Trillium: The code is the IR

Amogh Akshintala
The University of North Carolina
at Chapel Hill
Email: aakshintala@cs.unc.edu

Hangchen Yu, Arthur Peters
The University of Texas at Austin
Email: hyu@cs.utexas.edu
amp@cs.utexas.edu

Christopher J. Rossbach
The University of Texas at Austin
and VMware Research Group
Email: rossbach@cs.utexas.edu

***Abstract*—GPUs are the platform of choice for many general purpose workloads such as machine learning. This is driving demand for better GPGPU support in virtualized environments like the cloud. Despite significant research attention, GPGPU virtualization remains largely an open problem due to the challenge of balancing performance against key virtualization properties: compatibility, isolation, and interposition. Consequently, two different approaches to GPGPU virtualization have been adopted by the industry: Cloud service providers, such as AWS, support GPU-capable VMs using PCIe-passthrough techniques that bypass virtualization entirely, sacrificing its benefits; Virtualization vendors, such as BitFusion and Dell XaaS, support GPGPU virtualization using user-space API-remoting, which retains some of the benefits of virtualization, but elides hypervisor interposition, thereby giving up key virtualization properties.**

We hypothesize that while API-remoting may be the only viable software virtualization technique (as it interposes the only practical interface), API-remoting should not be implemented purely in user-space. We revisit VMware's SVGA in the context of GPGPU computing and find that hypervisor-mediated API-remoting is efficient: Decoupling *device* virtualization from GPU *ISA* virtualization is key to preserving the raw speedup from GPGPU acceleration, while also preserving the benefits of hypervisor-mediation: migration, isolation, fairness, etc.

***Index Terms*—Virtualization, GPGPU, Compute Accelerator**

I. Introduction

In many parallel computing domains, compute density and programmability [8, 60, 34] have made GPUs the clear choice for efficiency and performance [4]. Popular machine learning frameworks such as Caffe [38], Tensorflow [13], Microsoft CNTK [72], and Torch7 [25] rely on GPU acceleration heavily. GPUs have made significant inroads in HPC as well: five of the top seven supercomputers in the world are powered by GPUs [12].

Despite much prior research [69, 37, 14, 66] on GPGPU virtualization, practical options currently available to providers of virtual infrastructure all involve bypassing the hypervisor. The most commonly adopted technique is to dedicate GPUs to single VM instances via PCIe pass-through [16, 64], thereby giving up the consolidation and fault tolerance benefits of virtualization. More recently, industry players such as VMware, Dell and BitFusion have introduced user-space API-remoting [21, 42, 53, 68, 30] based solutions as an alternative to pass-through. API-remoting recovers the consolidation and encapsulation benefits of virtualization but bypasses hypervisor interposition. The absence of hypervisor interposition results in multiple disjoint resource managers (the remote user-space API executor and the hypervisor) with no insight into each others' decisions, thereby leading to poor decision making, and priority-inversion problems [54].

To recover hypervisor interposition while maintaining low-overhead, we retrofit GPGPU support into a virtual GPU device: We added support for OpenCL to an implementation of the SVGA [28] (see § II-B) design in Xen, by implementing the key missing component—a compiler for SVGA's TGSI virtual ISA. This effort helped us realize that because GPUs already support vendor-specific virtual ISAs (vISAs), the additional vISA provides little benefit. In fact, we found that it harms performance by necessitating a translation layer that obscures the program's semantic information from the final vendor-provided compiler. Drawing on this lesson, we adapted TRILLIUM to take a more flexible approach to ISA virtualization: eliding it entirely when the host GPU stack bundles a compiler (most do), and using LLVM IR, when necessary, to provide a common target for GPGPU drivers.

TRILLIUM represents an unexplored point in the GPGPU virtualization design space: hypervisor-mediated API-remoting. TRILLIUM is an existence proof of a viable alternative design that preserves desirable virtualization properties such as consolidation, hypervisor interposition, isolation, encapsulation, etc., without requiring full hardware virtualization. TRILLIUM outperforms a full virtualization system from the literature by up to 14× (5.5× on average) and outperforms the para-virtual SVGA-like design by as much as 7.3× (5.4× on average).

This paper make the following contributions:

- We show that API-remoting does not have to be done entirely in user-space and that it can be hypervisor-mediated with minimal loss of performance.
- We implement GPGPU support for an SVGA-like design in the Xen hypervisor, by completing a long-missing element—the TGSI compiler—in order to leverage OpenCL support provided by the Mesa/Gallium graphics stack for Linux, via the Clover [3] project.
- We propose an improved design called TRILLIUM that removes the necessity for the vISA defined by SVGA resulting in dramatic performance improvements.
- We provide the first (to our knowledge) comprehensive empirical and qualitative comparison of a wide range of fundamental virtualization techniques from the literature.

TABLE I: COMPARISON OF EXISTING GPU VIRTUALIZATION PROPOSALS, GROUPED BY APPROACH[a]

Technique	System	lib unmod	OS unmod	lib-compat	hw-compat	sharing	isolation	migration	sched. policy	graphics	GPGPU	I/D	benchmark	slowdown[b]	native speedup	virtual speedup
Full-virtual	**GPUvm [62]**	✓		✓		✓	✓		XC, BAND		✓	*D*	Rodinia	141×	11.4×	0.08×[c]
	gVirt [64]	✓		✓		✓	✓	✓	QoS	✓		*I*	2D [6], 3D [1]	1.6×	N/A	N/A
PCIe Pass-thru	**AWS GPU [15]**	✓	✓							✓	✓	*D*	Any	1×		
API remoting	**GViM [35]**				✓	✓	✓		RR, XC		✓	*D*	CUDA 1.1 SDK	1.16×	22×	19×
	gVirtuS [31]				✓	✓	✓		RR		✓	*D*	CUDA 2.3 MM	3.1×	11.1×	3.6×
	vCUDA [58]		✓		✓		✓	✓	HW		✓	*D*	CUDA 4.0 SDK	1.91×	6×	3.1×
	vmCUDA [68]		✓		✓	✓	✓		HW		✓	*D*	CUDA 5.0 SDK	1.04×	33×	31.7×
Distributed API remoting	**rCUDA [30, 53]**		✓		✓	✓	✓		RR		✓	*D*	CUDA 3.1 SDK	1.83×	49.8×	27.2×
	GridCuda [48]		✓		✓	✓	✓		FIFO		✓	*D*	CUDA MM, SOR	1.23×		
	SnuCL [42]		✓			✓	✓				✓	*D*	SNU NPB [57]			
	VCL [18]		✓		✓	✓	✓				✓	*D*	Stencil2D [26]			
Para-virtual	**GPUvm [62]**					✓	✓		XC, BAND		✓	*D*	Rodinia	5.9×	11.4×	1.9×
	HSA-KVM [37]	✓				✓	✓		HW		✓	*I*	AMD OCL SDK	1.1×		
	LoGV [33]	✓		✓		✓	✓	✓	RR		✓	*D*	Rodinia	1.01×	11.4×	11.3×
	SVGA2 [28]	✓				✓	✓	✓		✓		*D*	2D, gaming	3.9×		
	Paradice [17]	✓		✓		✓	✓		HW, QoS	✓	✓	*D*	OpenGL, OpenCL	1.1×		
	VGVM [65]				✓	✓	✓		HW		✓	*D*	CUDA 5.0 SDK	1.02×	33×	32.3×
	TRILLIUM			✓	✓	✓	✓		HW		✓	*D*	Rodinia	2.4×	11.4×	4.8×

