Demon: An Efficient Solution for on-Device MMU Virtualization in Mediated Pass-Through

Yu Xu[1], Jianguo Yao[1], Yaozu Dong[2], Kun Tian[2], Xiao Zheng[2], Haibing Guan[1]
[1]Shanghai Jiao Tong University [2]Intel Corporation
{vatiminxuyu, jianguo.yao, hbguan}@sjtu.edu.cn {eddie.dong, kevin.tian}@intel.com
zhengxiao@gmail.com

Abstract

Memory Management Units (MMUs) for on-device address translation are widely used in modern devices. However, conventional solutions for on-device MMU virtualization, such as shadow page table implemented in mediated pass-through, still suffer from high complexity and low performance.

We present Demon, an efficient solution for on-**DE**vice **M**MU virtualizati**ON** in mediated pass-through. The key insight is that Demon takes advantage of IOMMU to construct a two-dimensional address translation and dynamically switches the 2nd-dimensional page table to a proper candidate when the device owner switches. In order to support fine-grained parallelism for the device with multiple engines, we put forward a hardware proposal that separates the address space of each engine and enables simultaneous device address remapping for multiple virtual machines (VMs). We implement Demon with a prototype named gDemon which virtualizes Intel GPU MMU. Nonetheless, Demon is not limited to this particular case. Evaluations show that gDemon provides up to 19.73x better performance in the media transcoding workloads and achieves performance improvement of up to 17.09% and 13.73% in the 2D benchmarks and 3D benchmarks, respectively, compared with gVirt. The current release of gDemon scales up to 6 VMs with moderate performance in our experiments. In addition, gDemon simplifies the implementation of GPU MMU virtualization with 37% code reduction.

CCS Concepts • **Software and its engineering → Memory management**; *Virtual machines*;

Keywords On-Device MMU, Virtualization, Mediated Pass-Through, IOMMU

VEE '18, March 25, 2018, Williamsburg, VA, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5579-7/18/03…$15.00
https://doi.org/10.1145/3186411.3186416

1 Introduction

A memory management unit (MMU) is a computer hardware unit that primarily performs the translation of virtual memory addresses to physical addresses. Nowadays, Central Processing Unit (CPU) MMU virtualization is well explored. Hardware technologies, such as two-dimensional address translation[21, 23], not only simplify the implementation, but also minimize the overhead of virtualization. Meanwhile, modern devices use MMUs for on-device address translation as well. Typical devices with an internal MMU include Graphic Processing Unit (GPU)[12, 14], Image Processing Unit (IPU)[5], InfiniBand[40], FPGA[24], and so on.

Unfortunately, there is currently no satisfactory solution focused on on-device MMU virtualization. Among mainstream I/O virtualization technologies, *device emulation*[20] and *para-virtualization*[19] perform address translation with the help of CPU, leading to high CPU usage and thus low performance. Hardware virtualization technology, such as *direct pass-through*, utilizes input-output memory management unit (IOMMU)[22] and dedicates a device to a single virtual machine (VM) with best performance at the cost of device sharing capability. *SR-IOV*[26] creates multiple PCIe functions, wherein each function can be assigned to different VMs with distinct I/O page tables to remap device address. However, it is difficult for direct pass-through and SR-IOV to support some specific virtualization features that require hypervisor collaboration, such as VM migration, replication, and so on, because necessary hardware state is directly controlled by guest and thus invisible to the hypervisor.

Another major paradigm of I/O virtualization, known as *mediated pass-through*[47], has been another emerging trend in order to deliver a product-level GPU virtualization solution[1, 10, 45], especially when hypervisor collaboration is essential for some virtualization usages such as VM migration. Mediated pass-through not only enables relatively good performance and sharing capability, but also supports live migration[7, 13] and flexible surface sharing[11]. Linux kernel has introduced a mediated device framework to support such demand since 4.10[9]. The comparisons of mainstream I/O virtualization solutions are summarized in Table 1.

The page table used for internal device address translation is considered a privileged resource that needs to be emulated in mediated pass-through. Shadow page table is widely used

Table 1. The comparisons of IO virtualization solutions.

Solution	Performance	Sharing	Scalability	Migration
Para-Virtualization	Low	Yes	Good	Yes
Pass-Through	High	No	No	Difficult
SR-IOV	High	Yes	Medium	Difficult
Mediated Pass-Through	Medium	Yes	Medium	Yes

for on-device MMU virtualization, such as shadow graphics translation tables in gVirt[45]. However, the implementation of shadow page table in mediated pass-through increases complexity and decreases performance for memory-intensive workloads involving frequent page table updates[25]. Note that IOMMU usually does not fully solve this issue here, because only specific applications can benefit from the Process Address Space ID (PASID) capability, e.g., shared virtual memory utilized by OpenCL. Most application data (2D, 3D, media, etc.) as well as the global configuration structures (power management, display, etc.) are still accessed without PASID tagged by GPU. For direct memory access (DMA) request without PASID, modern IOMMU architectures typically implement a single translation table per PCIe root identifier, therefore cannot perform address translation for multiple VMs in mediated pass-through.

This paper proposes Demon, an efficient solution for on-device MMU virtualization in mediated pass-through. Demon utilizes IOMMU to construct the 2^{nd}-dimensional address translation while reusing guest on-device MMU as the 1^{st}-dimensional translation. In order to address the device sharing challenge of integrating IOMMU into mediated pass-through, Demon reloads IOMMU remapping structure with physical-to-machine (P2M) address mapping of the corresponding VM when the device owner switches. For multi-engine device that ever employs DMA Request-without-PASID, we put forward a minor hardware proposal which is easy to implement in a circuit to ensure that the address space of each engine does not overlap with each other. As a result, IOMMU remapping structure can be populated with multiple P2M mappings to perform addresses remapping for different VMs at the same time.

In order to verify the proposed solution, we apply Demon to Intel GPU MMU virtualization and implement a prototype named gDemon. Nonetheless, Demon is not limited to this particular implementation. gDemon introduces a patch containing 500 insertions and 1800 deletions to gVirt, which greatly simplifies the implementation of GPU MMU virtualization. Comprehensive evaluations show that gDemon can improve the performance in media transcoding benchmarks compared with gVirt and gHyvi[25] for up to 19.73x and 1.86x, respectively. In the case of 2D/3D workloads, gDemon outperforms gVirt and gHyvi with a performance improvement of up to 17.09%/13.73% and 7.56%/8.09%, respectively. The current release of gDemon supports only up to 7 guest vGPU instances (6 in our experiments due to system memory limitation), and the performance of gDemon in the scalability evaluation is slight better than gVirt and gHyvi.

In summary, the contributions of this paper are as follows:

- We present Demon, an efficient solution for on-device MMU virtualization in mediated pass-through by means of IOMMU multiplexing. Demon constructs and dynamically repopulates the 2^{nd}-dimensional address translation with the appropriate address remapping.
- We put forward a minor hardware proposal in order to enable simultaneous address remapping in mediated pass-through. This feature is demonstrated with software method and discussed in detail.
- We implement Demon with a prototype named gDemon which virtualizes GPU MMU. gDemon is implemented on the basis of gVirt and is now available on GitHub.
- Comprehensive evaluations show that gDemon achieves significant performance improvements over gVirt, except for scalability. In addition, a detailed evaluation of the runtime overhead of gDemon is presented.

The rest of this paper is organized as follows. In Section 2, the theoretical background of IOMMU and mediated pass-through technology is provided, following which the motivation of Demon is introduced. The detailed design of Demon is described in Section 3, and the implementation of gDemon is presented in Section 4. In Section 5, comprehensive evaluations of gDemon are obtained. We discuss the limitations of gDemon, including scalability, in Section 6. The related work is discussed in Section 7. Finally, the conclusion of this paper is given in Section 8.

2 Background and Motivation

2.1 IOMMU

Most I/O devices support DMA, which allows CPU-independent access to system memory in order to accelerate I/O transactions. With the aim to support DMA in a virtualized environment, DMA remapping is introduced in hardware as one of the main features of IOMMU. IOMMU connects a DMA-capable I/O bus to the main memory and remaps the guest DMA address to the host physical address (HPA). Specifically, IOMMU hardware intercepts DMA transaction, then utilizes the corresponding I/O page table to determine whether the request is permitted, and resolve the real HPA.

Typical IOMMU hardware includes Intel VT-d[17], AMD IOMMU[18], IBM IOMMU[22] and ARM SMMU[33]. There are two types of DMA requests, i.e., Requests-without-PASID and Requests-with-PASID[6]. For devices supporting PASID in PCIe capabilities, such as Kaveri GPU[3], OS can take advantages of PASID capability to share the virtual address space of application processes with I/O devices. DMA Request-with-PASID is utilized only by specific applications such as OpenCL computing, while DMA Request-without-PASID is widely used for all mixed-use (2D/3D/media/computing) GPUs as we know. vGPU also needs to support mixed use for full GPU virtualization, therefore DMA Request-without-PASID is essential.

