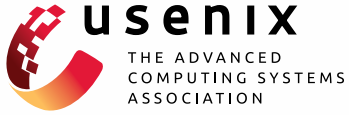

Boosting GPU Virtualization Performance with Hybrid Shadow Page Tables

Yaozu Dong and Mochi Xue, ***Shanghai Jiao Tong University and Intel Corporation;***
Xiao Zheng, ***Intel Corporation*****; Jiajun Wang,** ***Shanghai Jiao Tong University and Intel Corporation;*** **Zhengwei Qi and Haibing Guan,** ***Shanghai Jiao Tong University***

https://www.usenix.org/conference/atc15/technical-session/presentation/dong

This paper is included in the Proceedings of the 2015 USENIX Annual Technical Conference (USENIC ATC '15).

July 8–10, 2015 • Santa Clara, CA, USA

ISBN 978-1-931971-225

Open access to the Proceedings of the 2015 USENIX Annual Technical Conference (USENIX ATC '15) is sponsored by USENIX.

Boosting GPU Virtualization Performance with Hybrid Shadow Page Tables

Yaozu Dong[1, 2], Mochi Xue[1,2], Xiao Zheng[2], Jiajun Wang[1,2], Zhengwei Qi[1], Haibing Guan[1]
{*eddie.dong, xiao.zheng*}*@intel.com* {*xuemochi, jiajunwang, qizhenwei, hbguan*}*@sjtu.edu.cn*
[1]*Shanghai Jiao Tong University,* [2]*Intel Corporation*

Abstract

The increasing adoption of Graphic Process Unit (GPU) to computation-intensive workloads has stimulated a new computing paradigm called GPU cloud (e.g., Amazon's GPU Cloud), which necessitates the sharing of GPU resources to multiple tenants in a cloud. However, state-of-the-art GPU virtualization techniques such as gVirt still suffer from non-trivial performance overhead for graphics memory-intensive workloads involving frequent page table updates.

To understand such overhead, this paper first presents GMedia, a media benchmark, and uses it to analyze the causes of such overhead. Our analysis shows that frequent updates to guest VM's page tables causes excessive updates to the shadow page table in the hypervisor, due to the need to guarantee the consistency between guest page table and shadow page table. To this end, this paper proposes gHyvi[1], an optimized GPU virtualization scheme based on gVirt, which uses adaptive hybrid page table shadowing that combines strict and relaxed page table schemes. By significantly reducing trap-and-emulation due to page table updates, gHyvi significantly improves gVirt's performance for memory-intensive GPU workloads. Evaluation using GMedia shows that gHyvi can achieve up to 13x performance improvement compared to gVirt, and up to 85% native performance for multi-thread media transcoding.

1 Introduction

The emergence of HPC cloud [30] has shifted many computation-intensive workloads such as machine learning [24], molecular dynamics simulations [31] and media transcoding to cloud environments. This necessitates the use of GPU to boost the performance of such computation-hungry applications, resulting in a new computing paradigm called GPU cloud (such as Amazon's GPU cloud [2]). Hence, it is now vitally important to provide efficient GPU virtualization to provision elastic GPU resources to multiple users.

To address this challenge, two recent full GPU virtualization techniques, gVirt [29] and GPUvm [28], are proposed respectively. gVirt is the first open-source product-level full GPU virtualization approach based on Xen hypervisor [11] for Intel GPUs, while GPUvm provides a Graphic Process Unit (GPU) virtualization approach on the NVIDIA card. This paper mainly focuses on gVirt due to its open-source availability. Specifically, gVirt presents a vGPU instance to each VM to run native graphics driver, which achieves high performance and good scalability for GPU-intensive workloads.

While gVirt has made an important first step to provide full GPU virtualization, our measurement shows that it still incurs non-trivial overhead for media transcoding workloads. Specifically, we build GMedia using Intel's MSDK (Media Software Development Kit) to characterize the performance of gVirt. Our analysis uncovers that gVirt still suffers from non-trivial performance slowdown due to an issue called *Massive Update Issue*. This is caused by frequent updates on guest page tables, which lead to excessive VM-exits to the hypervisor to synchronize the shadow page table with the guest page table.

To address the Massive Update Issue, this paper introduces gHyvi, which provides a hybrid page table shadowing scheme to provide optimized full GPU virtualization based on Xen hypervisor for Intel GPUs. Inspired by the GPU programming model, we introduce a new asynchronous mechanism, namely relaxed page table shadowing, which removes trap-and-emulation and thus reduces the overhead of massive page table's modifications. To minimize the overhead of making guest and shadow page tables consistent, we combine the two mechanisms into a adaptive hybrid page table shadowing scheme, which take advantage of both the traditional strict and the new relaxed page table shadowing. When

[1]The source code of gHyvi will be available at `https://01.org/igvt-g`.

there are infrequent page table accesses, gHyvi works in strict page table shadowing; once the gHyvi detects the guest VM is frequently updating the page table, it will switch to the relaxed page table shadowing.

One critical issue of using the relaxed page table shadowing scheme is to reconstruct the shadow pages when shadow pages are inconsistent with guest pages. To better understand the tradeoff of different reconstruction policies, we implement and evaluate four page table reconstruction policies: full reconstruction, static partial reconstruction, dynamic partial reconstruction and dynamic segmented partial reconstruction. Our analysis shows that the last one usually has better performance than the others, which is thus used as the default policy for gHyvi.

We have implemented gHyvi based on gVirt, which comprises 600 LoCs. Experiments using GMedia on an Intel GPU card show that gHyvi can achieve up to 13x performance improvement compared to gVirt, and up to 85% native performance for multi-thread media transcoding. Our analysis shows that gHyvi wins due to the reduction of up to 69% VM-exits.

In summary, this paper makes the following contributions:

- A GPU-enabled benchmark for media transcoding performance (GMedia), by invoking functions from Intel MSDK to evaluate and collect the performance data on Intel's GPU platforms.
- A relaxed page table shadowing mechanism as well as a hybrid shadow page table scheme, which combines the strict page table shadowing with the relaxed page table shadowing.
- Four reconstruction policies: the full reconstruction policy, static partial reconstruction policy, dynamic partial reconstruction policy, and the dynamic segmented partial reconstruction policy for relaxed page table shadowing mechanism.
- An evaluation showing that gHyvi achieves up to 85% native performance for multi-thread media transcoding and a 13x speedup over gVirt.

The rest of the paper is organized as follows: Section 2 describes some background information on gVirt and GPU programming model. Section 3 presents our benchmark for media transcoding and discusses the Massive Update Issue in detail, followed by the design and implementation of gHyvi In section 4. Then, section 5 evaluates the gHyvi and section 6 discusses the related work. Finally, section 7 concludes with a brief discussion on future work.

