coIOMMU: A Virtual IOMMU with Cooperative DMA Buffer Tracking for Efficient Memory Management in Direct I/O

Kun Tian, Yu Zhang (presenter), Luwei Kang, Yan Zhao, Yaozu Dong

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.
Direct I/O

• The best performant I/O virtualization method, widely deployed in cloud and data centers.

• Guest directly interacts with I/O devices, eliminating the host intervention.

• Hardware IOMMU provides inter-guest protection with IOMMU page table (IOPT).
Static Pinning in Direct I/O

• Most devices do not support DMA page fault.
  ➢ DMA buffers need be pinned in the IOMMU.

• Hypervisor has no visibility of guest DMA activities.
Static Pinning in Direct I/O

- Pre-allocate and pin the entire guest memory before guest DMA starts.
  - E.g. at VM creation time.
The Problem of Static Pinning

- Much increased VM creation time
  - Up to 73x longer time observed for a VM with 128GB memory.

- Greatly reduced memory utilization
  - Prevent many memory optimizations (overcommitment, late allocation, swap, etc.).
Virtual IOMMU (vIOMMU)

• Primary purpose: intra-guest protection
  ➢ E.g. protection with virtual DMA remapping against bogus guest drivers.

• Side-effect: fine-grained pinning
  ➢ Guest uses vIOMMU to map/unmap DMA buffers.
  ➢ vIOMMU requests hypervisor to pin/unpin guest DMA buffers.

• A vIOMMU could be emulated or para-virtualized.
The Problem

- Emulation cost of established vIOMMUs could be significant!
  - E.g. 96.6% performance downgrade in memcached through 40Gbps NIC.
  - SLA violation if forcing all tenants to turn on vIOMMU.

- Aggressive optimizations may compromise security!
  - E.g. side-core emulation [8], map cache [52], etc.
The Reality

• Virtual DMA remapping is disabled in established vIOMMUs by most guest OSes.
  ➢ Users may opt in when security requirement is over performance concern.
  ➢ E.g. Linux uses ‘passthrough’ by default, leaving ‘strict’/‘lazy’ for user opt-in.

• The guest security requirement varies. E.g.
  ➢ when an untrusted device is plugged in;
  ➢ when a device is assigned to untrusted userspace.

Established vIOMMUs are not suitable as a reliable way for fine-grained pinning!
Motivation

• vIOMMU provides an architectural way for learning guest DMA buffers.

• However, mixing the requirements of protection and pinning, through the same costly DMA remapping interface, is needlessly constraining.
  - Protection is an OPTIONAL guest-side requirement.
  - Fine-grained pinning is a GENERAL host-side requirement.
Motivation

• Decouple DMA tracking and DMA remapping in vIOMMU.
Motivation

- Decouple DMA tracking and DMA remapping in vIOMMU.

Diagram:
- Intra-guest Protection
- Fined-grained Pinning

Goals:
- Orthogonal to remapping
- Low cost
- Non-intrusive
- Widely applicable
- Extensible
Cooperative DMA Buffer Tracking

• Bi-directional shared DMA buffer information
  ➢ To guest – whether a page is pinned in the IOMMU.
  ➢ To host – whether a page is mapped for DMA.

• A lightweight tracking interface for fine-grained pinning
  ➢ Minimize VM-exits when mapping DMA pages
  ➢ Eliminate VM-exits when unmapping DMA pages
  ➢ Enable flexible host memory management policies
coIOMMU Architecture

- DMA Tracking Table (DTT)
  - Hold shared DMA buffer info.

- coIOMMU driver
  - Hook to guest DMA API layer.

- coIOMMU backend
  - DMA remapping engine (remapEngine)
  - DMA tracking engine (trackEngine)
  - Page pinning manager (pManager)
coIOMMU Architecture

- **remapEngine**
  - Same as established DMA remapping interface.

- **trackEngine**
  - Holds base address of the DTT.
  - Emulates a doorbell register for notifying the host.

- **pManager**
  - Implements fine-grained pinning policy.
  - Invisible to guest.
DMA Tracking Table (DTT)

- A multi-level paging structure
  - Shared between host & guest.
  - Indexed by guest page frame numbers (GFNs).

- TU - Tracking Unit for each guest page frame number (GFN)
  - ‘M’ (mapped) – set/cleared by guest.
  - ‘P’ (pinned) - set/cleared by host.
  - ‘A’ (accessed) – set by guest, cleared by host.

- Extensible through 5 reserved bits
  - E.g. add a ‘D (dirty)’ bit to assist dirty page tracking in live migration.
Fine-grained Pinning

• Smart pinning
  - Instant pinning – pinning must be instantly done before any mapped page is used for DMA.
  - Precise notification – only notify the hypervisor for pages not pinned.
  - Speculative pinning – pManager speculatively pins frequently used pages.