a. The **lib unmod** and **OS unmod** columns indicate ability to support unmodified guest libraries and OS/driver. The **lib-compat** and **hw-compat** indicate the ability (compatibility) to support a GPU device abstraction that is independent of *framework* or *hardware* actually present on the host. **sharing**, **isolation** and **sched. policy** indicate cross-domain sharing, isolation and some attempt to support fairness or performance isolation (policies such as RR Round-Robin, XC XenoCredit, HW hardware-managed, etc.). The **migration** shows support for VM migration. **I/D** indicates it supports either integrated or discrete GPU.
b. The table includes performance entries for each system including the geometric-mean slowdown (execution time relative to native execution) across all reported benchmarks. We additionally include the benchmarks used, and where possible, a report (or estimate) of the geometric-mean speedup one should *expect* for using GPUs over CPUs using hardware similar to that used in this paper. The final column is the expected geometric-mean speedup for the given benchmarks running in the virtual GPGPU system over running on native CPUs. The column is computed as the expected speedup from GPUs divided by the slowdown induced by virtualization.
c. Entries where overheads eclipse GPU-based performance gains are marked in red; performance profitable entries are blue. The greyed out cells indicate the metric is meaningless for that design. Light grey cells mean the data is unavailable.

II. BACKGROUND

Existing GPU virtualization solutions [28, 44] support graphics frameworks like Direct3D [22], OpenGL [56]. In principle, there should be no fundamental difference between GPU virtualization for graphics versus *compute* workloads. In practice, they have significantly different goals: For graphics, virtualization designs target an interactive frame rate (18-30 fps [7]). For GPGPU compute, virtualization designs must preserve the raw speedup achieved by the hand-optimized GPGPU application, which is a considerably harder target to hit. As a result, GPGPU virtualization remains an open problem. While graphics devices have long enjoyed well-defined OS abstractions and interfaces [50], research attention to OS abstractions for GPGPUs [54, 55, 59, 39, 40, 43] has yielded little consensus.

A. Traditional Virtualization Techniques

An ideal GPGPU virtualization design would require no modification of guest applications, libraries and OSes (compatibility), arbitrate fair and isolated sharing of GPU resources between mutually distrustful VMs (sharing and isolation) at the native performance of the hardware (performance), while allowing virtualized software and physical hardware to evolve independently (encapsulation). Table 1 presents designs in the literature characterized by the properties sacrificed or preserved by traditional virtualization techniques.

Pass-through techniques provide a VM with full exclusive access to a physical GPU, yielding native performance at the cost of interposition, compatibility and isolation.

Device emulation [20] provides a full-fidelity software-backed virtual device which yields excellent compatibility, interposition, and isolation. However, device emulation can't support hardware acceleration making it untenable for virtualizing GPGPUs.

Full virtualization provides a virtual environment in which unmodified GPGPU programs run on unmodified guest software stacks. Full virtualization designs from the literature [64, 62] show that overheads can be staggering due to trap-based interposition of interactions through MMIO and memory-mapped command-queues.

Para-virtualization [62, 28, 65, 45, 36, 32, 49, 51, 61, 70, 17] refers to any design in which guest artifacts are modified to work in concert with the virtualization layer. For example, VMware's SVGA [28] supports an efficiently interposable para-virtual device abstraction, but sacrifices compatibility by requiring modified guest drivers and libraries.

API remoting designs interpose application-level API calls (e.g. by shimming a dynamic library) and remote them to a

user-level GPGPU framework (e.g. CUDA, OpenCL) in the host [58, 35, 31], on a dedicated appliance VM [68], or on a remote server [30, 53, 48, 42, 18, 29, 47]. API remoting can easily provide near-native performance, at a loss of interposition for the hypervisor, and poor compatibility — guest libraries or applications must change, and evolve with any changes in the underlying GPGPU framework.

Hardware virtualization support (e.g., Single Root I/O Virtualization (SR-IOV)) enables a single physical device to present itself as multiple virtual devices. A hypervisor can manage and distribute these virtual devices to guests, effectively deferring virtualization, scheduling, and resource management to the hardware. SR-IOV exhibits close to native performance [27], but this is achieved at the cost of interposition — the hypervisor can't interpose on any interactions with the hardware. SR-IOV also suffers from the multiple administrator problem: the hardware controller and the hypervisor/OS may make mutually inconsistent decisions leading to unpredictable behavior.

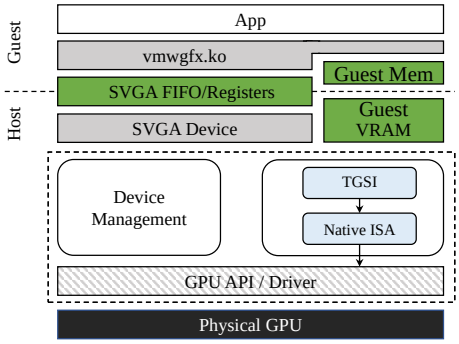

Fig. 1: The design of SVGA.

B. SVGA

SVGA [28] remotes DirectX and OpenGL over an emulated (software) PCIe device. The SVGA virtual device behaves like a physical GPU, by exporting virtual resources in the form of registers, extents of guest memory accessible to the virtual device, and a command queue. I/O registers (used for mode switching, IRQs, memory allocation) are mapped in an interposed PCIe Base Address Register (BAR) to enable synchronous emulation. Access to GPU memory is supported through asynchronous DMA. Figure 1 presents an overview of SVGA.

