

# IOMMU: Strategies for Mitigating the IOTLB Bottleneck

Nadav Amit, Muli Ben-Yehuda, Ben-Ami Yassour

▶ **To cite this version:**

Nadav Amit, Muli Ben-Yehuda, Ben-Ami Yassour. IOMMU: Strategies for Mitigating the IOTLB Bottleneck. WIOSCA 2010 - Sixth Annual Workshorp on the Interaction between Operating Systems and Computer Architecture, Jun 2010, Saint Malo, France. inria-00493752

**HAL Id: inria-00493752**
**https://inria.hal.science/inria-00493752**

Submitted on 21 Jun 2010

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# IOMMU: Strategies for Mitigating the IOTLB Bottleneck

Nadav Amit[1] Muli Ben-Yehuda[2] Ben-Ami Yassour[2]
namit@cs.techion.ac.il muli@il.ibm.com benami@il.ibm.com

[1]*Technion – Israel Institute of Technology*
[2]*IBM Research – Haifa*

## Abstract

The *input/output memory management unit* (IOMMU) was recently introduced into mainstream computer architecture when both Intel and AMD added IOMMUs to their chip-sets. An IOMMU provides memory protection from I/O devices by enabling system software to control which areas of physical memory an I/O device may access. However, this protection incurs additional *direct memory access* (DMA) overhead due to the required address resolution and validation.

IOMMUs include an *input/output translation lookaside buffer* (IOTLB) to speed-up address resolution, but still every IOTLB cache-miss causes a substantial increase in DMA latency and performance degradation of DMA-intensive workloads. In this paper we first demonstrate the potential negative impact of IOTLB cache-misses on workload performance. We then propose both system software and hardware enhancements to reduce IOTLB miss rate and accelerate address resolution. These enhancements can lead to a reduction of over 60% in IOTLB miss-rate for common I/O intensive workloads.

## 1 Introduction

The majority of current I/O devices support *direct memory access* (DMA), which allows them to access the system memory independently of the CPU, thus accelerating I/O transactions. Yet DMA, as commonly implemented in the x86 architecture, has three major drawbacks [3]. First, there is no protection from faulty drivers or devices, which might mistakenly or intentionally access memory regions that the device is not allowed to access. Second, DMA is unsuitable for use in virtualization environments by guest virtual machines, since on the one hand the guests cannot know the host physical address of I/O buffers that are utilized by I/O devices, and on the other hand the device is unaware of virtualization and the guest physical address space. Third, in the x86-64 architecture, some legacy I/O devices do not support long addresses and therefore cannot access the entire physical memory [7].

Even though software-based partial solutions such as bounce-buffering and DMA descriptors validation [3,19] address this issue, these solutions introduce additional work for the CPU, do not offer protection from faulty devices, and do not enable DMA usage in virtual guests. Therefore, *DMA Remapping* (DMAR) was introduced in hardware. DMAR is one of the main features of the IOMMU that allows definition of an abstract domain that serves as an isolated environment in the platform, to which a subset of the host physical memory is allocated [1,7]. IOMMU hardware intercepts DMA transactions and utilizes I/O page tables to determine whether the access is permitted and to resolve the actual host physical address that will be accessed. The setup of these I/O page tables is the responsibility of privileged system software (a hypervisor or bare-metal operating system).

According to its usage model, the operating system sets separate DMAR translation tables for different protection contexts, maps virtual I/O memory regions on-demand, and unmaps the region once it is no longer needed. Several strategies for deciding when to map and unmap were proposed [4, 19, 20], yet *single-use mappings* is the common strategy that offers maximal protection. According to this strategy, illustrated in Figure 1, a separate mapping is created for each DMA descriptor and this mapping is unmapped once the corresponding DMA transaction is completed. This scheme is required to prevent the device from using stale mappings and accessing disallowed memory regions. Consequently, a single-use mapping strategy requires recurring mapping and unmapping operations for streaming buffers, which can substantially raise the CPU utilization [4].

Hence, the computational cost of the mapping and unmapping of memory regions in the DMAR units is therefore considered the main bottleneck [3, 19]. One ap-

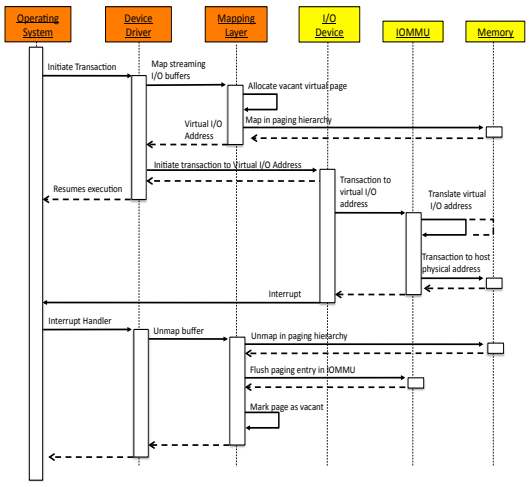



Figure 1: IOMMU Usage-Model

proach for mitigating this bottleneck was improving the free space management of the IOMMU mapping layer in order to decrease the overhead related to mapping and unmapping operations [17]. Another approach is to shorten the unmappings time by performing asynchronous IOTLB flushes during invalidation through the usage of invalidation queues [7].

In addition to the software-induced overhead in IOMMU manipulation, there is also an orthogonal question of whether the IOMMU hardware address resolution mechanism introduces significant overhead. To the best of our knowledge, this work is the first attempt to address this question. As noted before, whenever a DMA access is performed through the IOMMU, the IOMMU translates the virtual I/O address to the machine (host physical) address. To efficiently perform this task, an *I/O Translation Lookaside Buffer* (IOTLB) is included in the IOMMU. However, every IOTLB cache-miss presents high latency as it requires physical address resolution, which is performed by a page-walk through the DMAR paging hierarchy in the main memory. Thus, IOTLB implementation and its usage by operating systems may have significant impact on I/O throughput.

In this work we analyze the impact of software and hardware design choices and implementation on the performance of the IOMMU's DMAR address resolution mechanism. As demonstrated in Section 2, once the computational cost of frequent mapping and unmapping of IOMMU buffers is sufficiently reduced [19, 20], the address resolution mechanism becomes the main bottleneck. We examine the device memory access patterns of various devices and show, for the first time, strategies for reducing the miss-rate of the IOTLB—via pure software modifications or hardware changes with reasonable costs.