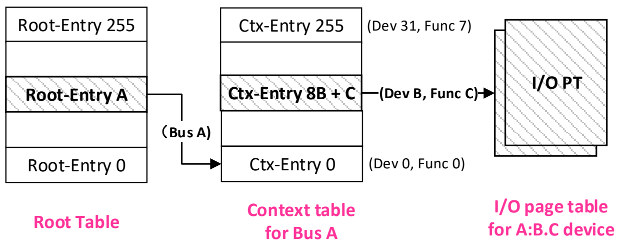


Figure 1. Remapping structure for Request-without-PASID.

Request-without-PASID is a common and fundamental case, and is therefore considered in this paper. Here, Intel VT-d is taken as an example in order to explain the remapping structure and indexing process of IOMMU. The remapping structure of VT-d is actually a multi-level I/O page table indexed by the PCI BDF (bus/device/function) number of I/O device, as shown in Figure 1. In order to determine the I/O page table used to perform address remapping for a given device, VT-d first finds the corresponding root entry based on the bus number, and then locates the appropriate context entry by the device and function number. The physical address of the root I/O page table is programmed in the context entry. Therefore, a device can correspond only to a unique I/O page table indexed by its BDF number in this context.

2.2 Mediated Pass-Through

In order to virtualize an I/O device, VMM can assign the device to single VM with the best performance at the cost of device sharing capability. Alternatively, VMM can build virtual device models that allow multiple VMs to share the physical device with lower performance. Mediated pass-through represents an intermediate between direct pass-through and full emulated virtual device, which passes through performance critical resources but traps and emulates privileged operations, as shown in Figure 2. In the case of full GPU virtualization using mediated pass-through, i.e., gVirt, the DMA space (also known as graphics memory) is passed through,

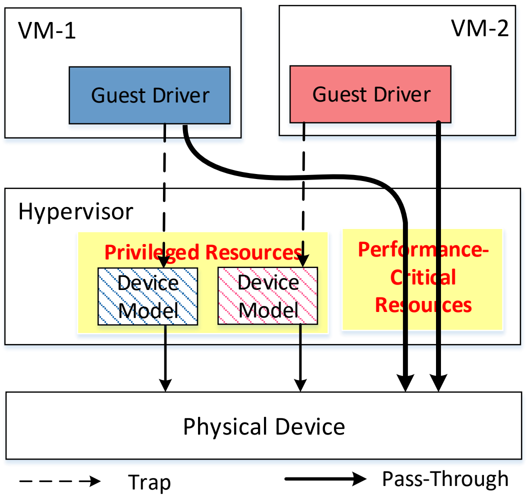


Figure 2. Architecture of mediated pass-through.

while the GPU registers and MMIO space are trapped and emulated. The guest GPU page tables are considered as privileged resources and shadowed in gVirt.

2.3 Drawbacks of Shadow Page Table in Mediated Pass-Through

Although mediated pass-through has been gradually recognized as an effective solution for I/O virtualization[9, 10, 28, 45, 47], it suffers from the complex and inefficient on-device MMU virtualization. The typical disadvantages can be illustrated on gVirt. Regardless the fact that gVirt performs well in most cases, it suffers from non-trivial performance overhead in memory-intensive graphics workloads that involves frequent page table updates. In the worst case, gVirt has 90% performance degradation in the media transcoding benchmark. The root cause is that gVirt synchronizes shadow page tables whenever the write-protected guest page tables are modified and then triggers VM exit.

Subsequently, an adaptive hybrid shadow page table that combines strict and relaxed shadowing schemes is introduced in gHyvi in order to mitigate the serious performance degradation. In contrast to strict (traditional) page table shadowing, relaxed page table shadowing removes the write-protection of guest page tables. Shadow page tables will be reconstructed to be consistent with guest page tables again to ensure correct translations before task context is submitted to the physical device. Although gHyvi mitigates the performance degradation issue, it consumes more memory to maintain snapshots, and slightly reduces the performance of general graphics workloads.

The most fundamental reason for this issue is the shadow page table implementation in mediated pass-through. Hypervisor intervention, such as handling of write-protection page fault, makes the shadowing process time-consuming. In addition to performance impact, the shadow page table implementation is rather complex. For example, shadow graphics translation tables in gVirt comprises about 3,500 lines of code (LOC), which makes it difficult to maintain and extend GPU MMU virtualization in gVirt and may cause potential bugs. Moreover, the shadowing mechanism requires guest driver/OS to notify hypervisor of the release of guest page tables so that hypervisor can remove write-protection for those guest pages. The creation of guest page tables can be notified immediately, or it can be inspected before task submission.

The drawbacks of shadow page table in mediated pass-through including inefficiency, complexity and compromise on guest driver/OS motivate this research. In this paper, an efficient solution for on-device MMU virtualization that eliminates the shadow page table implementation in mediated pass-through is proposed.

3 Design

The principle of Demon is to take advantages of IOMMU to construct the proper 2nd-dimensional address translation for devices that employ DMA Request-without-PASID. A device virtual address (DVA) programmed by device driver is firstly translated by guest page table to a guest physical address (GPA), and then remapped to a host physical address (HPA) through IOMMU with the corresponding I/O page table. The 2nd-dimensional address translation is transparent to guest VM and does not require modification of guest driver or OS, therefore Demon represents a general solution.

When IOMMU is integrated into mediated pass-through, there are significant challenges such as device sharing and potential performance degradation. The design of Demon has addressed the following three major challenges. Note that the design of Demon primarily applies to Request-without-PASID, and the case of Request-with-PASID will be discussed in Section 6.

3.1 I/O Page Table Switch

All the inbound DMA requests intercepted by IOMMU are required to identify the source device. The attribute that identifies the initiator is often labeled as *source-id*. For PCIe devices, the source-id is the device PCI BDF number, which is assigned by the configuration software. On the other hand, IOMMU employs hierarchical I/O page tables for the Request-without-PASID, also known as second-level translation[6], which remaps GPA to HPA. The hierarchical I/O page tables support a four-level paging structure, wherein each page table is 4KB in size and has 512 8-Byte entries. According to the IOMMU remapping structure described in Section 2.1, all the DMA Requests-without-PASID initiated by the same device can be remapped only through a unique I/O paging structure determined by the device BDF number. The *single BDF issue* represents the major limitation on sharing capability for devices that lack PASID support or mainly employ Requests-without-PASID.

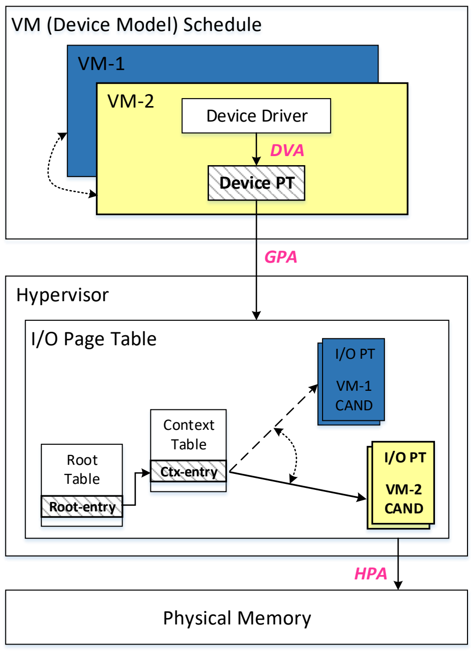


Figure 3. Time-division multiplexing of I/O page table.

Demon adopts a time-division multiplexing strategy to solve this issue, as shown in Figure 3. A complete I/O page table candidate is created based on the P2M mapping of the VM when a VM is booted. The physical device is bind to the privileged VM (Dom0), but the actual I/O page table, indexed by the device BDF number, is dynamically switched among the candidates when scheduling device models. Specifically, when a VM is scheduled to be the device owner, the root pointer of paging structure, programmed in the context entry, is reloaded with the root of the corresponding I/O page table candidate. In fact, a L3 I/O page table entry can map 1GB of address space, and the physical memory allocated to a VM is usually not very large. Considering that IOMMU might cache context entries and paging structures to minimize the overhead incurred by retrieving them from memory, the performance can be improved by keeping the context entry and the L4 page table unchanged, while reloading a certain number of L3 page table entries based on the maximum memory of all VMs.

The time-division multiplexing strategy enables all VMs to share the benefits of IOMMU, and most importantly, it provides an initial solution for on-device MMU virtualization in mediated pass-through.