SVGA combines many aspects of full-, para-virtual and API remoting designs. Unmodified guests can transparently use SVGA as a VGA device, making full virtualization possible where necessary. However, access to GPU acceleration requires para-virtualization through VMware's guest driver. SVGA processes commands from a memory mapped command queue; the command queue functions as a transport layer for protocols between the guest graphics stack and the hypervisor.

SVGA uses the DirectX [22] API as its internal protocol, thereby realizing an API-remoting design. The transport layer and protocol are completely under the control of the hypervisor, enabling many of the benefits of API-remoting while ameliorating its downsides. However, using the DirectX API as a transport protocol requires that the driver and hypervisor translate guest interactions into DirectX whether they are natively expressed in DirectX or not. Coupling the transport layer with a particular version of the DirectX protocol has led to serious complexity and compatibility *challenges*: supporting each new version of the API takes many person-years (VMware introduced support for DirectX 10 (introduced in 2006) in 2015!). SVGA also supports a virtual GPU ISA called TGSI [67]. TGSI maps naturally to the graphics features of the ISAs it was originally designed to encapsulate, but has failed to keep up with GPU ISAs that have evolved to support general purpose computation primitives.

C. Mesa3D OpenCL Support

The Mesa3D Graphics Library [11] is an open-source graphics framework that implements graphics runtime libraries (e.g., OpenGL [56], Vulkan [41], Direct3D [22], and OpenCL [60]) on most GNU/Linux installations. It also includes official device drivers, written in a common framework, Gallium3D [10], for Intel and AMD GPUs. Support for NVIDIA GPUs is provided via reverse-engineered open-source Nouveau driver. Gallium3D imposes TGSI as the common virtual ISA for compute shaders, and decomposes drivers into two components: *state trackers*, which keep track of the device state, and *pipe drivers*, which provide an interface for controlling the GPU's graphics pipeline.

OpenCL support was first introduced in Mesa3D 9.0 with the release of the Clover state tracker. It was envisioned that Clover would leverage the LLVM [46] compiler to lower the OpenCL source to TGSI. Despite much effort by the open-source community [2, 5], an LLVM TGSI back-end has remained incomplete. Clover currently supports an incomplete set of OpenCL 1.1 APIs on AMD GPUs and fails to operate correctly on NVIDIA GPUs.

D. GPU ISAs and IRs

GPU front-end compilers produce code in virtual ISAs (NVIDIA PTX and LLVM IR for AMD) which are subsequently finalized using JIT compilers in the GPU driver to the native ISA (SASS and GCN). The vISA remains stable across generations to preserve compatibility, while the physical ISA is free to evolve. TGSI, the virtual ISA used in both the Mesa stack and SVGA, plays a similar role—enabling interoperability between graphics frameworks and GPUs from different vendors. An improved virtual ISA, SPIR-V, has been proposed as a new standard [41] and an effort is under way to replace TGSI with SPIR-V in the Mesa3D stack.

LLVM has become the de-facto standard for building compilers: both NVIDIA and AMD use it to implement their virtual ISA compilers, as do all the compilers in the Mesa stack including the TGSI compiler we implemented. LLVM IR is in a unique position to become a standard IR.

III. DESIGN

TRILLIUM exports an abstract virtual device and a para-virtual guest driver, which we use to interpose and forward the OpenCL and CUDA APIs to the host. Unlike SVGA, which requires translation layers to ensure that all graphics frameworks APIs can be mapped to the SVGA protocol, Trillium forwards the lowest layer in the GNU/Linux Graphics stack: the pipe-driver, effectively remoting OpenCL/CUDA API calls in the guest to the OpenCL/CUDA library in the host.

Our experience implementing the required TGSI vISA support in the Mesa graphics stack led us to believe that the TGSI layer is unnecessary. Not only does this translation introduce additional complexity in the guest stack, it also hurts performance, as we demonstrate in Section VI. The guest OpenCL compiler cannot target the native GPU architecture, and semantic information is lost to the host compiler. Further, while incorporating a TGSI compiler is possible in open frameworks like OpenCL, the task is significantly more daunting for closed frameworks like CUDA. Attempts to translate between TGSI and NVIDIA SASS in the reverse-engineered Nouveau driver understandably results in code that is significantly less performant than that produced by the proprietary stack.

TRILLIUM takes a different approach: TRILLIUM forwards API calls for compiling OpenCL code to the hypervisor. The OpenCL compiler in the host OpenCL framework (optimized for the physical hardware by the hardware vendor) is invoked on the forwarded OpenCL code to lower it directly to the physical device ISA.

Figure 2a shows the Trillium design layers in a generic hypervisor stack. The OpenCL API is forwarded from the driver similar to the SVGA model. The OpenCL compute kernel (to be run on the GPU), can be passed through to the host via hypercalls in the driver, without being translated to any vISA, where it will be translated and optimized for the physical GPU in a virtual appliance (Dom 2 in Figure 2a).

TRILLIUM does not currently guarantee performance isolation and relies on the hardware scheduler. Performance isolation can easily be implemented via a rate-limiting API scheduler in the hypervisor, such as in GPUvm [62].

IV. IMPLEMENTATION

We evaluated the TRILLIUM design against a representative of each traditional virtualization technique: full-virtualization, user-space API-remoting and SVGA. Due to the difficulty of implementing a trap-based virtualization scheme, we chose to evaluate against GPUvm [62], the only existing open-source implementation. GPUvm is tightly coupled with the Xen hypervisor [19]. As a result, all the other prototypes were built on the Xen hypervisor to keep the platform common for fair comparison.

A. Trillium

We initially implemented TRILLIUM on Xen following the SVGA design, by implementing OpenCL support in a virtual device and extending the Mesa stack with TGSI support (see Section IV-A for details). The generated TGSI is sent to the host via RPC, and then finalized to a binary that can be run on the physical NVIDIA GPU using the open source Nouveau driver. Upon empirically finding that TGSI is a performance bottleneck, we revisited the basic design. We preserve the original prototype, hereafter called XEN-SVGA, as a baseline representative of the original SVGA design: this design is shown in Figure 2b. The current TRILLIUM design is shown in Figure 2c.