2 Background

2.1 GPU for Computing

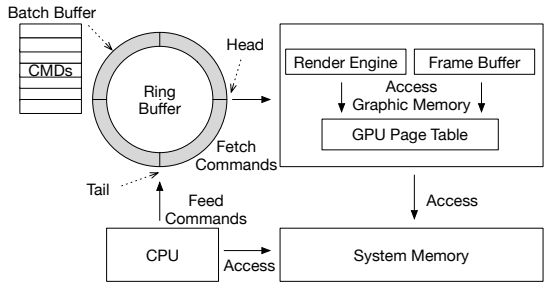

Figure 1: GPU Programming Model

GPU programming model: Figure 1 illustrates the GPU programming model. The graphics driver produces GPU commands into primary buffer and batch buffer, which is driven by the high level programming APIs like OpenGL and DirectX. GPU consumes the commands and fulfills the acceleration work accordingly. The primary buffer is a ring structure (ring buffer), which is designed to deliver the primary commands. Due to the limited space in the ring buffer, the majority (up to 98%) of commands are in the batch buffer chained to the ring buffer.

A register tuple, which includes a head register and a tail register, is implemented in the ring buffer. CPU fills commands from tail to head, and GPU fetches commands from head to tail, all within the ring buffer. The driver notifies GPU the submission and completion of the commands through the tail, while GPU updates the head. Once the CPU completes the placement of commands in the ring buffer and batch buffer, it informs GPU to fetch the commands. In general, GPU will not fetch the commands placed by the CPU in the ring buffer until the CPU updates the tail register [29].

GPU Cloud: Due to the massive computing power, GPU has been expanded from the original graphic computing to general purpose computing. The rising of GPU cloud, which extends today's elastic resource management capability from CPU to GPU, further enables efficient hosting of GPU workload in cloud and datacenter environments. The strong demand of hosting GPU applications calls for GPU clouds that offer full GPU virtualization solutions with good performance, full features and sharing capability.

2.2 GPU Benchmarks

While there are many GPU benchmarks evaluating the performance of GPU cards, they mainly focus on graphics ability of cards [1, 8] either for OpenGL or DirectX commands. Though there are a few benchmarks for general purpose computing (GPGPU) such as Rodinia [12] and Parboil [27], they are not available for Intel's GPU. Besides, existing benchmarks neglect the media processing workloads, which is a key to boost the performance of media applications in cloud.

To this end, this paper presents GMedia, a media transcoding benchmark shown in Figure 4, based on Intel's MSDK (Media Software Development Kit). Intel's MSDK grants media application developers access to hardware acceleration through a unified API. As a result, developers can take advantage of the media acceleration capabilities of future graphics-processing solutions without rewriting the code.

GMedia is a wrapper, which directly invokes the media functions of Intel's MSDK to generate common media transcoding workloads. By modifying the configuration files, we can assign source media file and target media file's settings like resolution, bitrate, FPS, etc. Besides, test cases can be run with assigned threads, which is quite helpful in order to evaluate multi-task performance. After running the benchmark, a report will be provided, which shows the average FPS (frame per second) for each thread and total average FPS. The FPS results intuitively reflect the performance.

3 gVirt and Massive Update Issue

3.1 Intel gVirt

gVirt [29], a product-level full GPU virtualization for Intel Graphics, achieves both good performance and scalability. In full GPU virtualization, a virtual machine monitor (VMM) traps and emulates the guest access to the privilege GPU resources for security and multiplexing, while passing through access to the performance critical resources, such as the access of CPU to graphic memory. For GPU commands, once the CPU submits them, they will be parsed and audited to ensure the safety. Most of the GPU commands will be executed in GPU without VMM intervention, resulting in the nearly native performance being achieved.

gVirt applies virtualization to the GPU page tables. The shared shadow global page table is implemented for all VMs in order to achieve resource partition and address space ballooning. Here, ballooning is the technique gVirt uses to isolate the address spaces of different VMs in shared shadow global page table. The shared shadow global page table is accessible for every VM. However,

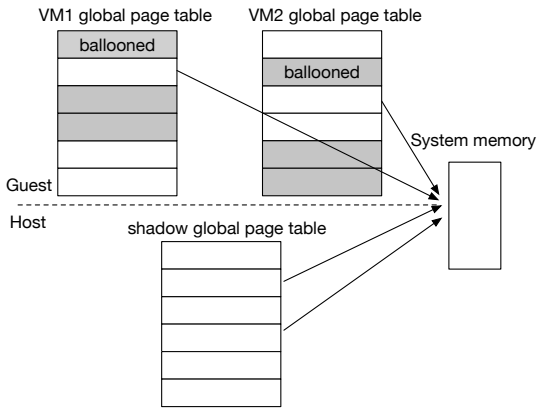

Figure 2: Shared shadow global page table

only part of the shared global page table can be accessed for one VM to guarantee the isolation, and the ballooning technique hides the rest part of shared shadow page table from this VM. As shown in Figure 2, each VM contains its own guest global page table to translate from the graphics memory frame number to the guest memory frame number. The shared shadow global page table maintains the translations from graphics memory frame number to the host memory frame number for all VMs.

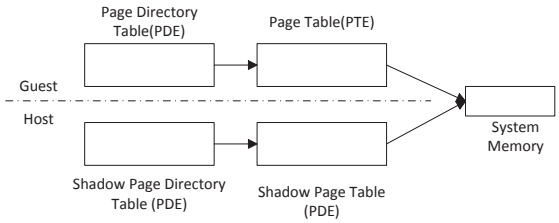

Figure 3: per-VM shadow local page table

Per-VM shadow local page table is implemented to achieve pass-through of local graphics memory access. As shown in Figure 3, the local page tables are with two-level paging structures, the first level being the Page Directory Entries (PDEs), which is located in the global page table. This, in turn, points to the second level Page Table Entries (PTEs), which is in the system memory.

The generic solution for keeping shadow page table consistent with guest page table is to write-protect the shadow page table at all points in time. When a write-protection page fault happens, VMM can potentially trap and emulate updates to the guest page table. In gVirt, shadow page tables are implemented in this *strict page table shadowing*, which is a mechanism that synchronously keeps the page table consistent with the corresponding guest page table all the time.

3.2 Massive Update Issue

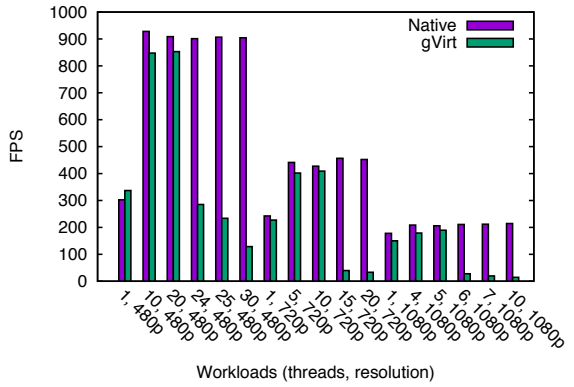

Figure 4: GMedia results of Native and gVirt

While gVirt achieves good performance in many cases, where the guest modifications of page table are infrequent, it suffers from poor performance when dealing with workloads such as media transcoding.