3.2 I/O Page Table Partition

The time-division multiplexing strategy based on dynamic switching of I/O page table integrates IOMMU into mediated pass-through preliminarily. At one time moment, only one VM/device model can process tasks because the I/O page table used for Request-without-PASID can be populated only with the P2M mapping of current VM. However, for multi-engine devices where each engine can process tasks independently, workloads from different VMs can be dispatched to these engines for parallelism and high performance. The *VM parallelism issue* represents the foundation of higher performance in Demon.

In order to achieve fine-grained parallelism, IOMMU should be able to perform device address remapping for multiple VMs simultaneously. That is to say, P2M mappings of various VMs should coexist in the IO page table at one time moment. For Request-without-PASID, this issue finally comes down to the hardware. If the address space of each engine in the device is assumed to be separate, the I/O page table can subsequently be partitioned into corresponding ranges, each of which is populated with the proper P2M mapping. This condition is exactly the minor but critical hardware proposal that Demon puts forward.

When each engine in the device shares the same (or partial) paging structure or has the same (or overlapping) addressing capability, the address space of each engine will be completely (or partially) overlapped. There are several ways to eliminate the overlapping of address spaces of multiple engines. The solution chosen in Demon is based on the position to change the hardware design as little as possible, i.e., expanding (or restricting) the address space of an engine by opening (or closing) one or more bits of the page table entry (PTE) used by on-device MMU. The output of the expanded last level page table can be invalid, e.g., exceeding the maximal memory allocated to the VM, because the 2^{nd}-dimensional translation will perform another address remapping. For instance, if the unused 32^{th} PTE bit of an engine is set, the original GPA will become GPA + 4G (GPA | (1 << 32)), which will never collide with the [0, 4G) space of another engine whose 32^{th} PTE bit is not set. On the other hand, the original (GPA, HPA) mapping in the I/O page table candidate should be changed to (GPA + 4G, HPA) to match the shifted GPA. In fact, a feasible and compatible solution can be introduction of virtualization mode for the device. The hardware change is essential only in the virtualization mode, while the non-virtualization mode is exactly the native situation.

The I/O page table partition scheme enables simultaneous device address remapping for multiple VMs, when the hardware proposal is guaranteed. The DMA requests initiated by multiple engines in the same device can be remapped simultaneously as long as they have non-overlapping physical address spaces. The hypervisor needs to manage corresponding P2M mappings in the I/O page table candidates and make them coexist in the partitioned I/O page table. A detailed example on this subject is presented in Section 4.2 and discussed in Section 4.3.

3.3 Efficient IOTLB Flush

In IOMMU, effective translations can be cached in the I/O translation look-aside buffer (IOTLB) to minimize the overhead caused by I/O page table walking. However, in Demon, IOTLB should be flushed when switching the I/O page table because some IOTLB entries can be dirty for the scheduled device owner, which may result in incorrect address translation. The issue here is that IOTLB flush can cause performance degradation.

In order to minimize the overhead of IOTLB flush, the *Page-Selective-within-Domain Invalidation* policy[6] is chosen instead of the *Global Invalidation* policy. The Page-Selective-within-Domain policy invalidates only the IOTLB entries in a specified page range within a specified domain, where a domain is abstractly defined as an isolated environment in IOMMU to which a subset of host physical memory is allocated. IOMMU can use the domain identifier to tag its internal caches. In normal conditions, each VM can be considered as a separate domain. In Demon, a separate domain is assigned to the virtualized device by setting the *domain_id* field in the corresponding context_entry to a valid magic number (e.g., max_domain_number - 1). Only the IOTLB entries that map the overlapping memory range for all VMs within the specified domain are flushed, while all other IOTLB entries are unaffected. Therefore, the overhead of IOTLB flush is minimized.

4 Implementation

Regarding the fact that GPU is one of the most complex devices and that current GPU MMU virtualization in gVirt suffers from high complexity and performance drawback, we decide to implement Demon into gDemon based on gVirt. The source code of gDemon[1][2] is now available on GitHub. Specifically, we choose gVirt 2016-Q2 release of XenGT[3] as the base.

The architecture of gDemon with emphasis on GPU MMU virtualization is presented in Figure 4. As other components in gVirt remain unchanged, they are selectively omitted in Figure 4. Each guest VM in gDemon is assigned a vGPU instance that works well with the native graphics driver. The guest per-process graphics translation tables (PPGTTs) are no longer supervised while the guest global graphics translation table (GGTT) remains shared and shadowed because of the characteristic of GGTT. At any time, the I/O page table

[1]https://github.com/xy3613/gDemon-kernel

[2]https://github.com/xy3613/gDemon-xen

[3]https://lists.freedesktop.org/archives/intel-gfx/2016-July/101338.html

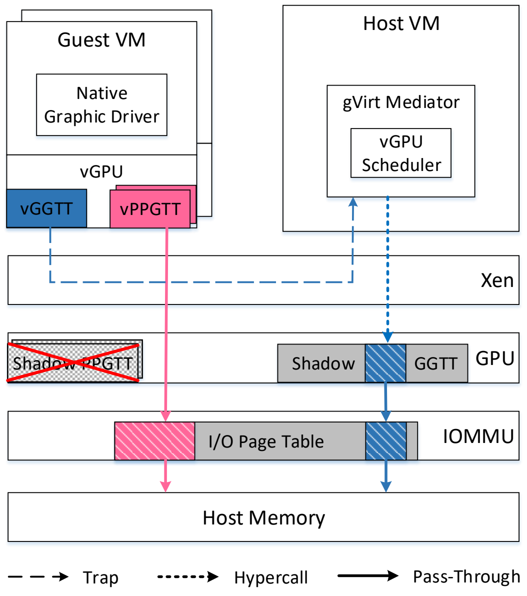


Figure 4. Architecture of gDemon.

is switched to the appropriate candidate so that IOMMU can properly remap the output of guest PPGTTs and shadow GGTT to real HPA.

4.1 Application of Demon in GTT

Intel Processor Graphics uses system memory as graphics memory, and employs graphics translation tables (GTT) to provide the translation from graphics memory address (GMA) to physical address (PA). There are two types of GTT, i.e., GGTT and PPGTTs. The global graphics memory is mapped through the unique one-level GGTT and serves as command buffer and frame buffer. Massive data accesses are made to the local graphics memory, which is accessed by GPU render engine through multi-level PPGTTs.

Application of Demon in gVirt intended for GPU MMU virtualization is straightforward, as described in Section 3.1. In our test platform, the BDF number of Intel GPU is 00:02.0 (VGA compatible controller). Consequently, the I/O page table indexed by 00:02.0 in IOMMU is multiplexed. Specifically, gVirt implements a virtual submit port for guest GPU workloads and a fair scheduler for scheduling of multiple vGPUs. Guest workloads are submitted to the real GPU device when the corresponding vGPU is scheduled to obtain the ownership on the GPU render engine.

During the process of vGPU scheduling, gDemon injects an extra hypercall to notify hypervisor to switch the I/O page table to corresponding candidate. With IOMMU facilitating address remapping, guest VM can program GPU workloads independently without being trapped and emulated by hypervisor during manipulation of page tables. Whenever a vGPU is scheduled for execution, the corresponding I/O page table is always ready, ensuring a proper address translation. Therefore, everything except the global graphics memory, which is mapped through GGTT, is satisfactory.

4.2 GGTT Coexistence

GGTT is mapped by PCI BAR0 (GTTMMADR) and located at MMIO region. Unlike PPGTT, which is used only by GPU render engine, GGTT is utilized by multiple components including GPU render engine, GPU display engine, and AGP Aperture which is a subset of global graphics memory that can be accessed simultaneously by CPU and GPU. Regarding the fact that the cost of GPU context switch is much greater than CPU, gVirt adopts the split CPU/GPU scheduling mechanism to significantly reduce GPU scheduling cost. Meanwhile, guest VM should be able to access its own global graphics memory, such as AGP Aperture, even when the corresponding vGPU is not current device owner. Therefore, gVirt adopts the partitioning approach for GGTT and the address space ballooning technique for significant performance optimization[45]. Consequently, GGTT remains partitioned and shadowed in gDemon to ensure functionality and high performance. However, in the gDemon architecture, wherein IOMMU remaps GPAs to HPAs for all DMA requests initiating by GPU, the access of global graphics memory will fail because GGTT entries have already been shadowed with HPAs. Regardless of which VM is the current device owner, IOMMU will incorrectly remap such DMA requests, which will cause improper memory access and memory corruption.

