PCIe terminology, devices need to support DMWr completer capability. In addition, it requires all switch ports to support DMWr routing and must be enabled by the PCIe subsystem, much like how PCIe atomic operations are managed for instance.

SWQ allows hardware to provision just a single address in the device. When used with ENQCMD to submit work, the device can

distinguish the process submitting the work since it will include the PASID assigned to that process. This helps the device scale to a large number of processes.

- Is this the same as a user space device driver?

Communicating with the device via the shared workqueue is much simpler than a full blown user space driver. The kernel driver does all the initialization of the hardware. User space only needs to worry about submitting work and processing completions.

- Is this the same as SR-IOV?

Single Root I/O Virtualization (SR-IOV) focuses on providing independent hardware interfaces for virtualizing hardware. Hence, it's required to be almost fully functional interface to software supporting the traditional BARs, space for interrupts via MSI-X, its own register layout. Virtual Functions (VFs) are assisted by the Physical Function (PF) driver.

Scalable I/O Virtualization builds on the PASID concept to create device instances for virtualization. SIOV requires host software to assist in creating virtual devices; each virtual device is represented by a PASID along with the bus/device/function of the device. This allows device hardware to optimize device resource creation and can grow dynamically on demand. SR-IOV creation and management is very static in nature. Consult references below for more details.

- Why not just create a virtual function for each app?

Creating PCIe SR-IOV type Virtual Functions (VF) is expensive. VFs require duplicated hardware for PCI config space and interrupts such as MSI-X. Resources such as interrupts have to be hard partitioned between VFs at creation time, and cannot scale dynamically on demand. The VFs are not completely independent from the Physical Function (PF). Most VFs require some communication and assistance from the PF driver. SIOV, in contrast, creates a software-defined device where all the configuration and control aspects are mediated via the slow path. The work submission and completion happen without any mediation.

- Does this support virtualization?

ENQCMD can be used from within a guest VM. In these cases, the VMM helps with setting up a translation table to translate from Guest PASID to Host PASID. Please consult the ENQCMD instruction set reference for more details.

- Does memory need to be pinned?

When devices support SVA along with platform hardware such as IOMMU supporting such devices, there is no need to pin memory for DMA purposes. Devices that support SVA also support other PCIe features that remove the pinning requirement for memory.

Device TLB support - Device requests the IOMMU to lookup an address before use via Address Translation Service (ATS) requests. If the mapping exists but there is no page allocated by the OS, IOMMU hardware returns that no mapping exists.

Device requests the virtual address to be mapped via Page Request Interface (PRI). Once the OS has successfully completed the mapping, it returns the response back to the device. The device requests again for a translation and continues.

IOMMU works with the OS in managing consistency of page-tables with the device. When removing pages, it interacts with the device to remove any device TLB entry that might have been cached before removing the mappings from the OS.

References

VT-D: https://01.org/blogs/ashokraj/2018/recent-enhancements-intel-virtualization-technology-directed-i/o-intel-vt-d

SIOV: https://01.org/blogs/2019/assignable-interfaces-intel-scalable-i/o-virtualization-linux

ENQCMD in ISE: https://software.intel.com/sites/default/files/managed/c5/15/architecture-instruction-set-extensions-programming-reference.pdf

DSA spec: https://software.intel.com/sites/default/files/341204-intel-data-streaming-accelerator-spec.pdf