By observing the pattern of guest page table modifications, we find that the guest VM is frequently swapping graphics memory pages, i.e., dropping the previous pages or contents and re-construct the contents later on when needed. Once the guest VM starts to construct the memory pages, it modifies the entries of page table contiguously, until the operation is complete. In turn, this causes a huge amount of page table entry modifications, and the excessive modifications result in busy trap-and-emulate, which eventually leads to low FPS media transcoding with multiple threads. When taking this into account, it is safe to conclude that the strict shadow page table shadowing mechanism is the root cause of the performance issue.

To confirm this, we used GMedia to investigate the media transcoding performance of gVirt under various workloads. Figure 4 shows the results of media transcoding on our test platform (detailed setting in section 5) with multiple threads normalized to one thread. We run 30 cases for each resolution to get a full coverage while selectively presenting the representative cases. For many cases, the performance discrepancy between gVirt and native is not obvious. For the 480p media file transcoding, the native machine works fine in each case with small performance degradation, yet the performance on DomU (the production VM in Xen) degrades very clearly with thread multiplies over 20. For high-resolution media file transcoding, the native machine still works adequately in each case, while DomU's performance degrades with multiple threads, with over 90% in the worst cases.

Transcoding a media file requires a large amount of graphic memory in order to read the file in and process it. Once the memory is limited, Intel's GPU driver [4] [5] allocates a new memory page and modifies the page table entry to point to the new memory page. In gVirt, the write-protection page faults of the shadow page table happen massively when the thread number becomes higher or when the video resolution is high, resulting in the low FPS. Because the guest VM frequently allocates new graphic memory from system memory and massively modifies the page table entries. Therefore, we define this performance overhead problem caused by frequent page table updates as the Massive Update Issue.

3.3 PTE Update Pattern

To further analyze the Massive Update Issue, we profile 6 media transcoding cases from GMedia: 5-thread 720p, 7-thread 720p, 15-thread 720p, 3-thread 1080p, 4-thread 1080p and 10-thread 1080p, to count the VM-exits happen during the workload running. We categorize the VM-exit reasons and find that the EPT-violation dominates in cases with the Massive Update Issue. By breaking down the EPT-violation we find that the guest VM frequently modifies the PTE pages when running issued cases. Furthermore, we analyze the PTE updates to find the pattern of workloads with the Massive Update Issue, which motivates the design of gHyvi.

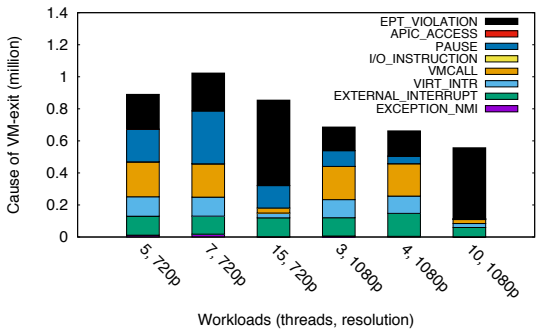

Figure 5: Break-down of VM-exit

Figure 5 shows the break-down of 6 media transcoding cases' VM-exits in the duration of 10s. Among these 6 cases, 15-thread 720p and 10-thread 1080p transcoding have much higher rates of Extended Page Tables violation (EPT-violation), which is caused by a page fault in the extended page table. As shown in Table 1, the percentages of EPT-violation are usually under 25% in other cases but dramatically increase to 62.40% in the case of 15-thread 720p and 79.45% in the case of 10-thread 1080p.

Threads	Resolution	EPT-violation Percentage
5	720p	24.43%
7	720p	23.06%
15	720p	62.40%
3	1080p	21.43%
4	1080p	23.82%
10	1080p	79.45%

Table 1: EPT-violation percentage in the 6 cases

Interestingly, when a VM guest graphics driver accesses CPU pages to prepare PTE pages for GPU, it triggers EPT-violation as well. We further provide a breakdown of the EPT-violations. PTE updates trigger 82.97% and 78.82% of VM-exit caused by EPT-violation for the cases of 15-thread 720p transcoding and 10-thread 1080p transcoding accordingly. The PTE page updates excessively expand the percentage of VM-exits caused by EPT-violation.

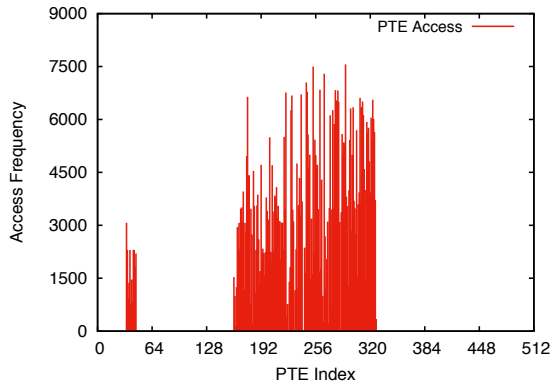

Figure 6: PTE update frequency

Furthermore, Figure 6 demonstrates the update frequency on 512 pages within 10s for 15-thread 720p transcoding case. The pages whose index lie between 150 and 320 are massively modified, and the frequency can be up to 7.5k times. Each PTE updates trigger the VM-exit, then the VMM traps and emulates the corresponding writes. However, there are some pages that are never accessed, like the pages whose index is between 320 and 512. This pattern encourages us to implement the partial reconstruction policies aside from reconstructing the whole page table, because part of the page table may stay unchanged.

We also collected the timestamp and page index to each PTE update to see the overall pattern. Figure 7 demonstrates all 627k PTE updates occurring within the 10s of 15-thread 720p transcoding case. This pattern is

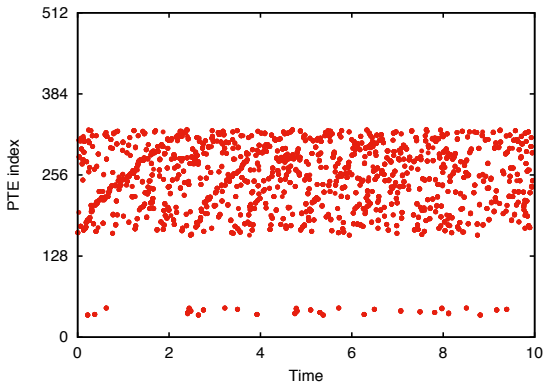

Figure 7: PTE update pattern (in 10s)

in correspondence with Figure 6. Updates on the same page repeat throughout the entire progress.