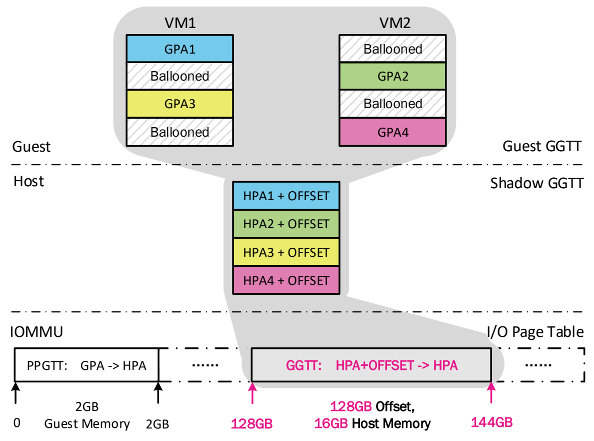


Figure 5. Coexistence model of GGTT.

For the moment, we adopt a software solution to temporarily substitute for the hardware proposal. The software solution is shifting of the global graphics memory to a dedicated address space that PPGTTs would not map, which can be achieved by adding of a large offset (e.g., 128G) to all shadow GGTT entries when GGTT updates are trapped and emulated. In the normal shadowing approach, when a guest VM intends to map GMA to GPA in guest GGTT entry, the GGTT entry should be shadowed with HPA, but here, HPA + 128G is filled in the shadow GGTT entry instead of HPA. On the other hand, IOMMU remaps the shifted HPA, i.e. HPA + 128G, to original HPA, as shown in Figure 5, where the memory allocated to each VM is 2GB and the host physical memory is 16GB. In order to achieve that, additional mapping that maps HPA + 128G to HPA for the entire physical memory range is required in all I/O page table candidates. Fortunately, the portion of I/O page table used for shadow GGTT remapping can be shared among VMs and remain unchanged during context switch, due to the fact that such remapping is identical for all VMs. Moreover, IOTLB entries caching the GGTT remapping need not to be invalidated.

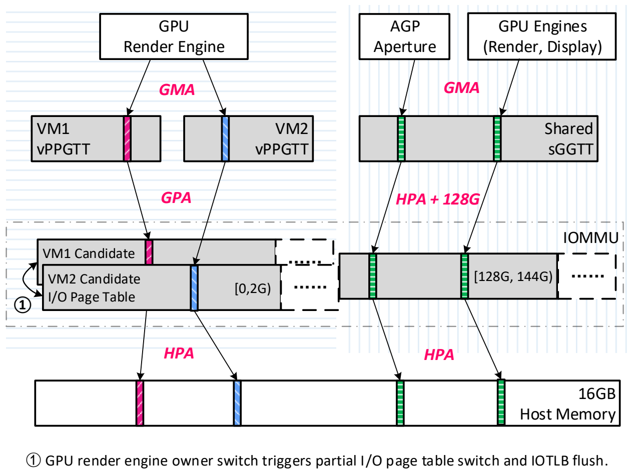

GPU Render Engine
AGP Aperture
GPU Engines (Render, Display)
GMA
GMA
VM1 vPPGTT
VM2 vPPGTT
Shared sGGTT
GPA
HPA + 128G
IOMMU
VM1 Candidate
VM2 Candidate I/O Page Table
[0,2G)
[128G, 144G)
①
HPA
HPA
16GB Host Memory
① GPU render engine owner switch triggers partial I/O page table switch and IOTLB flush.

Figure 6. Address translation example of gDemon.

A translation example obtained with 2GB guest memory, 16GB host memory and 128GB offset is presented in Figure 6. The local graphics memory access is fully passed-through. A GMA is firstly translated by guest vPPGTT to GPA, which represents the input of IOMMU, and then it is remapped to HPA. The global graphics memory access is passed-through, but GGTT is mediated. All GGTT updates are trapped and shadowed with a large offset, i.e., HPA + 128G. Similarly, the shifted HPA is translated to real HPA by IOMMU eventually. The owner switch of GPU render engine triggers partial I/O page table switch and IOTLB flush. The switch of I/O page table is accomplished by repopulation of a certain number of L3 I/O page table entries, which is similar to the switch of root pointer.

4.3 Discussion on gDemon

Despite of the comprehensive design, gDemon may introduce a security vulnerability. Since guest PPGTTs are no longer write-protected, hypervisor is not aware of any guest PPGTT modifications. In other words, DMA requests accessing the local graphics memory are passed through without hypervisor intervention. Therefore, there is a security vulnerability that enables the malicious guest driver to program the PPGTT entry with a large value, which falls into the shifted shadow GGTT space. Nonetheless, IOMMU has no knowledge whether this large value is a malicious GPA or a real shifted HPA. This scenario is possible because current PPGTT entry has the same addressing capability as GGTT entry, i.e., 39-bit addressing (0^{th}-38^{th}), while the 39^{th}-63^{th} bits are reserved or ignored[2].

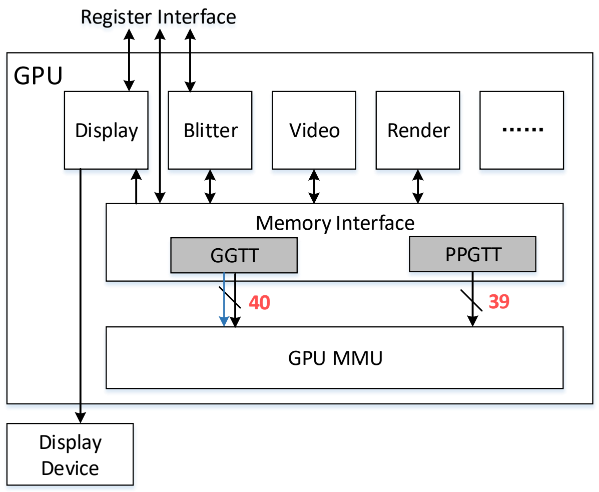


Figure 7. Minor hardware proposal of gDemon.

In order to maintain high performance, we should not intercept guest PPGTT updates, the ideal solution is hardware assistance for the device in the virtualization mode, i.e., to separate the address space of GGTT and PPGTT by opening of one of the reserved bits for GGTT entry, e.g., the unused 39^{th} bit. In this way, GGTT can be shifted to a higher address space that PPGTT cannot reach (i.e., 512GB offset). Toggling a bit of the GGTT PTE is easy to implement in the circuit, as shown in Figure 7. It is straightforward to add a circuit to the hardware or to unmask the reserved circuit. If gDemon becomes widely accepted, hardware vendors are likely to adopt this proposal.

5 Evaluations

In this section, comprehensive evaluations are performed in order to compare the performance of gDemon with original gVirt and enhanced gHyvi. We run 2D and 3D workloads on both Linux and Windows VMs. GMedia[25] benchmark is conducted in particular. In summary, the evaluation results show that gDemon outperforms gVirt and gHyvi in most cases, especially for the media transcoding workloads in GMedia benchmark. Specifically, for the media transcoding workloads, gDemon obtains the performance increases of up to 19.73x and 1.86x in comparison to gVirt and gHyvi, respectively. In addition, for 2D/3D workloads, gDemon outperforms gVirt with a performance improvement of up to 17.09%/13.73%, and outperforms gHyvi with a performance improvement of up to 7.56%/8.09%. The current release of gDemon supports up to 7 guest vGPU instances (6 in our experiments due to system memory limitation). Evaluations show that the scalability performance of gDemon is moderate, although slightly better than gVirt and gHyvi. In addition, gDemon simplifies the implementation of GPU MMU virtualization and provides the code reduction of 37% and 46% compared with gVirt and gHyvi, respectively, which represents an additional advantage besides the performance improvement.

5.1 Configuration

Our test platform consists of the 5th generation Intel Core processor i5-5300U with 4 CPU cores (2.30GHz), 16GB Kingston DDR3L 1600Mhz system memory and 240GB Intel 530 Series SSD. The processor graphics card in the test platform is the Intel HD Graphics 5500 (Broadwell GT2) with 4GB global graphics memory, of which AGP Aperture (low global graphics memory) takes up 1GB. We run 64-bit Ubuntu 14.04 LTS with 4.3.0 kernel in both Dom0 and Linux guest VM, and 64-bit Windows 7 in Windows guest VM, on Xen 4.6.0. All Linux and Windows VMs deploy the native graphics driver. Each standard VM is assigned with 2 vCPUs, 2GB system memory and 512MB partitioned global graphics memory, of which AGP Aperture takes up 128MB.

Moreover, the Cario-perf-trace 2D benchmark[4] which includes firefox-asteriods, firefox-scrolling, gnome-system-monitor and midori-zoomed, and the Phoronix Test Suite 3D benchmark[5] which includes lighsmark, nexuiz, openarena, urbanterror, warsow and xonotic are utilized to evaluate the 2D/3D performance on Linux. For evaluations on Windows, benchmarks including PassMark2D[6], 3DMark06, 3DMark11[7], Heaven[8] and Tropics[9] are chosen. All of the 2D/3D benchmarks run with 1920*1080 resolution. GMedia benchmark is conducted on Linux, and different configurations including channel and resolution are evaluated comprehensively. All of the VM options and evaluation benchmarks mentioned above are aligned with the experimental setup in the papers of gVirt[45] and gHyvi[25], which are not specifically set for gDemon.