XEN-SVGA and TRILLIUM, implement API-forwarding in a custom pipe-driver in Gallium3D, that we call `shadow-pipe`. We chose to forward the pipe-driver as it is presents a narrow interposition interface in the graphics driver. However, given that each OpenCL API call is decomposed into many different pipe-driver calls, other APIs higher up in the graphics stack may be better suited for interposition. The `shadow-pipe` is in the *application domain*'s graphics stack, and shims the pipe-driver interface as RPC calls to the actual Nouveau pipe-driver in the *privileged domain*.

XEN-SVGA manages user-level contexts, command queues and memory objects; and translates the input OpenCL GPGPU kernel to TGSI in the application domain. TRILLIUM skips the compilation phase in the application domain. The OpenCL kernel is forwarded to the privileged domain via RPC, where it is parsed and compiled by the LLVM NVPTX back-end in parallel. This binary is then loaded onto the GPU when the pipe-driver hits the binary loading phase. TRILLIUM can also emit LLVM IR if an OpenCL compiler is not available in the host.

Our implementation relies on gRPC as a transport mechanism between the guest and the host, as an implementation convenience. As zero-copy transfer [24, 63] and hypercall [52] mechanisms are well-studied, and a production-ready version of TRILLIUM would rely on these mechanisms, we measure and remove transport overhead from our reported measurements in Section VI. The overheads stem from remoting calls to the privileged domain over the network, which is especially significant since a single OpenCL API call may be decomposed into many pipe-driver APIs, and from the large amount of kernel input data that must be copied between VMs.

LLVM TGSI Back-end The Mesa3D stack implements OpenCL support via a state-tracker called Clover. Clover provides the library for the OpenCL application to link against, while most of the compilation is handled by invoking the OpenCL and C++ front-ends of the LLVM [46] compiler framework. Clover provides much of the front-end infrastructure required to support GPGPU computing in XEN-SVGA and TRILLIUM.

Historically, lack of a working TGSI back-end in LLVM, despite several attempts at building one in the past 5 years [2, 5],

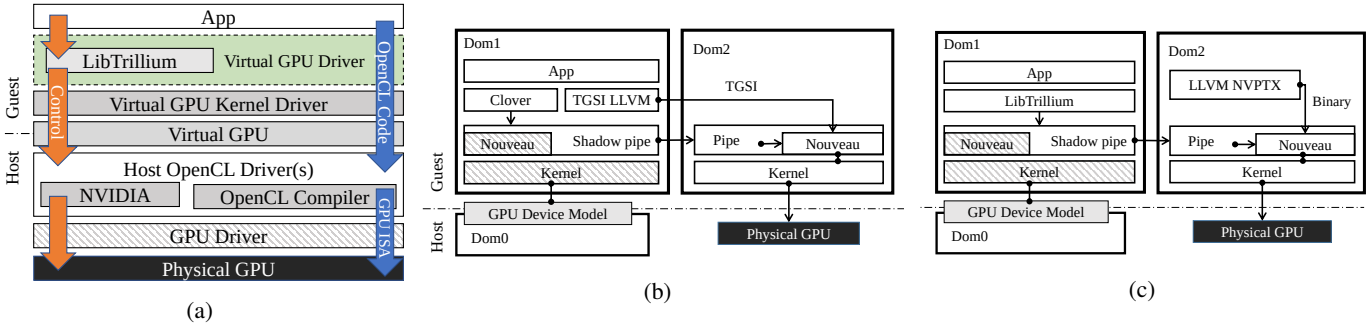

Fig. 2: XEN-SVGA and TRILLIUM designs. (a) The TRILLIUM stack. (b) XEN-SVGA approximates the SVGA model extended to support GPU Compute. (c) The design of TRILLIUM with shadow pipe.

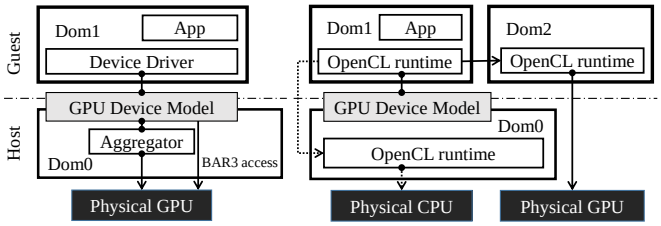

Fig. 3: Xen-based virtualizaton designs. (a) Trap-based virtualization: GPUvm. (b) User-space API remoting over RPC—dashed arrows indicate API-REMOTE-CPU, while solid ones indicate API-REMOTE-GPU.

has left OpenCL support for NVIDIA GPUs and SVGA in Mesa3D incomplete. In order to support OpenCL in XEN-SVGA, we implemented an LLVM TGSI back-end. While the TGSI back-end is not yet mature, we have added support for a majority of the 32-bit integer and floating point operations, intrinsics, memory barriers, and control flow. Using this backend we are able to compile and run 10 out of the 12 Rodinia benchmarks [23] used to benchmark GPUvm. Because the compiler is built using the LLVM framework, it enjoys all of the IR-level optimizations in LLVM.

LLVM IR handles control flow by using conditional and unconditional branches to and from Basic Blocks. A majority of the usual optimizations (constant propagation, loop unrolling, etc) are applied on the IR. On the other hand, TGSI assumes a linear control flow through the program, using higher level constructs such as IF-THEN-ELSE, FOR and WHILE loops. To accommodate this difference in control flow techniques, we leveraged a similar implementation in the AMDGPU back-end which calculates a Strongly-Connected-Components (SCC) graph from the Basic Block-based control flow in the LLVM IR, and then duplicates Basic Blocks as necessary. It is a testament to the maturity and flexibility of LLVM that the infrastructure to produce an SCC, and an example of how to use it to raise the control flow abstraction level were readily available.

B. GPUvm

GPUvm [62] is an open-source trap-based interposition design (a simplified block-diagram representation is shown in Figure 3a). The application domain (Dom 1) is presented with a GPU Device Model, which is emulated in the privileged domain (Dom 0). The emulation layer in Dom 0 interposes, validates, and fulfills all attempts to access the GPU. GPUvm has not been maintained: The last release, in 2012, is based on Xen 4.2.0 and runs on Fedora 16 [73]. In order to compare all prototypes on the same modern platform, we ported GPUvm to Ubuntu 16.04 with Xen 4.8.2.