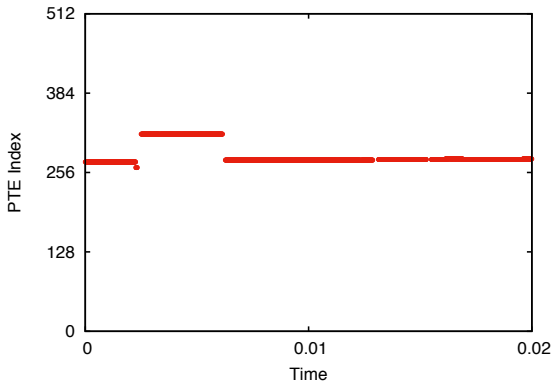

Figure 8: PTE update pattern (in 0.02s)

A small part is split from the 10s to see the detailed pattern of this case's PTE updates. Figure 8 demonstrates the PTE page update pattern in 0.2s, within the same case. The updates on one PTE page are continuous, i.e., once a PTE page is modified, there will be following updates on the same page. This pattern inspires us to remove the write-protection of PTE page once the page is modified for the first time.

4 Design and Implementation

To address the Massive Update Issue for media transcoding workload, this paper describes, gHyvi, a hybrid page table shadowing scheme for gVirt, as shown in Figure 9. gHyvi introduces a new page table shadowing mechanism for shadow page tables in gVirt, namely relaxed page table shadowing, which relaxes the constraints of write-protection to the guest page table. gHyvi switches between two different page table shadowing mechanisms, based on the pattern of GPU's current workload.

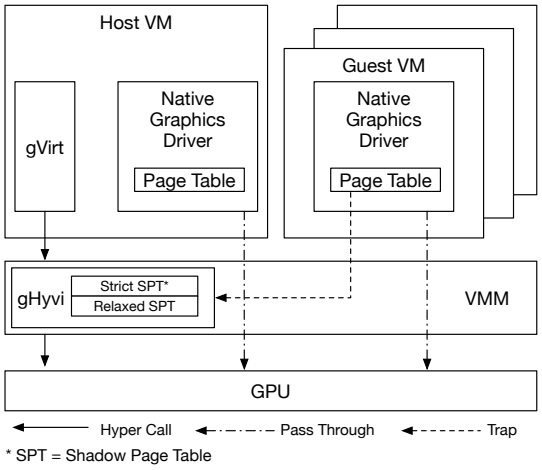

Figure 9: High level architecture of gHyvi

By combining traditional strict page table shadowing and relaxed page table shadowing mechanism, gHyvi takes advantage of both. For workloads with the Massive Update Issue like multi-thread media transcoding, gHyvi could efficiently improve the gVirt's performance.

4.1 Workflow of gHyvi

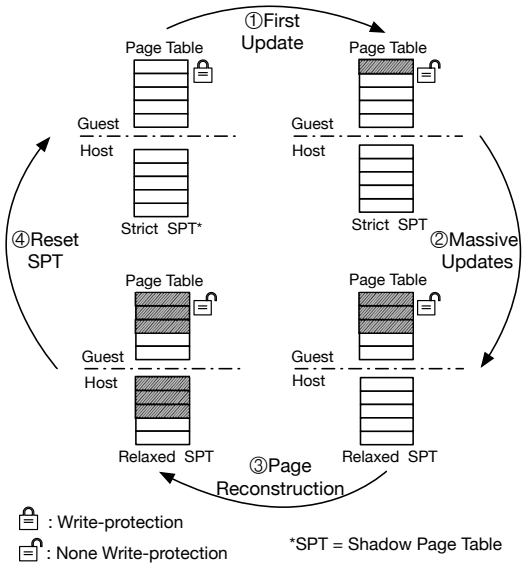

Figure 10: Workflow of gHyvi

Figure 10 illustrates the basic workflow of gHyvi:

(1) gHyvi initiates the shadow page table which is consistent with the guest page table, and it makes all the page table write-protected.

(2) If a page table entry is modified by the guest, it triggers page fault which will be trapped into gHyvi. gHyvi takes a snapshot of this page and removes the write-protection of this page. The corresponding page table entry of the shadow page table will be switched into the relaxed shadowing mechanism. Afterwards, the modifications on the guest page will not be updated to the shadow page table immediately.

(3) When the guest VM is scheduled in, the shadow page table has been already inconsistent with the guest page table. gHyvi will re-construct the shadow page table according to the previous snapshot to promote coherence with the guest page table again, so that it could guarantee the hardware engines use the correct translations.

(4) After the reconstruction of the shadow page table, gHyvi sets the page table entries in the relaxed page table shadowing back to the strict page table shadowing. Then, this workflow circle would be repeated again.

4.2 Relaxed Page Table Shadowing

From GPU's programming model, we observe that the guest VM's modifications of page table entries will not take effect until the GPU commands are submitted to physical engine by VMM. Inspired by this, we implement a new page table shadowing mechanism for page table called relaxed page table shadowing. This mechanism is applied to the guest VM's shadow page table when gHyvi detects that the guest VM modifies the page table entries massively, i.e., the trap-and-emulation of the guest page table frequently happens. In contrast to strict page table shadowing, the relaxed page table shadowing removes the write-protection of page tables to avoid the cost from trapping and emulating the modifications of page table.

For gHyvi, the relaxed page table shadowing will reduce the overhead of trapping and emulating due to continuous and massive modifications on the guest page table. After the shadow page table has been switched to the relaxed page table shadowing mechanism, modifications within the guest page table will not be updated to shadow page table temporarily. The latency is acceptable because of the GPU programming model in which GPU may fetch the commands and cache the page table translations internally at the time of command submission. At the time the commands are submitted to the physical engine, the shadow page table would be consistent with guest page table again to ensure correct translations by reconstructing the page table.

4.3 Hybrid Page Table Shadowing

As we discussed before, for many workloads there are infrequent modifications to the guest page table, where the strict page table shadowing mechanism fits well in this situation. In such cases, relaxed page table shadowing is not suitable, because reconstructing a page takes a longer period than trapping and emulating modifications on that page. To make gHyvi enjoy good performance for both cases and minimize the cost of updating shadow page table, we combine the two mechanisms into one hybrid page table shadowing, where gHyvi's shadow page tables adaptively switch between the strict shadowing and the relaxed shadowing mechanisms, based on the current workload's access pattern.

Since infrequent page table access pattern is ubiquitous, gHyvi will keep guest page table mostly working with the strict shadowing mechanism. Once the gHyvi detects the guest VM is frequently modifying the page table, it will automatically switch the guest page table into a relaxed mechanism. When the guest VM no longer frequently modifies page table, gHyvi may switch guest page table back to the strict shadowing mechanism. gHyvi can also selectively apply the relaxed shadowing mechanism to certain portions of the page table, instead of the whole page table.