[4]https://cairographics.org
[5]https://www.phoronix-test-suite.com
[6]https://www.passmark.com
[7]https://www.futuremark.com
[8]https://benchmark.unigine.com/heaven
[9]https://benchmark.unigine.com/tropics

5.2 Simplicity

Prior to performance testing, we firstly evaluate the complexity of GPU MMU virtualization modules using LOC counting. The shadow page table implementation of gVirt consists of 3,500 LOC in total, which includes 1,200 LOC for GGTT virtualization, 1,800 LOC for PPGTT virtualization, and 500 LOC for the common physical-to-machine translation service. gHyvi introduces a patch that comprises 600 LOC to PPGTT virtualization, which combines the relaxed shadow page table scheme with the original strict one.

gDemon mitigates GTT shadowing using IOMMU and eliminates all of 1,800 LOC for PPGTT virtualization. Meanwhile, gDemon introduces a patch with 500 LOC for the 2nd-dimensional address translation, wherein GGTT mediation contributes 50 LOC, and I/O page table management in the hypervisor contributes 450 LOC. Therefore, the implementation of two-dimensional address translation in gDemon consists of only 2,200 LOC, resulting in code reduction of 37% and 46% compared with gVirt (3,500 LOC) and gHyvi (4,100 LOC), respectively.

5.3 Performance

In this section, the performance of gDemon for various GPU workloads, including media transcoding workloads, Linux 2D/3D workloads, and Windows 2D/3D workloads, is presented.

5.3.1 GMedia Benchmark

GMedia benchmark is based on Intel Media Software Development Kit (MSDK) which provides hardware acceleration intended for generation of common media transcoding workloads. A characteristic of the GMedia benchmark is frequent page table modification due to large memory consumption and frequent page swapping. The greater the resolution and the number of channels are, the more obvious this characteristic is.

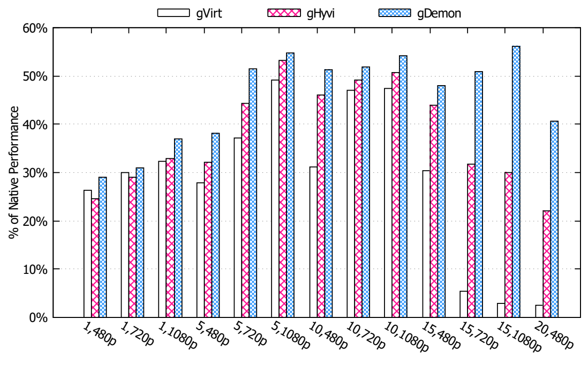


Figure 8. Performance of media transcoding workloads.

We profile a series of h264 media transcoding scenarios in GMedia with various channels (from 1 to 20) and various resolutions (from 480p to 1080p). The performance, i.e., frames per second (FPS), of gVirt, gHyvi and gDemon normalized to native case is presented in Figure 8. Generally, gDemon outperforms both gVirt and gHyvi in all test cases. When the workload is light, the performance of all three solutions is similar. However, when there are 15 channels, gVirt performance decreases dramatically, gHyvi performance degrades moderately, and gDemon performance increases. When there are 20 channels, gDemon performance starts to decrease slightly. Nevertheless, in the 15-channel 1080p scenario, gDemon achieves up to 19.73x and 1.86x better performance than gVirt and gHyvi, respectively.

Furthermore, there are some interesting observations in the test results. Firstly, gDemon does not perform well for small number of channels, but the performance increases as the number of channels increases. The impact of other overheads such as vCPU and GPU command auditing is dominated when GPU load is light, and the optimization of gDemon is reflected when GPU load becomes high. The effect of resolution is similar, when the number of channels is fixed, the performance of gDemon increases as the resolution increases. Additionally, even gDemon can achieve only up to 60% of native performance, which does not meet what gHyvi claims. There are two major reasons for that. Firstly, hardware semaphore is not well supported in the execlist mode. Namely, GPU workloads are submitted in the execlist mode in Broadwell microarchitecture, while in the ringbuffer mode in Haswell microarchitecture. The lack of hardware semaphore for efficient coordination of tasks in Broadwell results in excessive GPU idle time. Secondly, there are too many IRQs/MMIO accesses introduced by execlist mode. For a typical 1080p transcoding workload, there are 2k IRQs and 30k MMIO_RWs per second in our test platform, which induces high CPU utilization and plenty of GPU idle time. The problem is mostly related to GPU virtualization optimization, and slightly to GPU MMU virtualization.

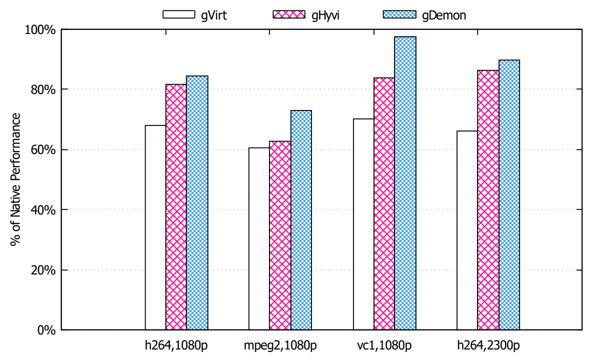


Figure 9. Performance of media decoding workloads.

In order to prove that, we decompose the transcoding task into decoding and encoding part, and evaluate the decoding part again. The decoding workload eliminates the creation of intermediate files, resulting in fewer IRQs and MMIO_RWs. In addition, the number of channels in the decoding workload is set to 1 in order to mitigate the impact of unsupported hardware semaphores. The decoding performance of various sources including h264, mpeg2 and vc1 is presented in Figure 9, wherein it can be observed that performance trends of gVirt, gHyvi and gDemon are similar to the trends presented in Figure 8. However, gDemon can achieve at least 80% native performance on average this time.

5.3.2 2D/3D Benchmark

The performance of 2D/3D workloads on Linux and Windows is presented in Figure 10 and Figure 11, respectively. All the performance data are normalized to native case.

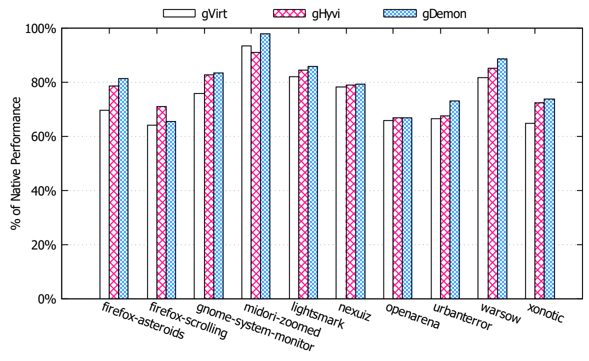


Figure 10. Performance of Linux 2D/3D workloads.

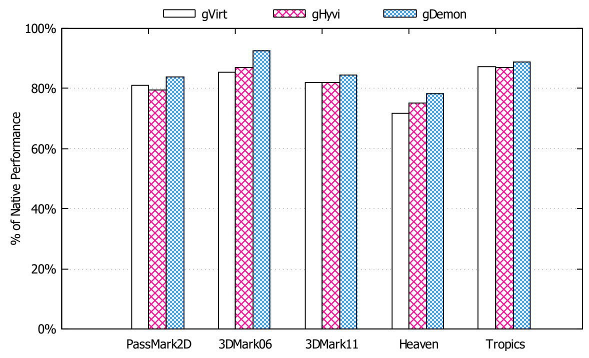


Figure 11. Performance of Windows 2D/3D workloads.

As expected, gDemon performs the best in most of the test cases, and achieves an average of 81.62% of native performance. For Linux 2D workloads, gDemon achieves performance improvement in firefox-asteroids, firefox-scrolling, gnome-system-monitor and midori-zoomed of 17.09%, 1.90%, 10.03%, 4.96,% and 3.38%, -7.79%, 1.19%, 7.56%, compared with gVirt and gHyvi, respectively. For Linux 3D workloads, gDemon achieves performance improvement in lightsmark, nexuiz, openarena, urbanterror, warsow and xonotic of 4.70%, 1.33%, 1.23%, 9.68%, 8.83%, 13.73%, and 1.64%, 0.26%, 0.00%, 8.09%, 4.35%, 1.81%, compared with gVirt and gHyvi, respectively. For Windows workloads, gDemon achieves performance improvement running PassMark2D, 3Dmark06, 3Dmark11, Heaven and Tropics of 3.43%, 8.61%, 2.77%, 9.05%, 1.85%, and 5.64%, 6.54%, 2.86%, 4.33%, 2.09%, compared with gVirt and gHyvi, respectively.