• Lazy unpinning
  - Asynchronously done by pManager.
  - Only tries to unpin the pages that are no longer mapped.
  - Unpinned pages are reclaimable.
Guest Mapping Operations

**Guest DMA map**

1. **pManager**
2. **trackEngine**
3. **Guest**

- **Guest DMA map**
  - **pManager**: Sets 'mapped' flag
  - **trackEngine**: Precise notification (only when 'pinned' is 0)
  - **Guest**: Instant Pinning

**Up to 99.9992% notifications can be avoided due to DMA buffer locality!**

**Guest DMA unmap**

1. **pManager**
2. **trackEngine**
3. **Guest**

- **Guest DMA unmap**
  - **pManager**: Clears 'mapped' flag
  - **trackEngine**: No notification is required

**Up to 99.9992% notifications can be avoided due to DMA buffer locality!**
Lazy Unpinning & Speculative Pinning

For pages that are not 'mapped'?

Y

N

1. Periodically checks the 'mapped' & 'accessed' flag

2. Lazy unpinning

3. Speculative pinning

Host asynchronously manages pinning & unpinning in a separate thread.
DMA Tracking vs. DMA Remapping

• When DMA remapping is disabled by guest (the majority case).
  ➢ DMA tracking is an efficient solution to achieve fine-grained pinning.

• When DMA remapping is conditionally enabled.
  ➢ E.g. only for selective devices (e.g. untrusted), or only in specific period (e.g. when the device is assigned to userspace).
  ➢ However, hypervisor requires full visibility of guest DMA activities for the entire VM life-cycle.
  ➢ In such cases, DMA tracking helps provide a reliable way for fine-grained pinning.

• When DMA remapping is always enabled (for all devices at all times).
  ➢ DMA tracking provides a consistent tracking interface as other two categories, with negligible cost.
Implementation

• Based on KVM/QEMU.

• Extend existing virtual Intel VT-d.
  ➢ Reused the remapping logic in vIOMMU as remapEngine.
  ➢ Developed pManager and trackEngine from scratch.
  ➢ Extended guest intel-iommu driver to support DMA tracking.

• Applicable to all kinds of direct I/O usages.
  ➢ No ad-hoc changes in hardware or device drivers.

• Applicable to other OSes.
  ➢ As long as a generic DMA API layer is afforded.

• Applicable to other vIOMMUs.
  ➢ New tracking interface is vendor-agnostic and self-contained.

<table>
<thead>
<tr>
<th></th>
<th>New/Changed LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Host</strong></td>
<td><strong>QEMU</strong></td>
</tr>
<tr>
<td>trackEngine</td>
<td>131 new</td>
</tr>
<tr>
<td>pManager</td>
<td>552 new</td>
</tr>
<tr>
<td><strong>Guest</strong></td>
<td><strong>Intel VT-d driver</strong></td>
</tr>
<tr>
<td></td>
<td>832 new</td>
</tr>
<tr>
<td></td>
<td>47 changed</td>
</tr>
</tbody>
</table>

• Less than 700 LOC in QEMU.
• Less than 1000 LOC in guest.
Implementation

• Huge page mappings
  ➢ The DTT tracks guest pages in 4KB granularity.
  ➢ pManager is optimized to conduct 2MB page pinning by merging continuous guest pages.

• Sub-page mappings
  ➢ Multiple DMA buffers may co-locate in the same 4KB guest page (e.g. network packets).
  ➢ Guest coIOMMU driver tracks the mapping count of each mapped page.
Implementation

• Kernel Bypassing
  ➢ Kernel bypass APIs require userspace to pre-register a trunk of memory.
  ➢ Pre-registered memory is mapped through kernel driver, thus still trackable in coOMMU.

• Concurrency
  ➢ coOMMU must properly handle concurrent pinning/unpinning requests between multiple vCPU threads and the unpinning thread.
Evaluation

- **Evaluation targets**
  - Performance overhead imposed by coIOMMU.
  - Memory footprint in various direct I/O usages.
  - The desired performance and security under different intra-guest protection policies.

- **Evaluated modes – coIOMMU vs. virtual VT-d**
  - Passthrough mode: no DMA remapping
    - PT-N (coIOMMU) vs. PT-O (virtual VT-d)
  - Strict mode: full protection with DMA remapping
    - ST-N (coIOMMU) vs. ST-O (virtual VT-d)
  - Lazy mode: relaxed protection with DMA remapping
    - LA-N (coIOMMU) vs. LA-O (virtual VT-d)