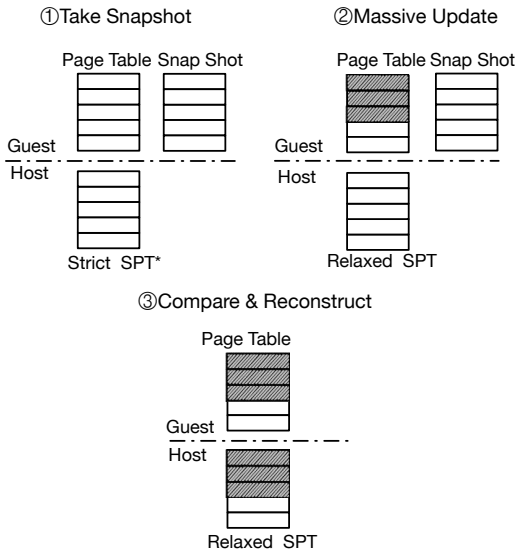

Figure 11: Page reconstruction with snapshot

4.4 Page Reconstruction

Page reconstruction is necessary when the shadow pages are not consistent with the guest pages. There are 1024 page entries in one page, and in order to reconstruct the shadow page, generally we need to re-write all the entries and make sure each entry is consistent with the corresponding entry of the guest page. However, when part of a page is modified, we do not necessarily need to rewrite all its entries when we reconstruct it, because rewriting the unmodified part of the page is costly. Hence, we introduce *snapshot* to accelerate the page reconstruction.

As shown in Figure 11, when a shadow page is consistent with the guest page after the reconstruction or initiation, we take a snapshot of the guest page and store it. When reconstructing a page, we will compare the current page with the snapshot and get the different entries. The different section is the modified part of the page. Hence, we just need to reconstruct this part to make the shadow page consistent with the guest page table. Although the cost of reconstructing a page is expensive, it is worthwhile compared to the efforts needed to trap and emulate the modification multiple times.

4.5 Reconstruction Policies

We implement four reconstruction policies for gHyvi and evaluate them to choose a final policy which delivers the best performance. When gHyvi switches a page into the relaxed shadowing mechanism, the write-protection of this page is removed. Moreover, *relaxed page table shadowing* is an asynchronous mechanism which allows the shadow page table to be inconsistent when it is not needed for delivering translations. Hence, the following modifications on it will not be updated to the shadow page immediately. Before the commands are submitted to the physical engine, gHyvi will reconstruct the page's corresponding shadow page to ensure the correct translation. The profiling of cases with Massive Update Issue in section 3.3 demonstrates that when the workload is accessing the page table massively, only certain pages are being accessed repeatedly, and the majority of the guest page table still remains untouched. Hence, it is essential for gHyvi to switch certain pages into relaxed shadowing mechanism and reconstruct them when necessary.

The full reconstruction policy is to switch all pages into the relaxed shadowing mechanism, and reconstruct them all before the commands are submitted to the physical engine. When a VM is created, it allocates 512 pages in total, and we will remove the write-protection of all 512 pages. After that, there will no longer be any trapping and emulating to update the shadow pages, and all the shadow pages will be reconstructed to guarantee that physical engine gets the correct translations.

The static partial reconstruction policy selects a certain amount of pages to apply with relaxed shadowing. It reconstructs the selected pages each time to make them consistent with their corresponding guest pages while the unselected pages still remain in the strict shadowing. Ac-

cording to the profiling of cases with the Massive Update Issue in section 3.3, there are some pages being accessed much more frequently than other pages, which are referred to as hot pages. These hot pages are specifically selected to utilize the relaxed shadowing mechanism based on the observed access pattern.

The dynamic partial reconstruction policy is utilized to apply the relaxed shadowing mechanism to pages dynamically, based on the access pattern of workload. At the time VM is created, all the pages are applied with strict shadowing and gHyvi maintains a list to record pages that are run with the relaxed shadowing. When a page is modified for the first time, a page fault occurs. gHyvi will add this page to the list and switch it into the relaxed shadowing mechanism. The new pages will then be continuously added to the list while the workload is running. Eventually the pages in the list will cover all the modified pages.

The dynamic segmented partial reconstruction policy is an optimization for the dynamic partial reconstruction policy. Like the dynamic partial reconstruction policy, gHyvi puts modified pages in the dirty list, and every time when the commands submitted to the physical engine, the shadow page table will be consistent with guest page table again, by reconstruction. However, in this optimized policy, gHyvi will reset the dirty list, and switch the pages in the list back to the strict shadowing mechanism after the reconstruction.

Currently, gHyvi uses the dynamic segmented partial reconstruction policy as default, according to the performance evaluation in section 5.2.

5 Evaluation

This section presents a set of evaluations to compare the performance of gHyvi with the original gVirt. We run media transcoding and 2D/3D workloads in Linux, along with 2D/3D workloads in Windows. We first compare the four reconstruction policies in gHyvi, which confirms that dynamic segmented partial reconstruction policy is with the best performance. Then, we use this policy to compare gHyvi with the original gVirt as well as native performance. In summary, our results show that gHyvi achieves 85% of native performance in most media transcoding test cases on Linux. For Linux 3D workloads, gHyvi has no negative effect in LightsMark, OpenArena, and UrbanTerror, respectively. For Linux 2D workloads, gHyvi shows no negative effect in firefox-asteroids, firefox-scrolling, midori-zoomed, and gnome-system-monitor, respectively. For windows 2D/3D workloads, gHyvi has no negative effect on performance in 3Dmark06 [1], Heaven3D [3], and PassMark2D [8] respectively.

5.1 Configuration

Our test platform deploys a 4th generation Intel Core processor i5 4570 with 4 CPU cores (3.2Ghz), Intel Z87 chipset, 8GB system memory and a 250GB Seagate HDD disk. The Intel Processor Graphics integrated in the CPU supports a 2GB global graphics memory space and multiple 2GB local graphics memory spaces. We run 64-bit Ubuntu 14.04 with a 3.14.1 kernel in both Dom0 and Linux guest, and 64-bit Windows 7 in Windows guest, on Xen 4.3. Both Linux and Windows run a native graphics driver. Each VM is allocated with 2 vCPUs, 2GB system memory and 672MB global graphics memory.

We evaluate the performance on native, gVirt, and gHyvi respectively. For evaluations on Linux, our customized media performance benchmark was used for media performance. The Phoronix Test Suite 3D benchmark including LighsMark, OpenArena, UrbanTerror are used for 3D performance. Additionally, Cario-perftrace 2D benchmark including firefox-asteriods (firefox-ast), firefox-scrolling (firefox-scr), midori-zoomed (midori), and gnome-system-monitor (gnome) is used for 2D performance. For evaluations on Windows, we run 3DMark06, Heaven3D and PassMark2D workloads. All the benchmarks are run under 1920*1080 resolution. We will compare the performance of VM under gHyvi, gVirt, and the native system.