The performance improvement of gDemon for 2D/3D benchmarks is moderate, because there are ordinary page table manipulations in normal 2D/3D workloads. In such scenario, the overhead of GPU MMU virtualization is not dominant, and other overheads, such as GPU command auditing, also affect the performance. Nonetheless, it has been demonstrated that gDemon can effectively handle all types of workloads, compared with gVirt and gHyvi.

5.4 Scalability

The scalability of gDemon is evaluated through simultaneous running of the same workloads on multiple VMs. We select gnome-system-monitor, lightsmark, urbanterror, xonotic, Heaven and Tropics as benchmarks for scalability evaluation, and the number of VMs varies from 1 to 6. The performance of the workload that runs simultaneously on multiple VMs is calculated as

$$\min(fps_1 ... fps_n) * n \tag{1}$$

or

$$n * frames / \max(time_1 ... time_n), \tag{2}$$

where n represents the number of guest VMs, $frames$ indicates the total frames of the specified workload, fps_i denotes the FPS performance of the workload runs on the i^{th} VM, and $time_i$ indicates the runtime of the workload runs on the i^{th} VM.

The host physical memory is 16GB and the AGP Aperture is 1GB, wherein Xen occupies a certain part of memory, host VM and each guest VM occupies 2GB system memory and 128MB AGP Aperture. Therefore, up to 6 guest VMs can be booted due to system memory limitation in our test platform, and up to 7 guest VMs due to AGP Aperture size limitation in the current release of gDemon, even if there is sufficient system memory.

The 2D/3D performance of multiple VMs hosted by gDemon is presented in Figure 12a. As shown in Figure 12a, the performance increases as the number of VMs increases, but slightly decreases when there are more than 4 VMs. The performance of some workloads, such as lightsmark, Heaven and Tropics, starts to decrease earlier, i.e., when there are more there 2 VMs, while the performance of gnome-system-monitor does not decline. Hence, running on multiple VMs improves the throughput of workloads when GPU is underutilized. When the turning point is reached, the scheduling overhead, such as context switch and IOTLB flush, is gradually revealed, which degrades the performance. Generally, gDemon can achieve the native performance of at least 60%, except for the Tropics benchmark when there are 6 VMs.

The performance of gVirt, gHyvi and gDemon for 2, 4 and 6 VMs, is presented in Figure 12b, Figure 12c and Figure 12d, respectively. We can observe that all of them scale in the similar tendency, and gDemon obtains slightly better performance than gVirt and gHyvi in most cases.

5.5 Overhead Analysis

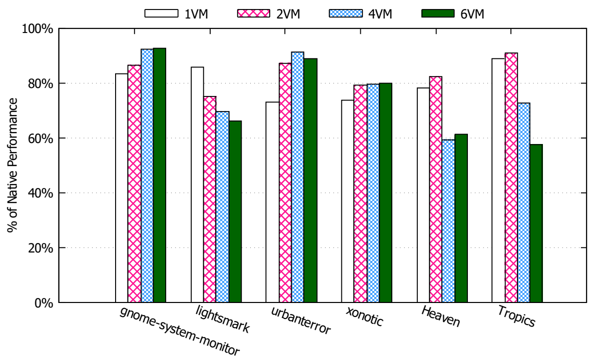
The evaluations of runtime overhead of GPU MMU virtualization in gHyvi and gDemon based on the lightsmark benchmark, are presented in Figure 13a and Figure 13b, respectively. In our test platform, it takes about 2.4k cycles to handle the GGTT write-protection page fault, 9.1k cycles to switch the I/O page table, and 20.1k cycles to flush the selected IOTLB entries in gDemon. The hybrid shadow page table in gHyvi extends the processing time of PPGTT write-protection page fault to 16.3k cycles, and it takes about 1.3k cycles to synchronize an out-of-sync PTE. When 6 VMs are booted, there are about 156 context switches in gDemon, while there are 6.2k out-of-sync PTEs in gHyvi. Therefore, it can be calculated that gDemon takes only half of time required by gHyvi for critical operations.

On the other hand, the cost of creation of a complete I/O page table candidate for guest VM is non-trivial, and it takes about 258,450k cycles in our test platform. However, the creation is performed only once, at the very beginning of guest VM booting, without compromising of user experience. IOMMU eliminates the overhead of the hypervisor intervention, but it increases the maximum path of page walk required to generate the HPA[23]. The implicit overhead of the 2nd-dimensional page walk is difficult to quantify and thus not studied in depth in this paper.

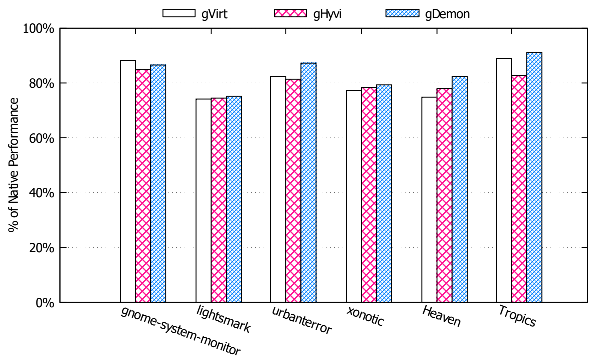
6 Discussion

6.1 Scope of Application

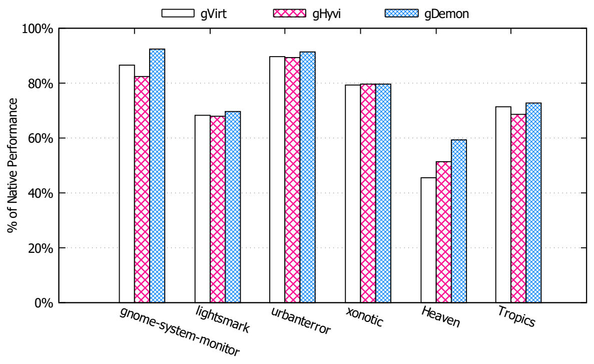
Demon plays a role in mediated pass-through, especially for devices that ever employ DMA Request-without-PASID, which is the mainstream of the GPU market. Even if some GPUs implement PASID capability, both Request-without-PASID and Request-with-PASID can coexist in such devices especially when mixed graphics usages (2D, 3D, media, computing) are supported simultaneously. It is possible that a GPU is built specifically for computing, so that all DMA requests initiated by this device can always be PASID-tagged. In this case, the nested address translation of IOMMU can also be utilized to eliminate shadow page tables, without relying on gDemon. However, there are challenges with utilizing nested IOMMU. Take Kaveri GPU[3] with PASID support for example, two-level address translation using nested page tables seems the first and best choice for the implementation of HSA system virtualization[35]. However, the shadow page table mechanism is adopted finally in this work due to the fact that there are two different paths for translating a GPU virtual address in Kaveri, i.e., IOMMU and GPUVM.



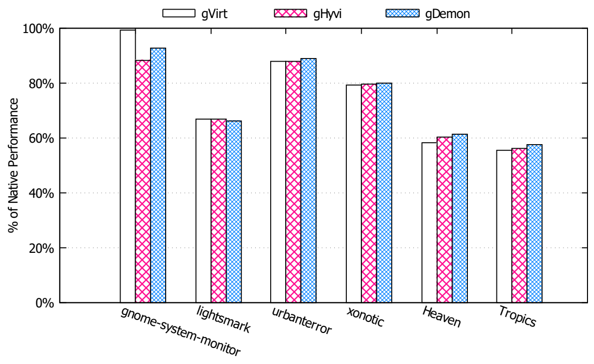
(a) gDemon with different numbers of VMs.



(b) 2 VMs with different solutions.

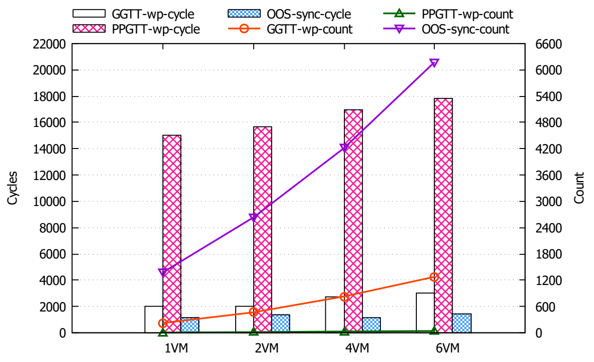


(c) 4 VMs with different solutions.