<table>
<thead>
<tr>
<th>mode</th>
<th>abbr.</th>
<th>DMA remapping</th>
<th>DMA buffer tracking</th>
<th>pinning model</th>
<th>protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>passthrough (virtual VT-d)</td>
<td>PT-O</td>
<td>unused</td>
<td>n/a</td>
<td>static</td>
<td>no</td>
</tr>
<tr>
<td>passthrough (coIOMMU)</td>
<td>PT-N</td>
<td>unused</td>
<td>used</td>
<td>fine-grained</td>
<td>no</td>
</tr>
<tr>
<td>strict (virtual VT-d)</td>
<td>ST-O</td>
<td>used</td>
<td>n/a</td>
<td>fine-grained</td>
<td>full</td>
</tr>
<tr>
<td>strict (coIOMMU)</td>
<td>ST-N</td>
<td>used</td>
<td>used</td>
<td>fine-grained</td>
<td>full</td>
</tr>
<tr>
<td>lazy (virtual VT-d)</td>
<td>LA-O</td>
<td>used</td>
<td>n/a</td>
<td>fine-grained</td>
<td>relaxed</td>
</tr>
<tr>
<td>lazy (coIOMMU)</td>
<td>LA-N</td>
<td>used</td>
<td>used</td>
<td>fine-grained</td>
<td>relaxed</td>
</tr>
</tbody>
</table>
Evaluation

• Three direct I/O usages: NIC/NVMe/GPU.

• Benchmarks
  ➢ Netperf: Aggregated throughput reported, for 16 concurrent Netperf instances running stream RX & TX tests.
  ➢ Nginx: Requests/second reported, for 16 concurrent requests to Nginx server installed in guest.
  ➢ Memcached: Requests/second reported, for 16*8 concurrent requests to Memcached installed in guest.
  ➢ FIO: IO requests/second reported, for 16 concurrent fio threads, each performing asynchronous direct random reads to NVMe.
  ➢ Open Arena: Frame-per-second (fps) reported as benchmark

<table>
<thead>
<tr>
<th>VM configuration</th>
<th>Direct I/O device</th>
<th>vCPU number</th>
<th>RAM size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel XL710 40Gbps NIC</td>
<td></td>
<td>16</td>
<td>32GB</td>
</tr>
<tr>
<td>Intel 760P series 1TB NVMe SSDs</td>
<td></td>
<td>16</td>
<td>32GB</td>
</tr>
<tr>
<td>Intel® Iris® Plus graphics 650 GPU</td>
<td></td>
<td>4</td>
<td>4GB</td>
</tr>
</tbody>
</table>
Performance

Throughput (higher is better)

- netperf stream rx (Gbps)
- netperf stream tx (Gbps)
- nginx (req/sec)
- memcached (req/sec)
- fio (iops)
- openarena (fps)

No observable performance impact with DMA Tracking!
(even in mixed netperf/fio scenario – data not shown here)
The number of pinned pages sampled in 3 second interval, taken from the beginning of the benchmarks to 6 seconds after their completion. ‘max’ indicates the total pages of guest memory.

Entire 32GB guest memory is statically pinned w/o DMA tracking

Fine-grained pinning with DMA Tracking, with only 0.5% of guest memory pinned

All four DMA remapping modes pin the minimal number of pages.
Memory Overcommitment

• Test setup
  ➢ Host: 64GB RAM size.
  ➢ VM1: 32GB RAM, running sysbench (no assigned device).
  ➢ VM2: 48GB RAM, assigned with Intel XL710 40Gbps NIC, running Netperf.
  ➢ Performance compared with running each benchmark alone.

• Results
  ➢ PT-O: Sysbench suffers 25+% performance drop, frequent page swaps.
  ➢ PT-N: No performance drop, with 49GB free memory.

The impact of memory overcommitment: static pinning (PT-O) vs. fine-grained pinning (PT-N)
Guest User Space Driver

- Run DPDK with coIOMMU and with the virtual VT-d respectively.

- No need to allocate and pin the entire guest memory in coIOMMU.

- No need to unpin the entire guest memory in coIOMMU.

- Likewise, static-pinning is avoided in coIOMMU when assigning NIC back to kernel driver.

- coIOMMU always adapts to the actual DMA buffer requirement, while virtual VT-d fails to do so when DMA remapping is off.
DMA Temporal Locality

• Test setup
  ❚ 16 Netperf TX instances ran for 15 minutes.
  ❚ ‘dd’ the virtual disk to /dev/zero, to contend the page allocation with the networking stack.

• Conclusion
  ❚ DMA temporal locality stays good, even in stressed scenario.
Future Work

- Co-work with DMA page faults
  - Help reduce the number of DMA faults by proactively pre-pin hot pages.
  - Mitigate non-faultable data paths if a device (e.g. many GPUs) only partially supports DMA page faults.

- Guest cooperation
  - A selfish guest may choose to not cooperate, e.g., by deliberately report fake DMA pages.
  - A quota mechanism can be applied, based on the service level agreement.

- Support two-level IOMMU address translation.
  - Hardware optimization to reduce virtual IOTLB invalidations.
Conclusions

• Established vIOMMUs cannot reliably eliminate static pinning in direct I/O.

• coIOMMU offers a reliable approach to achieve fine-grained pinning, with a cooperative DMA buffer tracking method.

• coIOMMU
  ➢ dramatically improves the efficiency of memory management in wide direct I/O usages with negligible cost;
  ➢ meanwhile sustains the desired security as required in specific protection usage;
  ➢ can be easily applied in various vIOMMU implementations.
Thanks!