The main contributions of this work are as follows:

- We identify the significance of the IOTLB as a potential system bottleneck and demonstrate that it can increase execution time of DMA operations by 47%.
- We present a new methodology for evaluation of I/O device memory access patterns in the presence of an IOMMU. Our method uses virtualization and does not require additional hardware. For achieving this goal we present the vIOMMU— the first virtual IOMMU implementation.
- We analyze actual device memory access patterns and show the resulting bottlenecks. Consequently, we propose software strategies and hardware modifications for reducing IOTLB miss-rates and evaluate their impact on the miss-rate of common workloads, devices, and operating systems.

Section 2 shows that the IOTLB is an actual bottleneck and analyzes the cache-miss impact on the overall throughput; Section 3 analyzes virtual I/O memory access patterns, and Section 4 proposes strategies for reduction of the IOTLB miss-rate and evaluates their impact; Section 5 describes related work,and our conclusions are presented in Section 6.

## 2 IOMMU Performance Analysis

Under regular circumstances, the IOTLB has not been observed to be a bottleneck so far. For several devices, the virtual I/O memory map and unmap operations consume CPU-time, which is greater than the time of the corresponding DMA transaction (data not shown).

Accordingly, to observe the IOTLB bottleneck under normal circumstances, a setup of synthetic configuration was required.

First, to eliminate the time required by the CPU for DMA map and unmap operations, we used the *pseudo pass-through* mode of the IOMMU. This mode works by using a fixed identity mapping in the IOMMU page tables for all of the devices, thus eliminating most of the mapping and unmapping operations and their associated overhead. In addition, this mode uses static mappings — it does not change mappings and does not flush IOTLB entries as a use-once mapping strategy does.

Second, we constructed a stress-test micro-benchmark using a high-speed I/O device. For this purpose our experiments utilized *Intel's I/O Acceleration Technology* (I/OAT) which enables asynchronous DMA memory copy in bandwidths of over 16Gbps per channel [18].

This benchmark goal was to experience different IOTLB miss-rates, according to the IOTLB utilization.

Since no IOTLB flushes occur in the pseudo pass-through mode, mappings could be reused and therefore IOTLB cache entries could be used for subsequent accesses of pages that were previously accessed by the I/O device. Accordingly, IOTLB cache-misses occur when a certain page mapping is evicted before its subsequent use due to cache conflicts.

To directly control the number of IOTLB cache misses caused by cache evictions, we varied the number of source pages (1–256) used for the copy operations, while keeping the total number of copy operations fixed. Hot-spots in IOTLB cache-sets were eliminated by accessing the source pages in a round-robin manner. As a result, IOTLB misses were most likely to occur once the IOTLB cache was fully utilized by the mappings of the source pages, the destination page and the I/OAT descriptors.

Each test was conducted twice—once with IOMMU enabled and once with IOMMU disabled—and the experienced execution time penalty was calculated. In addition, we used various block sizes as a copy source, expecting that the effect of a cache miss will be more pronounced compared to the memory copy overhead when small blocks are copied. To confirm that we managed to saturate the IOMMU and measure the actual execution time penalty imposed by its hardware, in each configuration we asserted that the processor is idle, after all DMA setup operations were done.

The experiments were conducted using Intel Xeon X5570 running at 2.93 GHz with the Intel X58 chipset. All the DMA copies were conducted using a single I/OAT channel. The experiment results are shown in Figure 2.

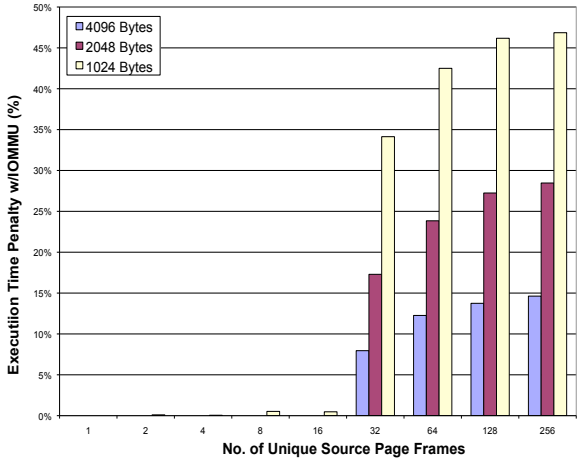



Figure 2: Execution time penalty due to IOMMU for DMA memory copy operations of various sizes

As can be easily seen, the execution time penalty imposed by the IOMMU remains very low while the number of source pages is lower or equal to 16. In contrast, a copy operation from 32 source pages increased the execution time by 8% for a whole 4KB page and by 34% for 1KB blocks. Increasing the number of pages from 32 has a lesser effect afterward, yet keeps increasing the execution time by up to 15% for a whole 4KB page and 47% for 1KB blocks when using 256 different pages as the source. Thus, we conjecture that these observed penalties are a result of IOTLB misses. Apparently, when 32 pages are used as sources, the IOTLB exceeds its capacity, resulting in IOTLB thrashing.

## 3 Virtual I/O Memory Access Patterns

The results of the execution time penalty associated with IOTLB misses show there should be substantial room for improvement of the IOTLB which would improve the throughput and latency of DMA operations by reducing the miss-rate. To propose strategies for such improvements, we investigated the virtual I/O memory access patterns of common devices. Therefore, we evaluated several devices configurations that are expected to behave differently and expose a representative variety of memory patterns and problems, thereby resulting in an educated proposal of IOTLB miss-rate reduction strategies.

### 3.1 vIOMMU

Currently, the common techniques for analyzing DMA transactions involve the usage of dedicated hardware such as PCI Pamette [14]. However, such hardware devices are not always available for researchers. In contrast, the evaluation methodology presented here uses virtualization for capturing a trace of I/O devices' memory access patterns, without any additional hardware. We implemented this methodology for the KVM hypervisor [11].