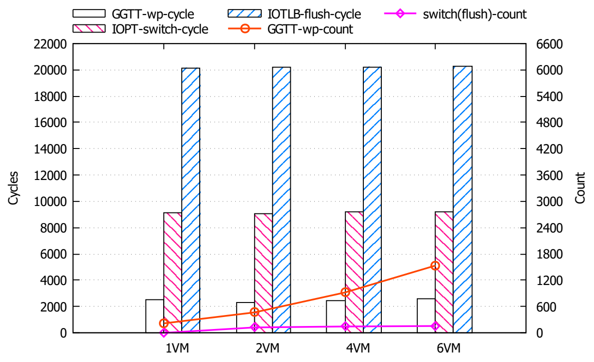


(d) 6 VMs with different solutions.

Figure 12. Scalability evaluations with different solutions (gDemon, gVirt and gHyvi).



(a) gHyvi.



(b) gDemon.

Figure 13. Statistics of cost and frequency of performance-critical operations, based on the lightsmark benchmark.

Demon deserves to be considered in the above cases. Although the implementation of Demon is based on Intel GPU, the overall design is architecture independent, whether it is integrated GPU or discrete GPU[16]. gDemon can be ported to other GPU virtualization solutions based on mediated pass-through, and give insight into generic on-device MMU virtualization.

6.2 Scalability

The current release of gDemon can support only up to 7 guest vGPU instances with moderate performance due to AGP Aperture size limitation even if there is sufficient system memory. gScale[48] addresses the scalability issue of gVirt with a novel sharing scheme, and supports up to 15 vGPU instances in Linux or 12 vGPU instances in Windows. The current release of gDemon is ready but has not merged this feature yet.

7 Related Work

7.1 Device MMU Virtualization

Nowadays, on-device MMUs are widely used in modern high-performance devices intended for complex functions. However, traditional I/O virtualization approaches are less concerned with on-device MMU virtualization.

Device emulation[20] and para-virtualization[27] perform on-device address translation with the help of CPU, which is inefficient. For hardware I/O virtualization technology, such as IOMMU[22] and SR-IOV[26], on-device MMU virtualization is usually facilitated through the two-dimensional hardware page walking. However, it is difficult for such direct pass-through solutions to support specific virtualization features that require hypervisor collaboration, such as VM migration, replication, and so on. Swiotlb[46] bypasses IOMMU and caches address mappings in virtual lookaside buffer, but memory replication between DMA buffers and dedicated bounce buffers incurs significant overhead. Furthermore, VPIO[47] introduces the concept of virtual pass-through I/O, wherein guest VMs can directly access hardware resources most of the time (e.g., legacy NICs such as NE2000 and RTL8139). Mediated pass-through directly passes performance-critical resources, while mediating privileged operations on device (e.g., GPU). Such mediated pass-through solutions adopt shadow page table for on-device MMU virtualization. Moreover, this work[35] takes advantage of shared virtual memory of HSA system, and utilizes DMA Request-with-PASID as well as shadow page table to virtualize the GPU memory. However, this solution is primarily applicable to HSA systems such as AMD Kaveri A10-7850K APU, which is not universal.

7.2 GPU Virtualization

GPU virtualization is an emerging field of research[34] in recent years. Typical GPU virtualization approaches and implementations are summarized in this section.

Device emulation is impractical for GPU virtualization, and Qemu[20] emulates only legacy VGA cards to support basic 2D usages. Para-virtualization is widely used in GPU virtualization. WireGL[36] and Chromium[37] intercept and render OpenGL commands. VMGL[38], Xen3D[43] and Blink[32] forward guest OpenGL API calls to the host by installation of new OpenGL libraries in VM. GViM[31], vCUDA[42] and rCUDA[29] implement similar API forwarding technologies focused on CUDA commands in GPGPU computing.

Two full GPU virtualization solutions, i.e., gVirt[45] and GPUvm[44], have been proposed recently. gVirt is the first full GPU virtualization solution, which applies mediated pass-through to Intel GPU. gVirt enables high performance and device sharing, and supports vGPU live migration in industrial products. There are many works based on gVirt such as gHyvi and gScale, in order to optimize page shadowing and scalability, respectively. GPUvm implements both para-virtualization and full-virtualization for Nvidia cards. However, full-virtualization exhibits non-trivial overhead for MMIO handling, while optimized para-virtualization achieves only half of native performance at best.

Nvidia GRID[8] virtualization platform provides vGPU instances with full GPU capabilities that allow local clients to interface with remote virtualization stations. Nvidia GRID vGPU[10] is actually an implementation of mediated pass-through, and supports as many as 16 VMs on the Nvidia GRID K2 board. Recently, Nvidia has also announced migration support[15] for GRID vGPU, which is demonstrated on Citrix XenServer. AMD has announced the hardware-assisted GPU virtualization solution labeled as MxGPU[4] recently. AMD claims that MxGPU is designed for both OpenCL and graphics performance, and supports up to 16 users on a single GPU. Currently, some features including live migration are not compatible with MxGPU. Limited technical details make it difficult to fully understand and validate GRID and MxGPU, since neither of them is open source.

In addition, there is a lot of research on vGPU resource sharing[49], scheduling[41, 50], QoS[30, 39] and so on. Particularly, [35] achieves GPU sharing, which allows processes of all guest VMs to share GPU in the HSA-compliant system, relying on HSA features such as shared virtual memory.

8 Conclusion

Demon represents an efficient solution for on-device MMU virtualization in mediated pass-through, which reuses guest on-device MMU as the 1st-dimensional translation and utilizes IOMMU to construct and dynamically switch the 2nd-dimensional address translation. In addition, a minor hardware proposal for separating the address space of each device engine is proposed and emulated in software to enable simultaneous device address remapping for multiple VMs in Demon. Demon is implemented with a prototype named gDemon with the aim of providing a better GPU MMU virtualization compared with the shadow page table approach. Evaluations show that gDemon provides up to 19.73x better performance for GMedia benchmark, performance improvement of up to 17.09%/13.73% for 2D/3D benchmarks, and code reduction of 37% for concise implementation, compared with gVirt. The current release of gDemon supports up to 7 guest vGPU instances (6 in our experiments), and the scalability performance of gDemon is moderate, although slightly better than gVirt and gHyvi.

Acknowledgment

This work was supported in part by the Program for NSFC (No.61772339, 61525204), and the STCSM project (No.16QA1402200). The corresponding author is Prof. Yao, Jianguo.

References

[1] 2012. KVM on System z: Channel I/O And How To Virtualize It. https://www.linux-kvm.org/images/1/13/2012-forum-channel-io-kvm-forum.pdf. (2012).

[2] 2015. Intel Open Source HD Graphics and Intel Iris Graphics Programmer's Reference Manual, Volume 5: Memory Views. https://01.org/sites/default/files/documentation/intel-gfx-prm-osrc-bdw-vol05-memory_views_3.pdf. (2015).

[3] 2016. AMD Kaveri. http://www.amd.com/en-us/products/processors/desktop/a-series-apu. (2016).

[4] 2016. AMD Multiuser GPU (MxGPU). http://www.amd.com/en-us/solutions/professional/virtualization. (2016).

[5] 2016. Dual-core ARM Cortex-M4 IPU subsystem. https://training.ti.com/sites/default/files/docs/Running_RTOS_on_Cortex_M4_SLIDES.pdf. (2016).

[6] 2016. Intel VT-d Architecture Specification. http://www.intel.com/content/dam/www/public/us/en/documents/product-specifications/vt-directed-io-spec.pdf. (2016).

[7] 2016. Live Migration of vGPU. http://schd.ws/hosted_files/xensummit2016/c7/XenGT-LiveMigration_1.00.pdf. (2016).

[8] 2016. NVIDIA GRID Virtual GPU Technology. https://www.nvidia.com/en-us/design-visualization/technologies/virtual-gpu/. (2016).

[9] 2016. VFIO Mediated devices. https://www.kernel.org/doc/Documentation/vfio-mediated-device.txt. (2016).

[10] 2016. VGPU on KVM, VFIO based mediated device framework. http://www.linux-kvm.org/images/5/59/02x03-Neo_Jia_and_Kirti_Wankhede-vGPU_on_KVM-A_VFIO_based_Framework.pdf. (2016).

[11] 2017. Generic Buffer Sharing Mechanism for Mediated Devices. https://kvmforum2017.sched.com/event/BnoJ/generic-buffer-sharing-mechanism-for-mediated-devices-tina-zhang-intel. (2017).

[12] 2017. Intel Processor Graphics. https://01.org/zh/linuxgraphics. (2017).

[13] 2017. Live Migration with Mediated Device. https://kvmforum2017.sched.com/event/BnoH/live-migration-with-mediated-device-yulei-zhang-intel. (2017).