5.2 Reconstruction Policy

In this section, we evaluate four reconstruction policies designed for gHyvi, full reconstruction, static partial reconstruction with four different settings (50, 100, 200, 300), dynamic partial reconstruction, and dynamic segmented reconstruction. The dynamic segmented reconstruction achieves the best performance, up to 13x of gVirt and 85% of native.

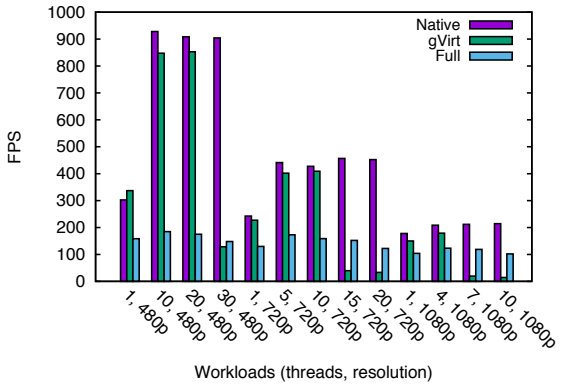

Figure 12: gHyvi with full reconstruction policy

Figure 12 presents the performance of gHyvi with the full reconstruction policy, and all multiple threads are

normalized into a single thread. Throughout all cases, the FPS of full reconstruction policy is between 100 and 200. gHyvi shows a worse performance than gVirt in cases without the Massive Update Issue, while achieving a better performance when the issue occurs. As we discussed in section 4.5, all 512 pages are applied with the relaxed mechanism, so full reconstruction brings more overhead on reconstructing non-accessed pages, which is the reason for cases with little page update showing poor performance.

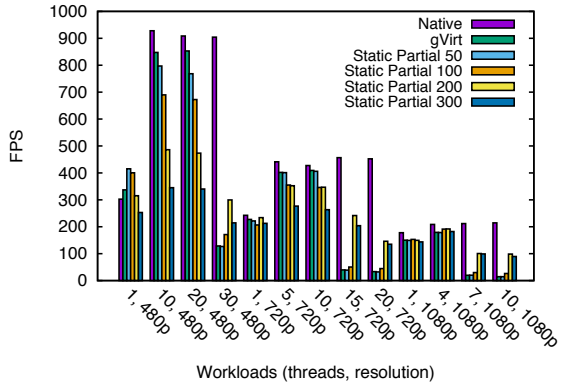

Figure 13: gHyvi with static reconstruction policy

We selectively switch 50, 100, 200, and 300 pages into the relaxed mechanism to evaluate the static partial reconstruction policy. As shown in Figure 13, for cases without the issue static partial reconstruction policy achieves a worse performance than gVirt. The more pages that are switched into the relaxed mechanism, the worse the performance static partial reconstruction becomes. For pages with few page table updates, reconstruction is meaningless. For cases with the Massive Update Issue, the static partial reconstruction policy works and achieves a superior performance than gVirt. Policy with 200 pages setting achieves the best performance for cases with the Massive Update Issue, because policies with less pages cannot cover all the frequently accessed pages, and policies with more pages include some useless pages.

Figure 14 confirms that the dynamic segmented partial reconstruction achieves better performance than dynamic partial reconstruction comprehensively. gHyvi performs better than gVirt in issued cases, and has similar performance in normal cases. The dynamic partial reconstruction switches the PTE pages into the relaxed mechanism progressively. However, some pages switched into the relaxed mechanism may never be accessed again, and reconstructing these pages will produce extra overhead. Dynamic segmented partial reconstruction resets the relaxed pages, after setting them to the guest pages. So for each cycle, dynamic segmented policy only reconstructs

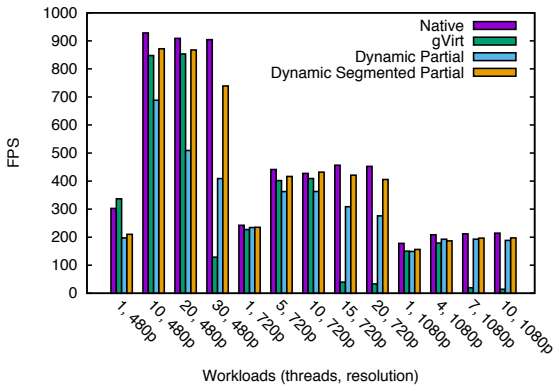

Figure 14: gHyvi with dynamic partial reconstruction and dynamic segmented partial reconstruction

pages that need to be reconstructed. Overall, dynamic segmented partial reconstruction is the most efficient policy, which is finally adopted by gHyvi.

5.3 2D and 3D performance

In this section, we evaluate the 2D and 3D performance of gHyvi under Linux and Windows. The results show that gHyvi has comparable performance with gVirt's 2D and 3D performance. Moreover, gHyvi achieves slightly superior performance than gVirt in some cases.

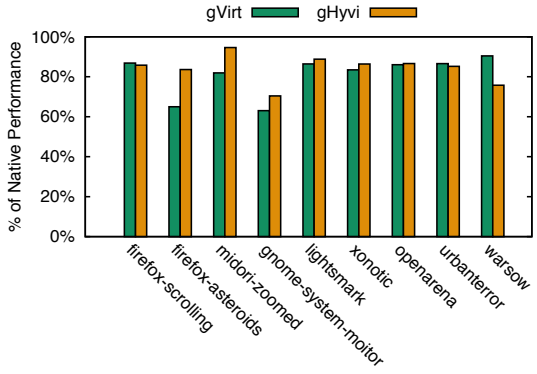

Figure 15: Performance running Linux 2D/3D workloads

Figure 15 demonstrates that gHyvi achieves up to 94.63% of native performance in 2D workloads and 88.81% in 3D workloads on Linux. Figure 16 demonstrates that gHyvi achieves up to 88.81% on Windows.

With the exception of the firefox-scrolling, urbanterror, warsow, SM2.0 and Pass2D, gHyvi outperforms gVirt. However, the performance discrepancy between gHyvi and gVirt are acceptable.

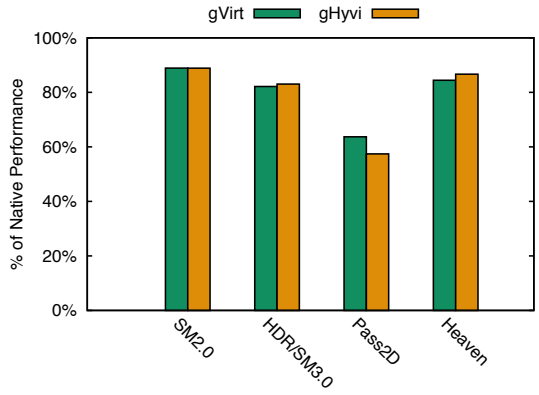

Figure 16: Performance running Windows 2D/3D workloads

6 Related Work

6.1 GPU Benchmarks

Since GPUs are used for acceleration of general purpose computing, some benchmarks have been implemented for evaluating their performance. Rodinia [12] is a benchmark suite for heterogeneous computing. It aids architects in the study of emerging platforms such as GPUs. Rodinia includes applications and kernels that target multi-core CPU and GPU platforms. And Parboil [27] is a set of throughput computing applications useful for studying the performance of throughput computing architecture and compilers. It collects benchmarks from throughput computing application researchers in many different scientific and commercial fields including image processing, bio-molecular simulation, fluid dynamics, and astronomy.