Primary to our evaluation methodology is the implementation of a "virtual IOMMU", vIOMMU. An operating system is run on a virtual machine, and the hypervisor captures all of the interactions between the operating system, its hardware devices, and the virtual platform's IOMMU. To capture realistic interactions, the vIOMMU implements the same hardware interfaces as Intel's VT-d IOMMU, in the same manner in which an emulated virtual device implements the interfaces as a real hardware device [2, 16]. Thus the operating system interacts with vIOMMU in exactly the same way that it interacts with Intel's VT-d IOMMU when running on bare-metal.

vIOMMU's implementation components and data structures are illustrated in Figure 3. First, vIOMMU emulates the IOMMU registers, enabling write operations of the guest to the registers, and returns the expected value

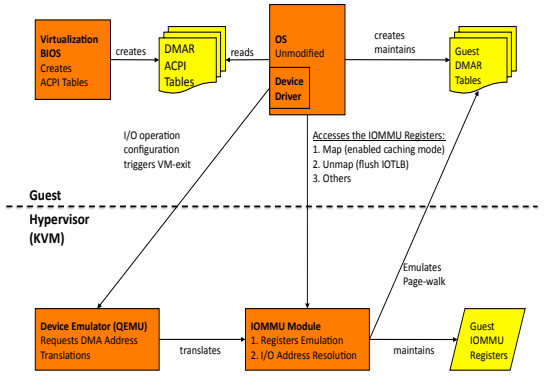



Figure 3: vIOMMU implementation modules and data structures

upon read access. Second, we modified the virtual BIOS code to set up *Advanced Configuration and Power Interface* (ACPI) tables for the DMAR and *DMA Remapping Hardware Unit Definition* (DRHD) structures. Last, we adapted the emulation code of the DMA read/write operations of various devices: E1000 NIC [6], LSI 53C895A SCSI [12] and IDE. Like the IOMMU hardware, our implementation intercepted DMA operations, and, prior to these accesses, performed proper machine address resolution according to the IOMMU virtual registers, the virtual device number and function number, and the guest's DMAR translation structures. These modifications enabled us to run unmodified Linux kernels in virtual machines, which access an IOMMU and program it according to their needs.

To log traces of virtual I/O memory related operations, in the hypervisor we traced each IOMMU access—every DMAR operation of mapping and unmapping executed by the OS, and every read and write DMA access executed by the emulated devices. The time axis in Figures 4, 5, 6 and 7 is measured in discrete virtual time. We advance to the next time step whenever an IOMMU access occurs.

Using vIOMMU we executed various benchmarks in the guest with virtual (emulated) devices. These traces were later analyzed through a TLB emulator in which we implemented the proposed strategies for lowering the IOTLB miss-rate and compared the various approaches.

Experiments were performed using Linux 2.6.31 as the guest operating system. For the experiments we used the mapping layer's strict mode, which performs immediate page specific invalidations when unmapping is requested. Our hypervisor implementation was based on KVM-88 [11].

## 3.2 Analysis of Virtual I/O Memory Access Patterns

Analysis of the results is aided by knowledge of the implementation of the relevant device drivers. As shown in Figures 5, 6 and 7, Linux's IOMMU driver starts mapping virtual memory I/O at a certain virtual address and continues in descending order.

An additional distinction can be observed with respect to the following two kinds of DMA mappings employed by Linux [13]:

- *Consistent DMA mappings* (sometimes referred to as *Coherent DMA Mappings*) are *persistent* mappings that are usually mapped once at driver initialization and unmapped when the driver is removed. These mappings are used by network cards for DMA ring descriptors, SCSI adapter mailbox command data structures, etc.

- *Streaming DMA mappings* are *ephemeral* mappings that are usually mapped for one DMA transaction and unmapped as soon as it completes. Such mapping are used for networking buffers transmitted or received by NICs and for file-system buffers written or read by a SCSI device.

As shown in Figure 5 consistent DMA mappings are performed by the device drivers before any streaming DMA mappings. In addition, the consistent DMA mapping region is consistently and rapidly accessed, as evident in Figure 4 and 6. These characteristics are expected to repeat for most I/O device drivers and operating systems since they stem from the different functions performed by the mapped regions (e.g., DMA ring descriptors vs. transient buffers).

As for the streaming DMA mappings, two methods for mapping its memory are available:

- *Scatter-gather list mapping* - Scatter-gather (vectored I/O) is used to map a non-physically contiguous memory region. In this case, a list of memory pages is delivered to the DMA mapping layer, which enables the usage of multiple buffers. The LSI SCSI driver uses such mappings. As apparent in Figure 6, the result of the Linux implementation is that consecutive pages in the list are allocated to mappings in ascending order within each scatter list.

- *Contiguous physical memory mapping* - Devices that do not support scatter-gather lists require the usage of contiguous memory mappings. Some device drivers that use these mappings map each page separately. As apparent in Figure 4, mappings in such manner results in descending addresses of the virtual I/O page frame.

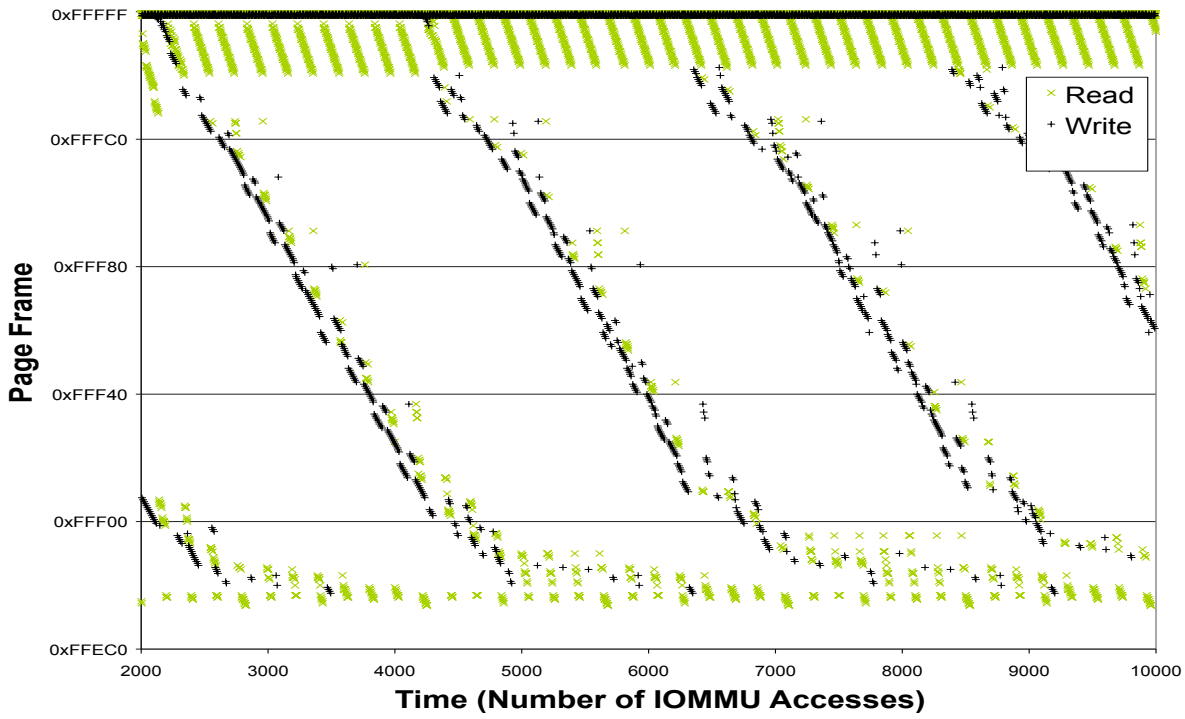



Figure 4: E1000 NIC—`netperf` send—virtual I/O memory access

In the case of multiple devices that issue simultaneous DMA transactions, each device has its own separate virtual I/O memory space (its own "domain"); thus, its utilized regions can overlap with other devices' mapped virtual I/O memory regions This scheme may result in multiple devices that access the same virtual I/O page in a separate virtual I/O address space. This behavior was demonstrated when two E1000 NIC were simultaneously used as shown in Figure 7. This issue is further studied in Section 4.2.