C. User-space API remoting over RPC

In order to faithfully mimic user-level API-remoting-over-RPC systems [30, 42, 21], OpenCL API calls are trapped by a user-space shim library and forwarded via RPC from one appliance VM, which is the OpenCL "client", to another appliance VM, which acts as the OpenCL "server". Figure 3b shows the setup of the two API-remoting schemes we considered: API-REMOTE-GPU and API-REMOTE-CPU. The black arrows indicate the workflow of API-REMOTE-GPU, where the OpenCL server runs the OpenCL commands on a physical GPU using the NVIDIA OpenCL framework. The grey arrows show the API-REMOTE-CPU setup, where the OpenCL commands are executed on a multi-core CPU (Intel CPU Xeon E5-2643) using the Intel OpenCL SRB 5.0 framework. RPC is implemented using gRPC 1.6 (based on Google ProtocolBuffers 3.4.0) and inter-service communications are implemented over XML-RPC 1.39. Lower-overhead data-movement techniques, such as zero-copy, can be applied when both the client and the server are on a local machine.

D. Optimizations

TRILLIUM interposes at the pipe-driver API yielding fine-grained interposition, and therefore finer-grained multiplexing of the GPGPU. However, interposing at this layer also results in significant transport overhead. Many pipe-driver functions are responsible for context management and information retrieval—operations that do not result in interaction with the GPU. We reduce communication overhead by batching these types of API-calls, taking care to fall back to synchronous API-forwarding when any pipe-driver API calls that interact with the physical GPU are invoked.

We optimize the API-REMOTE-GPU and API-REMOTE-CPU systems by preinitializing the device and preallocating contexts and command queues on the privileged domain. These contexts are assigned to applications as they execute context creation APIs and are reclaimed asynchronously.

V. METHODOLOGY

All experiments were run on a Dell Precision 3620 workstation with NVIDIA Quadro 6000 GPU and Intel Xeon CPU E5-2643 (3.40GHz) CPU. We implemented or ported all prototypes and benchmarks on Ubuntu 16.04 with Xen 4.8.2. VMs were hardware-accelerated via Xen Hardware Virtual Machines (HVM) with 2 virtual CPUs (pinned) and 4 GB memory.

Of the GPU hardware available to us, the NVIDIA Quadro 6000 GPU was the only one that GPUvm, the full-virtualization baseline ran on. GPUvm depends on GDev [40] an open source CUDA runtime (released in 2012) implemented using Nouveau [9] GPU drivers, and the CUDA 4.2 compiler on Linux Kernel 3.6.5. GDev has not been maintained since 2014, and the effort to update it was too onerous. Experiments to control for hardware versions are reported in V-B.

A. Benchmarks

XEN-SVGA depends on the TGSI back-end compiler that we implemented to leverage the Clover OpenCL runtime in Mesa3D. API-REMOTE-GPU and API-REMOTE-CPU leverage the NVIDIA and Intel OpenCL library respectively and support all of the Rodinia benchmarks. GPUvm is built on top of the GDev CUDA runtime. Care was taken to ensure that the CUDA and OpenCL versions of the benchmarks use the same parameters, datasets, memory barriers, sync points, etc. Experiments to control for the programming framework are reported in V-B.

TABLE II: EVALUATION BENCHMARKS IN THREE CATEGRORIES[a]

Benchmark	Description	Type
backprop	Back propagation (pattern recognition)	R
gaussian	256x256 matrix Gaussian elimination	D
lud	256x256 matrix LU decomposition	M
nn	*k*-nearest neighbors classification	D
nw	Needleman-Wunsh (DNA-seq alignment)	M
pathfinder	Search shortest paths through 2-D maps	R

a. Interposition-**d**ominant, interposition-**r**are, and **m**oderate workloads.

The 10 Rodinia benchmarks that our TGSI compiler could compile were categorized based on frequency of interposition: Interposition-**D**ominant workloads run kernels hundreds or thousands of times requiring frequent interposition to set arguments, etc. Interposition-**R**are workloads run a small number of long-running kernels, requiring very little interposition. **M**oderate-interposition workloads lie somewhere in between the other two. Two benchmarks were selected from each category to be used in the evaluation (the optimizations described in IV-D take significant manual effort).

B. Control Experiments

Software and platform version dependencies necessitated that our experimental environments vary slightly for the systems under evaluation — different front-end programming languages (CUDA vs. OpenCL), different runtime implementations (GDev CUDA vs. NVIDIA CUDA), or different drivers (Nouveau vs. NVIDIA). Resolving all of these differences would have taken monumental effort, but control experiments showed that these variables had negligible impact on our measurements.

OpenCL vs. CUDA GPUvm relies on the GDev implementation of the CUDA framework, while all the other designs rely on OpenCL. To assess the impact of different front-end languages on performance, we measured execution times for all benchmarks in both CUDA and OpenCL (Rodinia includes both implementations) holding all other variables constant, and found that the front-end language has near negligible impact, and the harmonic mean of differences in kernel execution time across all benchmarks is less than 1%; the worst (maximal) case is 15%. We also found negligible difference in performance between kernels compiled using CUDA 8.0 and the CUDA 4.2 required by GDev.

Hardware Generations. The performance improvements over the span of generations between the Quadro 6000 and modern cards is substantial. To estimate the effect of this variable we ran all benchmarks on both Quadro 6000 and a more recent GPU, Quadro P6000. While overall execution times are improved substantially, and the ratio of time spent on the host to time spent on the GPU changes as a result, the relative speedups are uniform across all benchmarks. This suggests that the trends that we observe on the Quadro 6000 still hold on newer hardware. We re-iterate that software dependencies of the GPUvm baseline prevent us from using more recent hardware. Our evaluation is performed on the newest (several generations older) GPU hardware that all our systems can run on.

VI. EVALUATION

We are interested in understanding the impact of a vISA on end-to-end performance, the effect of interposition frequency on performance, and the effectiveness of our proposed design, TRILLIUM.

A. The impact of vISA choice

Deferring the compilation of front-end code to the host not only eliminates redundant translations, and the need to have a compiler in the guest driver, but also ensures that the compiler has a high-fidelity view of the physical hardware. Typically, the execution/compilation framework is extremely tightly coupled with the vISA used, making the choice of vISA even more tenuous as it leads to the second order effect of having to rely on a particular implementation of the compute framework (e.g., Mesa3D OpenCL vs NVIDIA OpenCL).