Unfortunately, the benchmarks above are not available for Intel's GPU now. Meanwhile, GPU's media performance has become a big concern for service providers. However, there is no benchmark specifically for this kind of workload. So, this paper proposes GMedia, a media transcoding benchmark based on Intel's MSDK.

6.2 GPU Virtualization

Though virtualization has been studied extensively in recent years, GPU virtualization is still a nascent area of research. Typically, there are four ways to use GPU in a Virtual Machine (VM): I/O pass-through, device emulation, API remoting, and mediated pass-through.

A naive way to use GPU in virtualized environment would be to directly pass through the device to a specific VM [20, 14]. However, the GPU resources are dedicated and cannot be multiplexed.

Device emulation, similar to binary translation in CPU virtualization, is impractical. GPUs, unlike CPUs, whose specifications are not well documented, vary between vendors [15]. Emulating GPUs from different vendors requires vast engineering work. Notably, following up the new GPU hardware would make it a nightmare to maintain the codebase.

API remoting is widely used in commercial softwares such as VMWare and VirtualBox, and has been studied throughout many years. By using API remoting, graphic commands are forwarded from guest OS to host. VMGL [23] and Oracle VirtualBox [7], both based on Chromium [21], replace the standard OpenGL library in Linux Guests with its own implementation to pass the OpenGL commands to VMM. Nonetheless, forwarding OpenGL commands is not considered a general solution, since Microsoft Windows mainly uses their own DirectX API. Whether forwarding OpenGL or DirectX commands, it would be difficult to emulate the other API. gVirtuS [17], VGRIS [25], GViM [19], rCUDA [16] and vCUDA [26] use the same manner to forward CUDA and OpenCL commands, solving the problem of virtualizing GPGPU applications.

VMware's products consist of a virtual PCI device, SVGA II card [15], and the corresponding driver for different operating systems. The emulated device acts like a real video card which has registers, graphics memory and a FIFO command queue. All accesses to the virtual PCI device inside a VM is handled on the host side, by a user-level process, where the actual work is performed. Moreover, they have designed another graphic API called SVGA3D. The SVGA3D protocol is similar to Direct3D and shares a common abstraction. The purpose of SVGA3D is to eliminate the commands for a specific GPU. Meanwhile, a GPU can also emulate the missing features by SVGA3D protocol, which provides a practical portability for their products.

Recently, two full GPU virtualization solutions have been proposed, i.e., gVirt of Intel [29] and GPUvm [28], respectively. gVirt is the first open source product level full GPU virtualization solution in Intel platforms. gVirt presents a vGPU instance to each VM which allows the native graphics driver to be run in VM. The shadow page table is updated with a coarse-grained model, which could lead to a performance pitfall under some video memory intensive workloads, such as media transcoding.

GPUvm presents a GPU virtualization solution on a NVIDIA card. Both para- and full-virtualization were implemented. However, full-virtualization exhibits a considerable overhead for MMIO handling. The performance of optimized para-virtualization is two to three times slower than native. Since NVIDIA has individual graphics memory on the PCI card, while the Intel GPU uses part of main memory as its graphics memory, the way of handling memory virtualization is different. GPUvm cannot handle page faults caused by NVIDIA

GPUs [18]. As a result, they must scan the entire page table when translation lookaside buffer (TLB) flushes. As gHyvi allocates graphics memory within the main memory, VMM can write-protect the page tables to track the page table modifications. This fine-grained page table update mechanism mitigates the overhead incurred by the Massive Update Issue.

NVIDIA GRID [6] is a proprietary virtualization solution from NVIDIA for Kepler architecture. However, there are no technical details about their products available to the public.

6.3 Memory Virtualization

One important aspect in GPU virtualization is memory virtualization, which has been thoroughly researched. The software method employs a shadow page table to reduce the overhead of translating a VM's virtual memory address. This approach could incur severe overhead under some circumstances. Agesen *et al.* [10] listed three situations where the shadow page table cannot handle well: the hidden page fault, address space switching, and the tracing page table entries. They also pointed out some optimization techniques, such as the trace mechanism and eager validating. Unfortunately, it is hard to trade off these mutually exclusive techniques. Therefore, AMD and Intel have added the hardware support for memory virtualization. All three overheads previously listed before can be eliminated, but it is not the silver bullet, a TLB miss punishment is higher in the hardware solution. In the classical VMM implementations, VMM employs a trace technique to prevent its shadow PTEs from becoming inconsistent with guest PTEs, i.e. updating shadow page table strictly after the guest page table is modified. Typically, VM trace uses write-protection mechanism, which can be the source of overhead. This technique is similar to the current gVirt's strict page table shadowing mechanism, which frequently traps and emulates the page faults of the shadow page table, and it causes overhead. gHyvi removes the write-protection from shadow page table to eliminate the overhead caused by excessive trap-and-emulation, taking advantage of the GPU programming model [9].

7 Conclusion and Future Work

gHyvi is an optimized full GPU virtualization solution, based on the Xen hypervisor, with the adaptive hybrid page table shadowing scheme, which improves performance for workloads with the Massive Update Issue when compared to gVirt. To address this issue, this paper provides a hybrid page table shadowing scheme, i.e., strict and relaxed page table shadowing, to provide an optimized full GPU virtualization based on Xen hypervisor for Intel GPUs. gHyvi combines these two page table shadowing mechanisms to reduce VM-exits to the hypervisor. Further, gHyvi automatically switches page table between them by detecting GPU's current workloads, potentially showing significantly improvement to gVirt's performance for workloads with the Massive Update Issue. In order to decide what type of the page need to be reconstructed, four reconstruction policies are introduced. By running the same testcase through the four policies, the dynamic segmented partial reconstruction policy performs the best.

For future work, we will adapt gHyvi to support KVM [22] when gVirt for KVM is ready. Additionally, gHyvi will be released in the open source community soon. We will focus on the areas of portability, scalability, and scheduling issues. With previous GPU command scheduling methods, such as VGRIS and Pegasus [13], we will investigate the low level access pattern of massive page table modification with the detailed analysis of the performance bottleneck of high level applications. We hope this optimized full GPU virtualization solution gives insight into designing the support of efficient distributed systems for GPU acceleration applications.