It is apparent in the virtual I/O memory access pattern of the two E1000 NIC configuration which is demonstrated in Figure 7 that the two virtual I/O memory regions in use overlap. Indeed, according to the Linux IOMMU mapping layer implementation it is clear the same virtual I/O pages are likely to be accessed in general by multiple devices. Specifically, the consistent DMA mappings of all the devices are evidently mapped in the same few virtual I/O pages. Possible issues of such mappings are further described in detail and studied in Section 4.2.

# 4 IOTLB Miss-Rate Reduction Approaches

To increase DMA throughput, the number of IOMMU IOTLB misses should be decreased, as each miss requires a new page-walk for translation that can result in several consecutive accesses to memory. Until the translation is done and the physical host address is resolved, a DMA transaction cannot be completed, thereby increasing latency. Next, on the basis of the device virtual I/O memory access patterns, we considered several strategies likely to decrease the miss-rate.

## 4.1 Streams Entries Eager Eviction

Streaming DMA mappings, such as those used by NICs and SCSI devices, are likely to be mapped for the duration of a single DMA transaction [13]. Therefore, caching of these entries is likely to have only a small positive impact on lowering the IOTLB miss-rate. Moreover, caching of these entries may cause evictions of consistent DMA mappings that are cached in the IOTLB and even increase the IOTLB miss-rate.

Intel approached this issue by suggesting Eviction Hints (EH) [7]. Once EH is enabled, pages that are marked as *transient mappings* (TM) in the page-table can be eagerly evicted by the IOMMU hardware as needed.

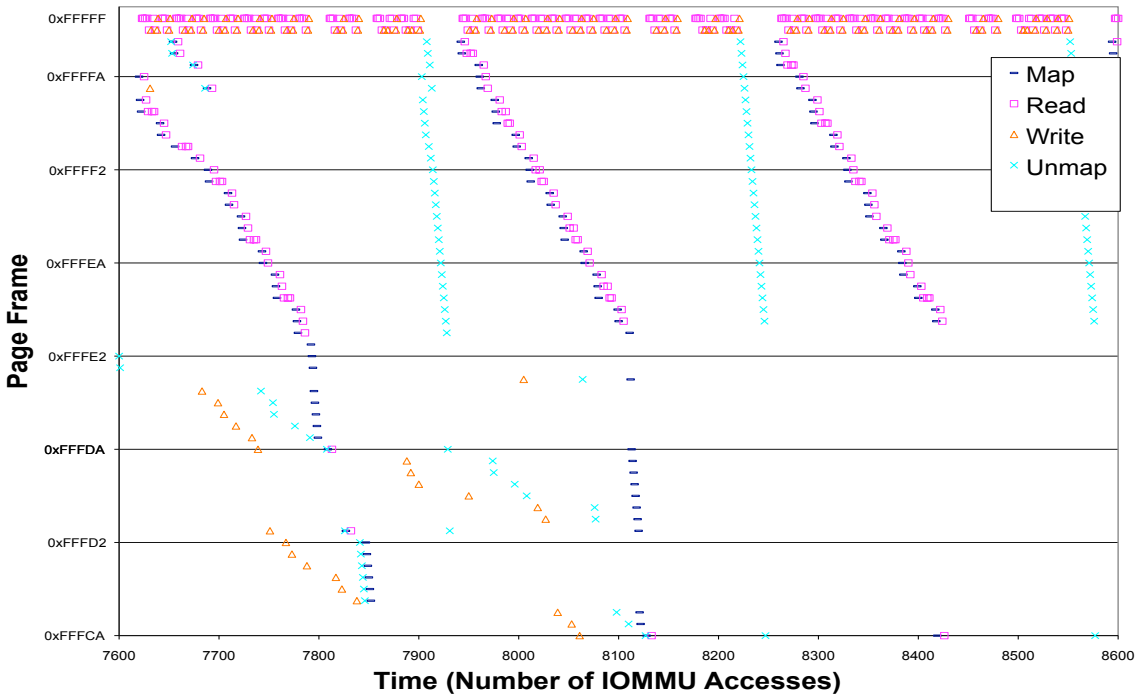



Figure 5: E1000 NIC—`netperf` send—map/unmap operations

Accordingly, a simple approach is to mark all streaming DMA mappings as TM, assuming they are likely to be accessed by the I/O device only once. The EH mechanism should not contradict the prefetch of streaming DMA mappings into a prefetch buffer where eviction hints have no effect. We note that the EH mechanism complements the prefetching of streaming DMA mappings into a prefetch buffer, as described in Section 4.5. Prefetching helps because the mapping is prefetched before it is used; EH helps because the mapping is discarded as soon as it is used, making room for other mapping

## 4.2 Non-Overlapping Coherent Frames

Unlike streaming DMA mappings, consistent DMA mappings that are mapped at the device driver initialization and unmapped when the driver is removed, are likely to be accessed frequently. According to the method used in the IOTLB for determining each frame's IOTLB set, it is desirable that those entries be evenly distributed.

No public data was available for us to see how the IOTLB set is determined for each mapped frame, yet the simplest scheme is to determine the set according to the virtual I/O address, without taking into consideration the originating device or address space (domain). Since operating systems such as Linux allocate virtual I/O regions from a certain virtual I/O address for each domain, and since most of the drivers perform coherent memory mappings before they perform the streaming DMA mappings, it is likely that the coherent page frames will not be evenly distributed. Under these conditions, when multiple devices are in use and each has its own coherent mapping, hot-spots will appear in some of the IOTLB sets, causing rapid evictions and IOTLB thrashing, and resulting in a higher miss-rate.

To address this issue, and since the virtual I/O addresses for both consistent and streaming DMA mapping are allocated from the same pool, we propose *virtual I/O page-coloring*, i.e., offsetting each device's virtual I/O address space by a different number of frames. The offset can either be determined by the number of frames for coherent mappings that were previously allocated to other devices, or by a fixed number of frames. This solution does not require any hardware modification.