To understand the impact of the virtual ISA on the quality of the generated GPU code we measured GPU execution time

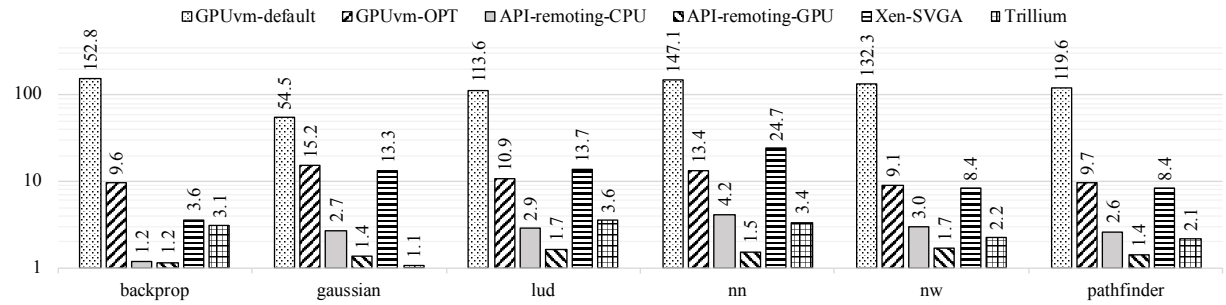

Fig. 4: End-to-end execution times of benchmarks on virtualization prototypes, relative to end-to-end execution time on the NVIDIA CUDA runtime in a native setting. The gRPC transport overhead is removed from the reported measurements, which is up to 10% of the total execution time for API remoting, and 40% for TRILLIUM.

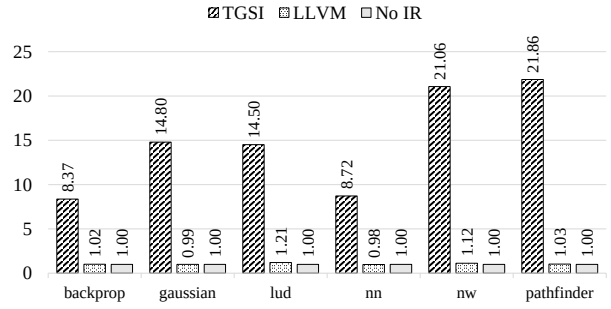

Fig. 5: Kernel execution slowdown due to virtual ISAs. TGSI: the LLVM TGSI back-end compiler used in XEN-SVGA. LLVM: LLVM NVIDIA PTX (NVPTX) back-end used in TRILLIUM. No IR: native NVIDIA compiler.

for NVIDIA SASS kernels generated in 3 ways: a) using the Mesa3d OpenCL stack (OpenCL→ TGSI→ SASS), b) using the LLVM OpenCL stack (OpenCL→ LLVM IR→ SASS), and c) using the native NVIDIA OpenCL compiler (OpenCL→ SASS). These measurements are reported in Figure 5 relative to kernel execution time in a native setting.

Code generated from TGSI IR is dramatically slower in all cases than code generated by the NVIDIA OpenCL framework. We observe slowdowns of up to 22×, with a harmonic mean of 13× across the 6 benchmarks that were optimized for evaluation. While we predicted the basic trend these experiments show, we were surprised by the magnitude of the difference. We found quality of the kernel generated by the LLVM NVPTX compiler to be comparable to native, at least in terms of execution time. This is unsurprising given recent efforts [71] to optimize the LLVM tool-chain for NVIDIA GPUs.

The TRILLIUM design uses LLVM IR as the common virtual ISA for GPGPU applications, where necessary: OpenCL code is compiled to PTX using the LLVM NVPTX back-end in the guest, and then finalized and executed in the host using the NVIDIA CUDA framework.

B. End-to-End

We compare TRILLIUM against full-virtual (GPUVM-DEFAULT and GPUVM-OPT), API remoting (API-REMOTE-GPU and API-REMOTE-CPU) and para-virtual (XEN-SVGA) systems. XEN-SVGA approximates an SVGA-like design in Xen (Mesa3D with TGSI). TRILLIUM bypasses translation from OpenCL to TGSI. To characterize the behavior a full-virtualization design, we measure GPUvm [62] in its default configuration (worst case) shown as GPUVM-DEFAULT, and in its fully optimized configuration, labeled GPUVM-OPT.

Figure 4 shows the end-to-end execution time (relative to native GPU execution) for the six chosen benchmarks for all the systems evaluated. As expected, traditional API remoting designs incur the lowest overhead, which is achieved by giving up hypervisor interposition. TRILLIUM fares well with best case performance of just 1.1× over native, and within 3.6× at worst. XEN-SVGA is sensitive to the performance lost in GPU kernel code resulting from redundant compilation through TGSI (which adds significant overheads as previously shown in Figure 5). GPUVM-OPT exhibits about 9.1× slowdown for applications with short-lived kernels (e.g. Needleman-Wunsh algorithm); the overhead can be as high as 15.2× when the workload has long-running kernels (e.g. Gaussian Elimination).

We find that remoting calls intended to a CPU is uniformly more performant than full-virtualization of the GPU, and sometimes performs just as well as (backprop) or better than remoting to the GPU (1.6× faster for the bfs benchmark. The performance gain from accelerating the bfs kernel on the GPU is severely dwarfed by the cost of initialization on the GPU). GPGPU compute is only economical when it provides acceleration over the CPU; if overheads make the CPU competitive, the profitability threshold has been crossed. Further, the competitiveness of API-REMOTE-CPU suggests opportunity: systems could back a virtual GPU with CPU if they can detect when it is profitable to do so.

VII. CONCLUSION

TRILLIUM represents a local optima in the GPGPU virtualization space—by decoupling device virtualization from GPU ISA virtualization, it maintains the virtualization benefits of a para-virtual system, while exhibiting the performance of a user-space remoting system.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers, the SCEA group, and Donald E. Porter for their helpful feedback. This work was supported by NSF grants CNS-1618563 and CNS-1718491.

REFERENCES

[1] Cairo-perf-trace. http://www.cairographics.org. Jan. 2018.

[2] Francisco Jerez's TGSI back-end. https://github.com/curro/llvm. Jan. 2018.

[3] GalliumCompute. https://dri.freedesktop.org/wiki/GalliumCompute/. Accessed: 2018-2-6.

[4] GPU Applications Catalog. https://www.nvidia.com/en-us/data-center/gpu-accelerated-applications/catalog/. Jan. 2018.

[5] Hans de Goede's TGSI back-end. https://cgit.freedesktop.org/~jwrdegoede/llvm. Jan. 2018.

[6] Phoronix test suites. http://phoronix-test-suite.com. Jan. 2018.