[14] 2017. NVIDIA GeForce series. http://www.geforce.com/hardware. (2017).

[15] 2017. NVIDIA GRID Showcases vGPU Monitoring and Migration. https://blogs.nvidia.com/blog/2017/06/22/high-availability-nvidia-grid-showcases-vgpu-monitoring-and-migration/. (2017).

[16] 2018. Radeon RX Vega M Graphics. https://newsroom.intel.com/wp-content/uploads/sites/11/2018/01/8th-gen-radeon-rx-vega-m-product-overview.pdf. (2018).

[17] Darren Abramson, Jeff Jackson, Sridhar Muthrasanallur, Gil Neiger, Greg Regnier, Rajesh Sankaran, Ioannis Schoinas, Rich Uhlig, Balaji Vembu, and John Wiegert. 2006. Intel Virtualization Technology for Directed I/O. *Intel technology journal* 10, 3 (2006).

[18] I AMD and O Virtualization. 2007. Technology (IOMMU) Specification. (2007).

[19] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. 2003. Xen and the art of virtualization. In *ACM SIGOPS Operating Systems Review*, Vol. 37. ACM, 164–177.

[20] Fabrice Bellard. 2005. QEMU, a fast and portable dynamic translator.. In *USENIX Annual Technical Conference, FREENIX Track*. 41–46.

[21] Muli Ben-Yehuda, Michael D Day, Zvi Dubitzky, Michael Factor, Nadav Har'El, Abel Gordon, Anthony Liguori, Orit Wasserman, and Ben-Ami Yassour. 2010. The Turtles Project: Design and Implementation of Nested Virtualization.. In *OSDI*, Vol. 10. 423–436.

[22] Muli Ben-Yehuda, Jon Mason, Jimi Xenidis, Orran Krieger, Leendert Van Doorn, Jun Nakajima, Asit Mallick, and Elsie Wahlig. 2006. Utilizing IOMMUs for virtualization in Linux and Xen. In *OLS'06: The 2006 Ottawa Linux Symposium*. Citeseer, 71–86.

[23] Ravi Bhargava, Benjamin Serebrin, Francesco Spadini, and Srilatha Manne. 2008. Accelerating two-dimensional page walks for virtualized systems. In *ACM SIGARCH Computer Architecture News*, Vol. 36. ACM, 26–35.

[24] Klaus Danne. 2004. Memory management to support multitasking on fpga based systems. In *Proceedings of the International Conference on Reconfigurable Computing and FPGAs*. 21.

[25] Yaozu Dong, Mochi Xue, Xiao Zheng, Jiajun Wang, Zhengwei Qi, and Haibing Guan. 2015. Boosting GPU virtualization performance with hybrid shadow page tables. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*. 517–528.

[26] Yaozu Dong, Xiaowei Yang, Jianhui Li, Guangdeng Liao, Kun Tian, and Haibing Guan. 2012. High performance network virtualization with SR-IOV. *J. Parallel and Distrib. Comput.* 72, 11 (2012), 1471–1480.

[27] Yaozu Dong, Jianguo Yao, Halbing Guan, R Ananth Krishna, and Yunhong Jiang. 2017. MobiXen: Porting Xen on Android devices for mobile virtualization. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 946–949.

[28] Micah Dowty and Jeremy Sugerman. 2009. GPU virtualization on VMware's hosted I/O architecture. *ACM SIGOPS Operating Systems Review* 43, 3 (2009), 73–82.

[29] José Duato, Antonio J Pena, Federico Silla, Rafael Mayo, and Enrique S Quintana-Ortí. 2010. rCUDA: Reducing the number of GPU-based accelerators in high performance clusters. In *High Performance Computing and Simulation (HPCS), 2010 International Conference on*. IEEE, 224–231.

[30] Haibing Guan, Jianguo Yao, Zhengwei Qi, and Runze Wang. 2015. Energy-efficient SLA guarantees for virtualized GPU in cloud gaming. *IEEE Transactions on Parallel and Distributed Systems* 26, 9 (2015), 2434–2443.

[31] Vishakha Gupta, Ada Gavrilovska, Karsten Schwan, Harshvardhan Kharche, Niraj Tolia, Vanish Talwar, and Parthasarathy Ranganathan. 2009. GViM: GPU-accelerated virtual machines. In *Proceedings of the 3rd ACM Workshop on System-level Virtualization for High Performance Computing*. ACM, 17–24.

[32] Jacob Gorm Hansen. 2007. Blink: Advanced display multiplexing for virtualized applications. In *Proceedings of NOSSDAV*.

[33] ARM Holdings. 2013. ARM system memory management unit architecture specificationâĂŤSMMU architecture version 2.0. (2013).

[34] Cheol-Ho Hong, Ivor Spence, and Dimitrios S Nikolopoulos. 2017. GPU Virtualization and Scheduling Methods: A Comprehensive Survey. *ACM Computing Surveys (CSUR)* 50, 3 (2017), 35.

[35] Yu-Ju Huang, Hsuan-Heng Wu, Yeh-Ching Chung, and Wei-Chung Hsu. 2016. Building a kvm-based hypervisor for a heterogeneous system architecture compliant system. In *Proceedings of the12th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*. ACM, 3–15.

[36] Greg Humphreys, Matthew Eldridge, Ian Buck, Gordan Stoll, Matthew Everett, and Pat Hanrahan. 2001. WireGL: a scalable graphics system for clusters. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, 129–140.

[37] Greg Humphreys, Mike Houston, Ren Ng, Randall Frank, Sean Ahern, Peter D Kirchner, and James T Klosowski. 2002. Chromium: a stream-processing framework for interactive rendering on clusters. *ACM transactions on graphics (TOG)* 21, 3 (2002), 693–702.

[38] H Andrés Lagar-Cavilla, Niraj Tolia, Mahadev Satyanarayanan, and Eyal De Lara. 2007. VMM-independent graphics acceleration. In *Proceedings of the 3rd international conference on Virtual execution environments*. ACM, 33–43.

[39] Qiumin Lu, Jianguo Yao, Zhengwei Qi, Bingsheng He, et al. 2016. Fairness-efficiency allocation of cpu-gpu heterogeneous resources. *IEEE Transactions on Services Computing* (2016).

[40] Gregory F Pfister. 2001. An introduction to the infiniband architecture. *High Performance Mass Storage and Parallel I/O* 42 (2001), 617–632.

[41] Zhengwei Qi, Jianguo Yao, Chao Zhang, Miao Yu, Zhizhou Yang, and Haibing Guan. 2014. VGRIS: Virtualized GPU resource isolation and scheduling in cloud gaming. *ACM Transactions on Architecture and Code Optimization (TACO)* 11, 2 (2014), 17.

[42] Lin Shi, Hao Chen, Jianhua Sun, and Kenli Li. 2012. vCUDA: GPU-accelerated high-performance computing in virtual machines. *IEEE Trans. Comput.* 61, 6 (2012), 804–816.

[43] Christopher Smowton. 2009. Secure 3D graphics for virtual machines. In *Proceedings of the Second European Workshop on System Security*. ACM, 36–43.

[44] Yusuke Suzuki, Shinpei Kato, Hiroshi Yamada, and Kenji Kono. 2014. GPUvm: Why not virtualizing GPUs at the hypervisor?. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. 109–120.

[45] Kun Tian, Yaozu Dong, and David Cowperthwaite. 2014. A full GPU virtualization solution with mediated pass-through. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. 121–132.

[46] David E Williams. 2007. *Virtualization with Xen (tm): Including XenEnterprise, XenServer, and XenExpress*. Syngress.

[47] Lei Xia, Jack Lange, Peter Dinda, and Chang Bae. 2009. Investigating virtual passthrough I/O on commodity devices. *ACM SIGOPS Operating Systems Review* 43, 3 (2009), 83–94.

[48] Mochi Xue, Kun Tian, Yaozu Dong, Jiacheng Ma, Jiajun Wang, Zhengwei Qi, Bingsheng He, and Haibing Guan. 2016. gScale: Scaling up GPU Virtualization with Dynamic Sharing of Graphics Memory Space.. In *USENIX Annual Technical Conference*. 579–590.

[49] Jianguo Yao, Qiumin Lu, and Zhengwei Qi. 2017. Automated Resource Sharing for Virtualized GPU with Self-Configuration. In *Reliable Distributed Systems (SRDS), 2017 IEEE 36th Symposium on.* IEEE, 250–252.

[50] Chao Zhang, Jianguo Yao, Zhengwei Qi, Miao Yu, and Haibing Guan. 2014. vGASA: Adaptive scheduling algorithm of virtualized GPU resource in cloud gaming. *IEEE Transactions on Parallel and Distributed Systems* 25, 11 (2014), 3036–3045.