## 4.3 Large TLB and Higher TLB Associativity

Obviously, the greater the number of entries in the IOTLB, the less likely it is that caching a new entry in the IOTLB will cause an eviction of another entry that will be used later. Enlarging the IOTLB can be done by either

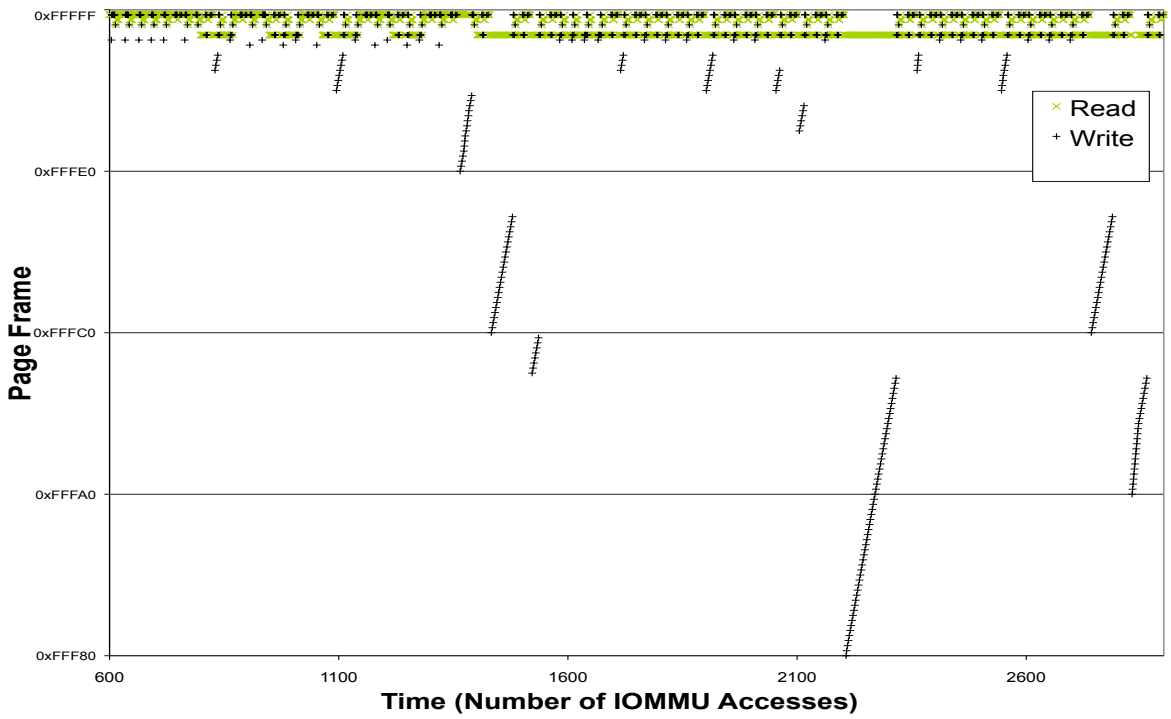



Figure 6: SCSI controller - `bonnie++` write—virtual I/O memory access

increasing the number of sets or increasing the IOTLB associativity. However, since streaming DMA mappings are only cached for a short while and show spatial locality IOTLB thrashing is less likely to occur. Increasing the number of IOTLB entries increases the complexity and cost of implementation in hardware.

## 4.4 Super-Pages

For a long time, the MMU *super-pages* strategy has been argued to improve TLB coverage and relieve TLB pressure [5,15]. In the super-page strategy, the operating system can use multiple page-sizes according to its needs. Accordingly, fewer TLB entries are utilized when the bigger page-size is in use, and evictions are less likely to occur.

Both Intel's VT-d and AMD's Pacifica IOMMU architecture support multiple IOMMU page sizes other than 4KB, 2MB, 1GB, 512GB, and 256TB [1, 7], and AMD's architecture also supports additional page sizes. However, AMD specifically notes that implementations are free to cache translations of super-pages by splitting them into multiple 4KB cache entries. Obviously, the IOTLB miss-rate is not likely to improve, unless the super-page is cached as a whole in one entry.

The main drawback of super-pages usage in the IOMMU is the coarse protection granularity the IOMMU can offer. The bigger the mapped page, the bigger the area of contiguous memory inside its protection domain. Any entity (device) that has access to any memory buffer inside that area, by definition then has access to all of the other memory inside that area. Therefore, the bigger the mapped page, the more likely it is that one device I/O buffer might reside with another device I/O buffer or operating system code within the same super-page.

Due to the coarse granularity of the protection offered by super-pages, and since the number of TLB entries dedicated for super-pages is usually low, the use-once usage model of the IOMMU does not seem to use super-pages efficiently as many opportunities to share IOTLB entries are lost. One usage model that seems to fit super-pages is the shared usage-model [19] in which mappings are shared among DMA descriptors that point to the same physical memory.

This shared usage-model was used for our evaluation. Figure 8 and Figure 9 demonstrate the desired cache set size, which is the number of IOTLB entries in use in E1000/`netperf` and SCSI/`bonnie++` benchmarks, respectively. As it appears, in both configurations, the use of super-pages does in fact drastically decrease the number of entries required in the IOTLB, indicating that IOTLB miss-rates substantially decrease when super-pages are in use.