[7] Why frame rate and resolution matter. https://www.polygon.com/2014/6/5/5761780/frame-rate-resolution-graphics-primer-ps4-xbox-one. 2011.

[8] NVIDIA CUDA 4.0. http://developer.nvidia.com/cuda-toolkit-40, 2011. 2011.

[9] Freedesktop Nouveau open-source driver. http://nouveau.freedesktop.org, 2017. Accessed: 2017-04.

[10] Gallium3D technical overview, 2017.

[11] The Mesa 3D graphics library, 2017.

[12] TOP500 Supercomputer Sites. https://www.top500.org/lists/2018/11/, 2019.

[13] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[14] Neha Agarwal, David Nellans, Mike O'Connor, Stephen W Keckler, and Thomas F Wenisch. Unlocking bandwidth for GPUs in CC-NUMA systems. In *HPCA*, 2015.

[15] Amazon. *Amazon Elastic Compute Cloud*, 2015.

[16] Inc or Its Affiliates Amazon Web Services. Amazon EC2 P3 Instances. https://aws.amazon.com/ec2/instance-types/p3/. Accessed: 2018-2-6.

[17] Ardalan Amiri Sani, Kevin Boos, Shaopu Qin, and Lin Zhong. I/o paravirtualization at the device file boundary. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, pages 319–332, New York, NY, USA, 2014. ACM.

[18] A. Barak, T. Ben-Nun, E. Levy, and A. Shiloh. A package for OpenCL based heterogeneous computing on clusters with many GPU devices. In *Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS), 2010 IEEE International Conference on*, pages 1–7, Sept 2010.

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

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

[21] BitFusion Inc. Bitfusion FlexDirect Virtualization Technology White Paper. http://bitfusion.io/wp-content/uploads/2017/11/bitfusion-flexdirect-virtualization.pdf, 2019. Accessed: 2019-2-28.

[22] David Blythe. The Direct3D 10 system. *ACM Trans. Graph.*, 25(3):724–734, 2006.

[23] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pages 44–54. Ieee, 2009.

[24] Hsiao-keng Jerry Chu. Zero-copy tcp in solaris. In *Proceedings of the 1996 annual conference on USENIX Annual Technical Conference*, pages 21–21. Usenix Association, 1996.

[25] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.

[26] Anthony Danalis, Gabriel Marin, Collin McCurdy, Jeremy S Meredith, Philip C Roth, Kyle Spafford, Vinod Tipparaju, and Jeffrey S Vetter. The scalable heterogeneous computing (shoc) benchmark suite. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, pages 63–74. ACM, 2010.

[27] Yaozu Dong, Zhao Yu, and Greg Rose. Sr-iov networking in xen: Architecture, design and implementation. In *Proceedings of the First Conference on I/O Virtualization*, WIOV'08, pages 10–10, Berkeley, CA, USA, 2008. USENIX Association.

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

[29] José Duato, Francisco D. Igual, Rafael Mayo, Antonio J. Peña, Enrique S. Quintana-Ortí, and Federico Silla. An efficient implementation of gpu virtualization in high performance clusters. In *Proceedings of the 2009 International Conference on Parallel Processing*, Euro-Par'09, pages 385–394, Berlin, Heidelberg, 2010. Springer-Verlag.

[30] Jose Duato, Antonio J. Pena, Federico Silla, Juan C. Fernandez, Rafael Mayo, and Enrique S. Quintana-Orti. Enabling CUDA acceleration within virtual machines

using rCUDA. In *Proceedings of the 2011 18th International Conference on High Performance Computing*, HIPC '11, pages 1–10, Washington, DC, USA, 2011. IEEE Computer Society.

[31] G. Giunta, R. Montella, G. Agrillo, and G. Coviello. A gpgpu transparent virtualization component for high performance computing clouds. *Euro-Par 2010-Parallel Processing*, page 379391, 2010.

[32] Abel Gordon, Nadav Har'El, Alex Landau, Muli Ben-Yehuda, and Avishay Traeger. Towards exitless and efficient paravirtual i/o. In *Proceedings of the 5th Annual International Systems and Storage Conference*, SYSTOR '12, pages 10:1–10:6, New York, NY, USA, 2012. ACM.

[33] M. Gottschlag, M. Hillenbrand, J. Kehne, J. Stoess, and F. Bellosa. Logv: Low-overhead gpgpu virtualization. In *High Performance Computing and Communications 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCCEUC), 2013 IEEE 10th International Conference on*, pages 1721–1726, Nov 2013.

[34] Kate Gregory and Ade Miller. C++ AMP: accelerated massive parallelism with Microsoft Visual C++. 2014.

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

[36] Nadav Har'El, Abel Gordon, Alex Landau, Muli Ben-Yehuda, Avishay Traeger, and Razya Ladelsky. Efficient and scalable paravirtual i/o system. In *Proceedings of the 2013 USENIX Conference on Annual Technical Conference*, USENIX ATC'13, pages 231–242, Berkeley, CA, USA, 2013. USENIX Association.

[37] Yu-Ju Huang, Hsuan-Heng Wu, Yeh-Ching Chung, and Wei-Chung Hsu. Building a kvm-based hypervisor for a heterogeneous system architecture compliant system. In *Proceedings of the12th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '16, pages 3–15, New York, NY, USA, 2016. ACM.

[38] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[39] Shinpei Kato, Karthik Lakshmanan, Ragunathan Rajkumar, and Yutaka Ishikawa. Timegraph: Gpu scheduling for real-time multi-tasking environments. In *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'11, pages 2–2, Berkeley, CA, USA, 2011. USENIX Association.

[40] Shinpei Kato, Michael McThrow, Carlos Maltzahn, and Scott Brandt. Gdev: First-class gpu resource management in the operating system. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, USENIX ATC'12, pages 37–37, Berkeley, CA, USA, 2012. USENIX Association.

[41] Khronos Group. *Vulkan 1.0.64 - A Specification*, 2017.

[42] J. Kim, S. Seo, J. Lee, J. Nah, G. Jo, and J. Lee. Snucl: an opencl framework for heterogeneous cpu/gpu clusters. In *Proceedings of the 26th ACM international conference on Supercomputing*, page 341352. ACM, 2012.

[43] Sangman Kim, Seonggu Huh, Yige Hu, Xinya Zhang, Emmett Witchel, Amir Wated, and Mark Silberstein. GPUnet: Networking abstractions for GPU programs. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, pages 201–216, Berkeley, CA, USA, 2014. USENIX Association.

