Goals

Revamp the current code to support:

1. DMA request with PASID
2. SVA, i.e. sharing page tables between CPU and IOMMU, implies DMA address == KVA.
Background: DMA without SVA, No PASID
Background: DMA with SVA & PASID

- CPU TLBs
- MMU
- IOMMU
- IOTLBs
- Physical Memory/Address
- CPU Page Table
- Shared (per PASID-mm)
Treat in-kernel SVA as a special case of user SVA

- Met both goals (DMA w/ PASID & SVA)
- Use the SVA lib APIs to bind init_mm.pgd to a special system PASID
- Minus I/O page fault handling
- Assume only kernel direct map is used and static

Problems with the current code

- Exposes the entire kernel mapping without any restrictions
- Lack of IOTLB synchronization – no mmu notifier for kernel page table updates: vmalloc & direct map alias updates
- Bypasses DMA APIs
Proposed solutions per address types

1. **Physical address** (bypass) mode. Similar to DMA direct where trusted devices can DMA pass-through IOMMU on a per PASID basis.

2. **IOVA** mode. DMA API compatible. Map a supervisor PASID the same way as the PCI requester ID (RID)

3. **KVA** mode. New map/unmap APIs. Support strict and fast sub-modes transparently based on device trustfulness.
Proposed new IOMMU APIs

```c
int iommu_dma_pasid_enable(struct device *dev, struct iommu_domain **domain, enum iommu_dma_pasid_mode mode)

int iommu_map_kva(struct iommu_domain *domain, void *cpu_addr, size_t size, int prot)

enum iommu_dma_pasid_mode {
    /* Pass-through mode, use physical address */
    IOMMU_DMA_PASID_BYPASS = 1,
    /* Compatible with DMA APIs, same mapping as DMA w/o PASID */
    IOMMU_DMA_PASID_IOVA,
    /* Use kernel direct mapping memory, page_offset_base */
    IOMMU_DMA_PASID_KVA,
};
```

https://lkml.org/lkml/2021/9/22/39
### API usage matrix

<table>
<thead>
<tr>
<th>Address</th>
<th>DMA w/ PASID</th>
<th>SVA</th>
<th>Security</th>
<th>Device Driver Usage Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>PA</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td><code>pasid = iommu_dmaPasid_enable(dev, NULL, IOMMU_DMA_PASID_BYPASS); /* Use the returning PASID and PA for work submission */</code></td>
</tr>
<tr>
<td>IOVA</td>
<td>Yes</td>
<td>No</td>
<td>Yes, best</td>
<td><code>pasid = iommu_dmaPasid_enable(dev, NULL, IOMMU_DMA_PASID_IOVA); /* Use the returned PASID and DMA API allocated IOVA for work submission */</code></td>
</tr>
<tr>
<td>KVA</td>
<td>Yes</td>
<td>Yes¹</td>
<td>Yes</td>
<td><code>pasid = iommu_dmaPasid_enable(dev, &amp;domain, IOMMU_DMA_PASID_KVA); iommu_map_kva(domain, &amp;buf, size, prot); /* Use the returned PASID and KVA to submit work */</code></td>
</tr>
</tbody>
</table>

¹ Fast mode: Shared CPU page tables for trusted devices only  
² Strict mode: IOMMU domain returned for the untrusted device to replicate KVA-HPA mapping in IOMMU page tables.
Opens in KVA fast mode

• Need to flush IOTLB following CPU page table update. Kernel MMU notifier? E.g.
  ➢ Direct map updates to avoid Write+Execute after module/eBPF load. Prevent attackers leveraging a writable DMA mapping to write code that the CPU can later execute.
  ➢ CONFIG_DEBUG_PAGEALLOC
  ➢ vmalloc large page fragmentation (https://lore.kernel.org/lkml/20210405203711.1095940-1-rick.p.edgecombe@intel.com/)

• How to determine device trustfulness? (On VT-d, ACPI secure address translation cache (SATC) and RCIEP). We Need to converge on a generic flag, is !pci_dev.untrusted sufficient?

• IOMMU aux_domain is used for KVA strict mapping with system PASID, will aux domain stay in the long term? Any replacement for sub-device isolation?

• Is limiting KVA to direct map range sufficient? PAGE_OFFSET < addr < VMALLOC_START
Q & A
Backup: KVA fast vs strict modes expanded

- **Strict mode** (untrusted devices, user input)
  - Map KVA-HPA in IOMMU page table
  - Mirror kernel page table mapping for direct map range

- **Fast mode** (trusted devices)
  - Share CPU page table
  - Only performance sanity check for direct map range
  - KVA map/unmap becomes nop mostly
Backup1: Why in-kernel SVA?

Security over DMA to PA
- DMA access is limited to kernel page tables

Performance over IOVA
- No need to do DMA map/unmap flush IOTLBs

Potentially simplified programming model
- Use KVA directly, kmalloc etc.

New Feature
- Shared WQ model such as Intel DSA can only do DMA with PASID
- Current DMA API does not support PASID (IOMMU agnostic)
Backup 2: The benefits of user SVA

Security
• DMA access is limited to user process page tables

Simplicity
• programming model involves process virtual address, no special treatment for DMA buffers, e.g. malloc just works for DMA.
• I/O page faults are handled behind the scene

Performance
• No need to do DMA map/unmap flush IOTLB every time, only on IO page fault or CPU flush