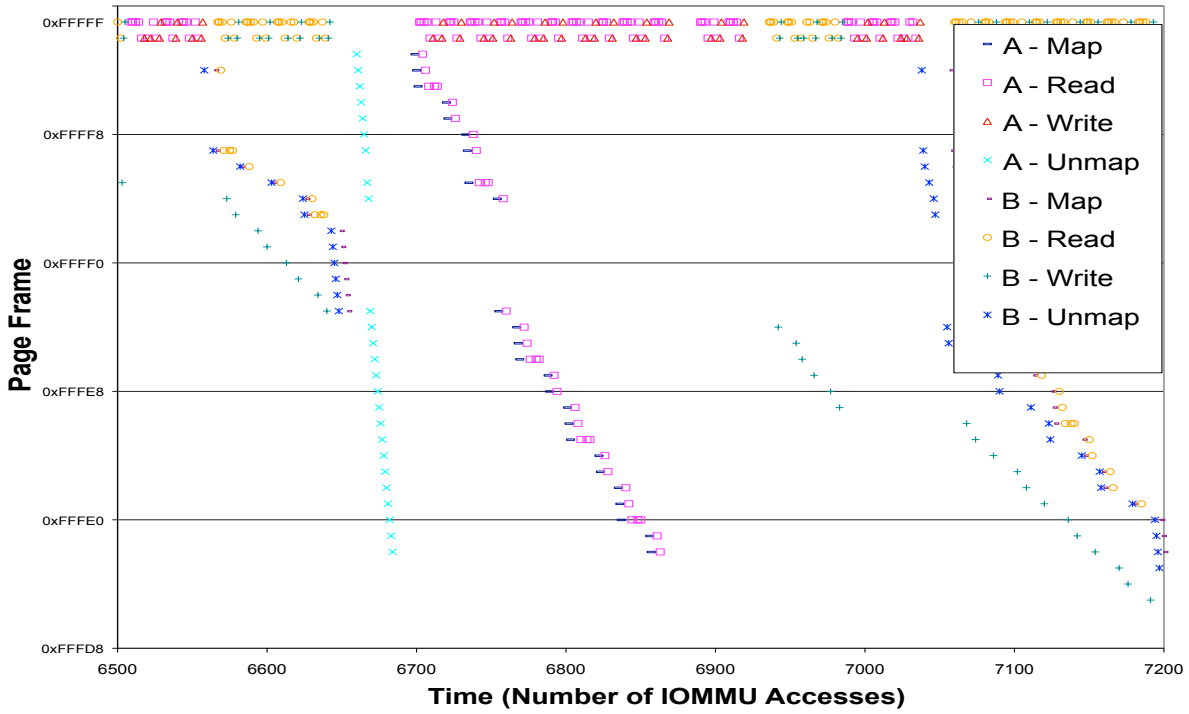



Figure 7: Two E1000 NICs—`netperf` send—map/unmap operations

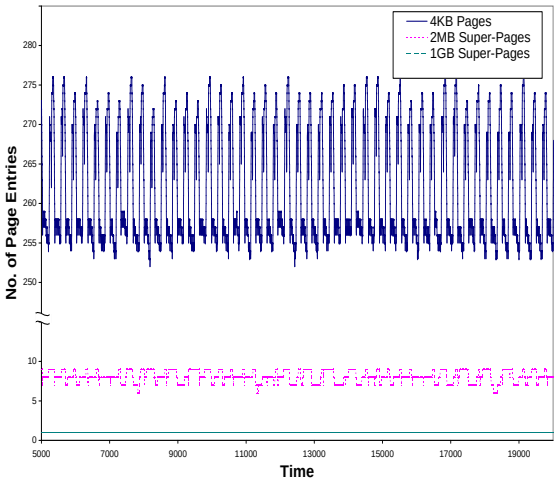



Figure 8: Number of IOTLB entries in use—E1000/`netperf`

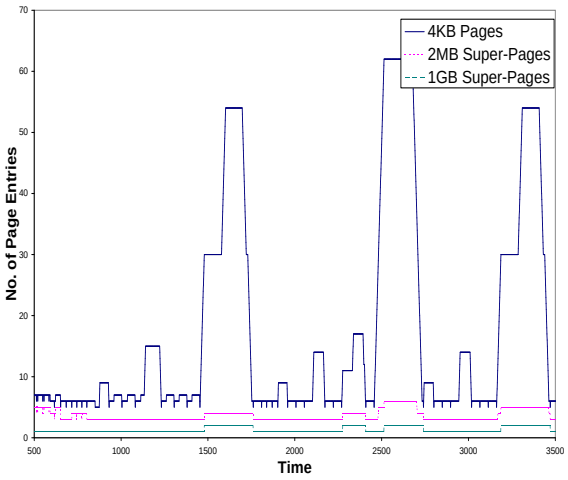



Figure 9: Number of IOTLB entries in use—SCSI/`bonnie++`

## 4.5 Prefetching Techniques

As cache structure improvements have only a limited opportunity to reduce the miss-rate, it is important to reduce the latency incurred on a miss by reducing or hiding some or all of its cost. Prefetch is one of the popular approaches for this issue, widely used in TLBs [8].

Prefetching techniques can be divided into two distinct categories: techniques that are designed for strided reference patterns, and techniques that base their decisions on history [10]. Methods intended for strided reference patterns are likely to benefit from the spatial locality of DMA accesses, shown in the examples in Figures 4 and 6.

In contrast, we argue that methods that base their decision on history, whose hardware implementation is inherently more complicated and thus have a higher cost, are not likely to show a real benefit. This, we argue, is due to the fact that mappings are usually used only once before they are unmapped and their next use is not likely to be related to the current use.

## 4.6 Adjacent Mappings Prefetch

Under certain conditions, streaming DMA mappings of devices are likely to be accessed with some level of spatial address locality. The first condition is that the virtual I/O addresses allocated by the mapping layer show spatial locality. In Linux, we can induce from the mapping layer implementation that this is indeed the case for pages of blocks within a scatter-gather list and for mappings of a contiguous block that spans multiple pages. From our experience, the red-black tree allocation mechanism also demonstrates spatial locality in the general case when the virtual I/O memory is lightly fragmented, as shown for the E1000 NIC `netperf` benchmark in Figure 5.

The second condition is that the hardware device accesses the streaming memory mappings in an orderly fashion. This is the case for high-throughput devices with small payload requests such as NICs [7]. It can be argued that hardware offloading mechanisms make devices more likely to access memory mappings in an orderly fashion.

For the cases where those two conditions are fulfilled, Intel suggests the usage of Address Locality Hints (ALH) [7]. When supported, each I/O device can be marked to note whether it is likely to demonstrate spatial locality, which may be used by the IOMMU hardware to prefetch adjacent mappings. This mechanism resembles other streaming buffers prefetches mechanisms [8].

However, it is questionable whether both higher and lower adjacent mappings should be prefetched, as the direction is likely to be predetermined. As mentioned in Section 3.2, in Linux pages of a certain mapping get ascending addresses, whereas the subsequent mapping get a lower address. Therefore, during the mapping of an I/O virtual page, the mapping layer already knows the location of the subsequent I/O page. Therefore, we propose a variant of the ALH mechanism—*Single-Direction Address Locality Hints* (SD-ALH): An ALH mechanism with domain context level or page table leaves hints that mark whether the ALH mechanism should prefetch higher adjacent pages or lower adjacent pages.

Multiple pages mapping through a single call raises another issue with the existing ALH implementation. The result of multiple pages mappings, as can be seen in the LSI SCSI `bonnie++` benchmark in Figure 6, is that even if there are no gaps within the virtual I/O address space due to alignment, the last virtual I/O page frame in a certain scatter-gather list and the first frame in the subsequent list will *not* be adjacent. Thus, the first virtual I/O page frame in the subsequent list cannot be prefetched under this scheme. Therefore, it is preferable to modify the mapping layer implementations so that multiple pages mappings are performed in descending order as well.