8 Acknowledgements

We thank our shepherd Dan Tsafrir, Haibo Chen, and the anonymous reviewers for their insightful comments. This work was supported by National Science and Technology Major Project (No. 2013ZX03002004), National R&D Infrastructure and Facility Development Program (No. 2013FY111900), NRF Singapore CREATE Program E2S2, the Shanghai Science and Technology Development Fund for High-Tech Achievement Translation under Grant No. 14511100902, and Shanghai Key Laboratory of Scalable Computing and Systems. Prof. Haibing Guan is the corresponding author.

References

[1] 3dmark06. http://www.futuremark.com.

[2] Amazone high performance computing cloud using gpu. http://aws.amazon.com/hpc/.

[3] Heaven3d. http://unigine.com/products/heaven.

[4] Intel graphics driver. http://www.x.org/wiki/IntelGraphicsDriver/.

[5] Intel processor graphics prm. https://01.org/linuxgraphics/documentation/2013-intel-core-processor-family.

[6] Nvidia grid: Graphics-accelerated virtualization. http://www.nvidia.com/object/grid-technology.html.

[7] Oracle vm virtualbox. https://www.virtualbox.org/.

[8] Passmark2d. http://www.passmark.com.

[9] ADAMS, K., AND AGESEN, O. A comparison of software and hardware techniques for x86 virtualization. *ACM Sigplan Notices 41*, 11 (2006), 2–13.

[10] AGESEN, O., GARTHWAITE, A., SHELDON, J., AND SUBRAHMANYAM, P. The evolution of an x86 virtual machine monitor. *ACM SIGOPS Operating Systems Review 44*, 4 (2010), 3–18.

[11] BARHAM, P., DRAGOVIC, B., FRASER, K., HAND, S., HARRIS, T., HO, A., NEUGEBAUER, R., PRATT, I., AND WARFIELD, A. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review 37*, 5 (2003), 164–177.

[12] CHE, S., BOYER, M., MENG, J., TARJAN, D., SHEAFFER, J. W., LEE, S.-H., AND SKADRON, K. Rodinia: A benchmark suite for heterogeneous computing. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on* (2009), IEEE, pp. 44–54.

[13] DEELMAN, E., SINGH, G., SU, M.-H., BLYTHE, J., GIL, Y., KESSELMAN, C., MEHTA, G., VAHI, K., BERRIMAN, G. B., GOOD, J., ET AL. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming 13*, 3 (2005), 219–237.

[14] DONG, Y., DAI, J., HUANG, Z., GUAN, H., TIAN, K., AND JIANG, Y. Towards high-quality i/o virtualization. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference* (2009), ACM, p. 12.

[15] DOWTY, M., AND SUGERMAN, J. Gpu virtualization on vmware's hosted i/o architecture. *ACM SIGOPS Operating Systems Review 43*, 3 (2009), 73–82.

[16] DUATO, J., PENA, A. J., SILLA, F., MAYO, R., AND QUINTANA-ORTÍ, E. S. rcuda: Reducing the number of gpu-based accelerators in high performance clusters. In *High Performance Computing and Simulation (HPCS), 2010 International Conference on* (2010), IEEE, pp. 224–231.

[17] GIUNTA, G., MONTELLA, R., AGRILLO, G., AND COVIELLO, G. A gpgpu transparent virtualization component for high performance computing clouds. In *Euro-Par 2010-Parallel Processing*. Springer, 2010, pp. 379–391.

[18] GOTTSCHLAG, M., HILLENBRAND, M., KEHNE, J., STOESS, J., AND BELLOSA, F. Logv: Low-overhead gpgpu virtualization. In *High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on* (2013), IEEE, pp. 1721–1726.

[19] GUPTA, V., GAVRILOVSKA, A., SCHWAN, K., KHARCHE, H., TOLIA, N., TALWAR, V., AND RANGANATHAN, P. Gvim: Gpu-accelerated virtual machines. In *Proceedings of the 3rd ACM Workshop on System-level Virtualization for High Performance Computing* (2009), ACM, pp. 17–24.

[20] HIREMANE, R. Intel virtualization technology for directed i/o (intel vt-d). *Technology@ Intel Magazine 4*, 10 (2007).

[21] HUMPHREYS, G., HOUSTON, M., NG, R., FRANK, R., AHERN, S., KIRCHNER, P. D., AND KLOSOWSKI, J. T. Chromium: a stream-processing framework for interactive rendering on clusters. In *ACM Transactions on Graphics (TOG)* (2002), vol. 21, ACM, pp. 693–702.

[22] KIVITY, A., KAMAY, Y., LAOR, D., LUBLIN, U., AND LIGUORI, A. kvm: the linux virtual machine monitor. In *Proceedings of the Linux Symposium* (2007), vol. 1, pp. 225–230.

[23] LAGAR-CAVILLA, H. A., TOLIA, N., SATYANARAYANAN, M., AND DE LARA, E. Vmm-independent graphics acceleration. In *Proceedings of the 3rd international conference on Virtual execution environments* (2007), ACM, pp. 33–43.

[24] LOPES, N., AND RIBEIRO, B. Gpumlib: An efficient open-source gpu machine learning library. *International Journal of Computer Information Systems and Industrial Management Applications 3* (2011), 355–362.

[25] QI, Z., YAO, J., ZHANG, C., YU, M., YANG, Z., AND GUAN, H. Vgris: Virtualized gpu resource isolation and scheduling in cloud gaming. *ACM Transactions on Architecture and Code Optimization (TACO) 11*, 2 (2014), 17.

[26] SHI, L., CHEN, H., SUN, J., AND LI, K. vcuda: Gpu-accelerated high-performance computing in virtual machines. *Computers, IEEE Transactions on 61*, 6 (2012), 804–816.

[27] STRATTON, J. A., RODRIGUES, C., SUNG, I.-J., OBEID, N., CHANG, L.-W., ANSSARI, N., LIU, G. D., AND HWU, W.-M. Parboil: A revised benchmark suite for scientific and commercial throughput computing. *Center for Reliable and High-Performance Computing* (2012).

[28] SUZUKI, Y., KATO, S., YAMADA, H., AND KONO, K. Gpuvm: why not virtualizing gpus at the hypervisor? In *Proceedings of the 2014 USENIX conference on USENIX Annual Technical Conference* (2014), USENIX Association, pp. 109–120.

[29] TIAN, K., DONG, Y., AND COWPERTHWAITE, D. A full gpu virtualization solution with mediated pass-through. In *Proc. USENIX ATC* (2014).

[30] VECCHIOLA, C., PANDEY, S., AND BUYYA, R. High-performance cloud computing: A view of scientific applications. In *Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on* (2009), IEEE, pp. 4–16.

[31] YANG, J., WANG, Y., AND CHEN, Y. Gpu accelerated molecular dynamics simulation of thermal conductivities. *Journal of Computational Physics 221*, 2 (2007), 799–804.