## 4.7 Explicit Caching of Mapped Entries

Streaming DMA mappings are usually mapped for a single DMA transaction. Therefore, it is likely that the first access to a streaming mapping will cause a miss unless it is prefetched. The adjacent mappings prefetch mechanism can usually perform a prefetch of these pages, yet it is likely not to prefetch the translation of the next frame in use in certain common situations:

- The next adjacent frame is still not mapped when the last mapped frame is accessed.
- Several groups of pages are mapped, when no spatial locality is demonstrated between the groups. Such group of pages is mapped when one maps a scatter-gather list or a contiguous physical memory which consists several pages. This scenario can be observed in the case of SCSI LSI `bonnie++` benchmark in Figure 6.

Therefore, we propose the *Mapping Prefetch* (MPRE) approach. With this approach, the operating system explicitly hints the IOMMU hardware to prefetch the first mapping of each group of streaming DMA mappings, where a group is defined as a scatter-gather list or several contiguous pages that are mapped consecutively. The number of additional unnecessary prefetches is likely to be negligible as only the first mapping of each group is explicitly prefetched under this approach.

## 4.8 Evaluation of Strategies

We performed a trace-driven simulation of the IOTLB to evaluate our proposed approaches for reducing the IOTLB miss-rate. It should be noted that except for the architecture specifications, no data was published regarding the structure of Intel's and AMD's IOMMU. Therefore, we could not use their existing IOTLB designs as base-lines for the evaluation. Instead, our base-line configuration was of a reasonable default IOTLB design: 32 entries, a two-way cache, a *least recently used* (LRU) eviction policy, and four entries in a fully-associative prefetch-buffer. We evaluated the effect of the proposed methods on the miss-rate of an E1000 NIC device running `netperf`, a SCSI device running the `bonnie++`

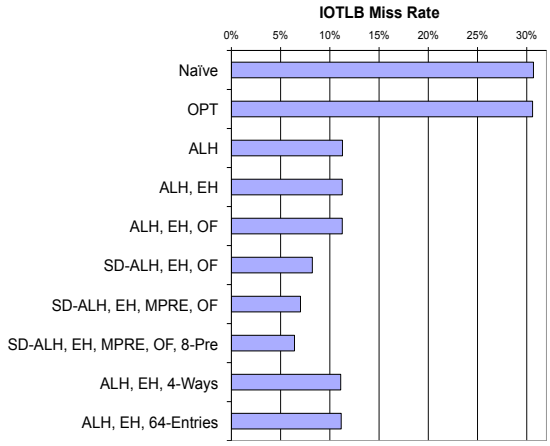



Figure 10: Miss-rate of different configurations - E1000/`netperf`

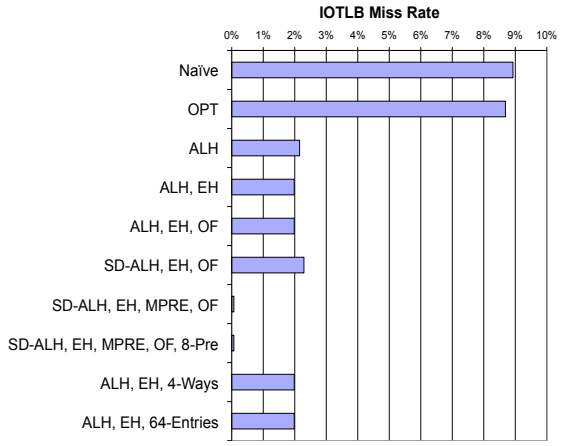



Figure 11: Miss-rate of different configurations - SCSI/`bonnie++`

write test, and two E1000 devices running `netperf` concurrently.

In addition, we evaluated the optimal eviction policy (OPT-EV) algorithm as a reference. This algorithm has complete knowledge of the future access sequence and evicts the IOTLB entry that will be used furthest in the future. It can be proved that no other eviction policy can perform better than this reference algorithm.

The strategies notation is as follows:

- OPT-EV—Optimal evictions policy (without prefetches)
- EH—Evictions hint—see Section 4.1
- ALH—Address locality hints—see Section 4.6
- SD-ALH—Single-direction address locality hints—see Section 4.6
- OF—Offsetting coherent mappings—see Section 4.2
- 8-Prefetch—Eight entries are set in the prefetch buffer instead of four
- MPRE—Mapping prefetch—See Section 4.7

## 4.9 Discussion

As seen in the simulation results of the various configurations in Figures 10, 11, and 12, the current miss-rate in these scenarios can easily be reduced by 30% by enabling features in the chipset that are already defined as well as performing some reasonable software modifications. An additional reduction of 20% is also possible by making relatively simple hardware modifications. In contrast, increasing the number of IOTLB entries or ways number showed relatively little improvement.

Apparently, the two strategies that reduced the miss-rate the most are address locality hints (ALH) and intelligent prefetch of mappings right after they are mapped (MPRE). Offsetting coherent mappings also had a significant positive impact in the case of multiple NIC devices, and as expected had neither a positive nor negative impact in the cases of single SCSI or NIC devices.

As is apparent in the simulation results, application of the optimal eviction policy (OPT-EV) without an additional prefetch technique does not reduce the miss-rate significantly. This is a clear indication that no eviction policy can substitute for prefetch techniques in substantially reducing the miss-rate. This result is in accordance with the use-once usage-model in which stream-buffer mappings are very unlikely to reside in the IOTLB unless they were prefetched. The main goal of an intelligent eviction policy should be to keep coherent mappings in the cache, and evict other mappings instead, yet the eviction policy enabled by Eviction Hints, which is intended for this matter, had little positive impact on the hit-rate, if any. In fact, in the case of multiple NICs, it even resulted in a lower hit-rate.

The eviction policies are not affected by prefetches, as prefetched entries are traditionally kept in a separate buffer until they are actually used. Accordingly, in contrast with the minor potential that even the optimal eviction policy offers, increasing the number of entries in the mapping prefetch buffer increased the hit-rate for NIC devices. As the SCSI device access pattern was fairly fixed, only a small number of prefetch entries were actually utilized and increasing the number of prefetch entries had no impact in this case.

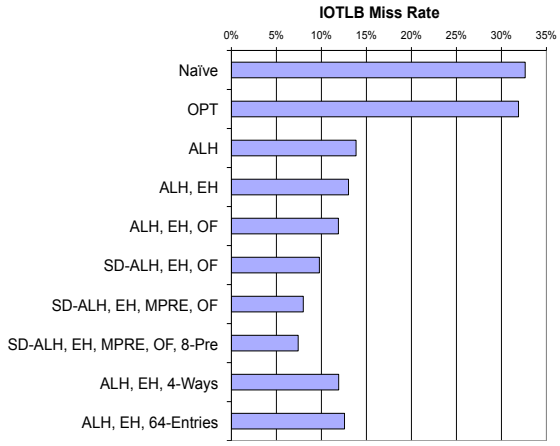



Figure 12: Miss-rate of different configurations - 2 x E1000/`netperf`

## 5 Related Work

Most IOMMU research to date has concentrated on the CPU-load caused by the mappings and unmappings of I/O buffers, IOMMU flushes, and related interrupts handling. Ben-Yehuda *et al.* [4] evaluated the performance of the Calgary and DART IOMMUs in native mode, and acknowledged and reviewed the possible effect of IOMMU hardware design on the performance of IOMMUs. In their work they concluded that most of the overhead is due to the software implementation, yet do not quantify the hardware-induced overhead.

Unlike IOMMU's IOTLBs, MMU's TLBs were widely researched. Kandiraju and Sivasubramaniam researched the miss-rate of applications in the Spec CPU2000 benchmarks suite by executing them on an architectural simulator [9]. One interesting result in light of our research is that a comparable d-TLB configuration resulted in miss-rate of less than 5% for all but two of the applications in the benchmark suite. This miss-rate is considerably lower than the 32% we observed for the IOTLB.

## 6 Conclusions

We presented for the first time an investigation of IOMMU IOTLBs and the bottleneck imposed by their address resolution mechanism. Our evaluation of memory access patterns resulted in several strategies that reduce the miss-rate by 50% and can be relatively easily implemented in software without any expected negative side-effects. First, enabling the ALH feature in IOMMU drivers; Second, mapping multiple pages of a single buffer in descending order; and third, offsetting virtual I/O memory to avoid hot-spots of IOTLB sets used by different devices for consistent memory mappings. We also propose additional methods that we believe are easy to implement in hardware: explicit prefetch of mapped entries and refinements of the ALH mechanism for configuring the prefetch direction of adjacent pages by the IOMMU device driver.

## References

[1] AMD. IOMMU architectural specification. `http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/34434.pdf`.

[2] BELLARD, F. QEMU, a fast and portable dynamic translator. *ATEC '05: Proceedings of the Annual Conference on USENIX*, 41–41 (2005).

[3] BEN-YEHUDA, M., MASON, J., XENIDIS, J., KRIEGER, O., VAN DOORN, L., NAKAJIMA, J., MALLICK, A., AND WAHLIG, E. Utilizing IOMMUs for virtualization in Linux and Xen. In *OLS '06: The 2006 Ottawa Linux Symposium* (July 2006), pp. 71–86.

[4] BEN-YEHUDA, M., XENIDIS, J., OSTROWSKI, M., RISTER, K., BRUEMMER, A., AND VAN DOORN, L. The price of safety: Evaluating IOMMU performance. In *OLS '07: The 2007 Ottawa Linux Symposium* (July 2007), pp. 9–20.

[5] HILL, M. D., KONG, S. I., PATTERSON, D. A., AND TALLURI, M. Tradeoffs in supporting two page sizes. Tech. rep., Mountain View, CA, USA, 1993.

[6] Linux 2.6.31:drivers/Documentation/networking/e1000.txt.

[7] INTEL. Intel virtualization technology for directed I/O, architecture specification. `http://download.intel.com/technology/computing/vptech/Intel(r)_VT_for_Direct_IO.pdf`.

[8] JOUPPI, N. P. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. *SIGARCH Comput. Archit. News 18*, 3a (1990), 364–373.

[9] KANDIRAJU, G. B., AND SIVASUBRAMANIAM, A. Characterizing the d-TLB behavior of SPEC CPU2000 benchmarks. *SIGMETRICS Perform. Eval. Rev. 30*, 1 (2002), 129–139.

[10] KANDIRAJU, G. B., AND SIVASUBRAMANIAM, A. Going the distance for TLB prefetching: An application-driven study. *Computer Architecture, International Symposium on 0* (2002), 0195.

[11] KIVITY, A., KAMAY, Y., LAOR, D., LUBLIN, U., AND LIGUORI, A. KVM : the Linux Virtual Machine Monitor. *Proceedings of the Linux Symposium, Ottawa, Ontario, 2007* (2007).

[12] LSI53C895A PCI to ultra2 SCSI controller technical manual. `http://www.lsi.com/DistributionSystem/AssetDocument/files/docs/techdocs/storage_stand_prod/SCSIControllers/lsi53c895a_tech_manual.pdf`.

[13] MILLER, D. S., HENDERSON, R., AND JELINEK, J. Linux 2.6.31:Documentation/DMA-mapping.txt.

[14] MOLL, L., AND SHAND, M. Systems performance measurement on PCI pamette. In *FPGAs for Custom Computing Machines, 1997. Proceedings., The 5th Annual IEEE Symposium on* (Apr 1997), pp. 125–133.

[15] NAVARRO, J., IYER, S., DRUSCHEL, P., AND COX, A. Practical, transparent operating system support for superpages. In *OSDI '02: Proceedings of the 5th symposium on operating systems design and implementation* (New York, NY, USA, 2002), ACM, pp. 89–104.

[16] SUGERMAN, J., VENKITACHALAM, G., AND LIM, B.-H. Virtualizing I/O devices on VMware workstation's hosted virtual machine monitor. In *USENIX Annual Technical Conference* (Berkeley, CA, USA, 2001), USENIX Association.

[17] TOMONORI, F. DMA representations sg_table vs. sg_ring IOMMUs and LLDś restrictions. LSF 08 `http://iou.parisc-linux.org/lsf2008/IO-DMA_Representations-fujita_tomonori.pdf`.

[18] VAIDYANATHAN, K., HUANG, W., CHAI, L., AND PANDA, D. K. Designing efficient asynchronous memory operations using hardware copy engine: A case study with I/OAT. In *21th International Parallel and Distributed Processing Symposium (IPDPS 2007), Proceedings, 26-30 March 2007, Long Beach, California, USA* (2007), IEEE, pp. 1–8.

[19] WILLMANN, P., RIXNER, S., AND COX, A. L. Protection strategies for direct access to virtualized I/O devices. In *ATC'08: USENIX 2008 Annual Technical Conference on Annual Technical Conference* (Berkeley, CA, USA, 2008), USENIX Association, pp. 15–28.

[20] YASSOUR, B.-A., BEN-YEHUDA, M., AND WASSERMAN, O. On the DMA mapping problem in direct device assignment. In *SYSTOR '10: The 3rd Annual Haifa Experimental Systems Conference* (2010).