[44] H. Andres Lagar-Cavilla, Niraj Tolia, M. Satyanarayanan, and Eyal de Lara. Vmm-independent graphics acceleration. In *Proceedings of the 3rd International Conference on Virtual Execution Environments*, VEE '07, pages 33–43, New York, NY, USA, 2007. ACM.

[45] H. Andres Lagar-Cavilla, Niraj Tolia, M. Satyanarayanan, and Eyal de Lara. Vmm-independent graphics acceleration. In *Proceedings of the 3rd International Conference on Virtual Execution Environments*, VEE '07, pages 33–43, New York, NY, USA, 2007. ACM.

[46] Chris Lattner and Vikram Adve. Llvm: A compilation framework for lifelong program analysis & transformation. In *Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization*, page 75. IEEE Computer Society, 2004.

[47] Teng Li, Vikram K. Narayana, Esam El-Araby, and Tarek El-Ghazawi. Gpu resource sharing and virtualization on high performance computing systems. In *Proceedings of the 2011 International Conference on Parallel Processing*, ICPP '11, pages 733–742, Washington, DC, USA, 2011. IEEE Computer Society.

[48] Tyng-Yeu Liang and Yu-Wei Chang. Gridcuda: A grid-enabled cuda programming toolkit. In *Advanced Information Networking and Applications (WAINA), 2011 IEEE Workshops of International Conference on*, pages 141–146, March 2011.

[49] Jiuxing Liu, Wei Huang, Bulent Abali, and Dhabaleswar K. Panda. High performance vmm-bypass i/o in virtual machines. In *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference*, ATEC '06, pages 3–3, Berkeley, CA, USA, 2006. USENIX Association.

[50] Microsoft Inc. *Windows GDI*, 2017.

[51] Himanshu Raj and Karsten Schwan. High performance and scalable i/o virtualization via self-virtualized devices. In *Proceedings of the 16th International Symposium on High Performance Distributed Computing*, HPDC '07, pages 179–188, New York, NY, USA, 2007. ACM.

[52] Kaushik Kumar Ram, Jose Renato Santos, and Yoshio Turner. Redesigning xens memory sharing mechanism for safe and efficient i/o virtualization. In *Proceedings of the 2nd conference on I/O virtualization*, pages 1–1. USENIX Association, 2010.

[53] C. Reano, A. J. Pena, F. Silla, J. Duato, R. Mayo, and E. S. Quintana-Orti. Cu2rcu: Towards the complete rcuda remote gpu virtualization and sharing solution. *20th Annual International Conference on High Performance Computing*, 0:1–10, 2012.

[54] Christopher J Rossbach, Jon Currey, Mark Silberstein, Baishakhi Ray, and Emmett Witchel. PTask: operating system abstractions to manage GPUs as compute devices. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 233–248. ACM, 2011.

[55] Christopher J. Rossbach, Yuan Yu, Jon Currey, Jean-Philippe Martin, and Dennis Fetterly. Dandelion: a compiler and runtime for heterogeneous systems. SOSP'13: The 24th ACM Symposium on Operating Systems Principles, November 2013.

[56] Mark Segal and Kurt Akeley. The opengl graphics system: A specification. Technical report, Silicon Graphics Inc., December 2006.

[57] Sangmin Seo, Gangwon Jo, and Jaejin Lee. Performance characterization of the nas parallel benchmarks in opencl. In *Workload Characterization (IISWC), 2011 IEEE International Symposium on*, pages 137–148. IEEE, 2011.

[58] Lin Shi, Hao Chen, Jianhua Sun, and Kenli Li. vcuda: Gpu-accelerated high-performance computing in virtual machines. *IEEE Transactions on Computers*, 61(6):804–816, 2012.

[59] M. Silberstein, B. Ford, I. Keidar, and E. Witchel. GPUfs: Integrating a file system with GPUs. 2013.

[60] John E Stone, David Gohara, and Guochun Shi. Opencl: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering*, 12(3):66–73, 2010.

[61] Jeremy Sugerman, Ganesh Venkitachalam, and Beng-Hong Lim. Virtualizing i/o devices on vmware workstation's hosted virtual machine monitor. In *Proceedings of the General Track: 2001 USENIX Annual Technical Conference*, pages 1–14, Berkeley, CA, USA, 2001. USENIX Association.

[62] Yusuke Suzuki, Shinpei Kato, Hiroshi Yamada, and Kenji Kono. Gpuvm: Why not virtualizing gpus at the hypervisor? In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, pages 109–120, Berkeley, CA, USA, 2014. USENIX Association.

[63] Hiroshi Tezuka, Francis O'Carroll, Atsushi Hori, and Yutaka Ishikawa. Pin-down cache: A virtual memory management technique for zero-copy communication. In *Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International... and Symposium on Parallel and Distributed Processing 1998*, pages 308–314. IEEE, 1998.

[64] Kun Tian, Yaozu Dong, and David Cowperthwaite. A full gpu virtualization solution with mediated pass-through. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, pages 121–132, Berkeley, CA, USA, 2014. USENIX Association.

[65] Dimitrios Vasilas, Stefanos Gerangelos, and Nectarios Koziris. VGVM: efficient GPU capabilities in virtual machines. In *International Conference on High Performance Computing & Simulation, HPCS 2016, Innsbruck, Austria, July 18-22, 2016*, pages 637–644, 2016.

[66] Jan Vesely, Arkaprava Basu, Mark Oskin, Gabriel H. Loh, and Abhishek Bhattacharjee. Observations and Opportunities in Architecting Shared Virtual Memory for Heterogeneous Systems. In *ISPASS*, 2016.

[67] VMware, X.org, Nouveau. *Tungsten Graphics Shader Infrastructure*, 2012.

[68] Lan Vu, Hari Sivaraman, and Rishi Bidarkar. Gpu virtualization for high performance general purpose computing on the esx hypervisor. In *Proceedings of the High Performance Computing Symposium*, HPC '14, pages 2:1–2:8, San Diego, CA, USA, 2014. Society for Computer Simulation International.

[69] Carl Waldspurger, Emery Berger, Abhishek Bhattacharjee, Kevin Pedretti, Simon Peter, and Chris Rossbach. Sweet spots and limits for virtualization. In *Proceedings of the12th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '16, pages 177–177, New York, NY, USA, 2016. ACM.

[70] Paul Willmann, Scott Rixner, and Alan L. Cox. Protection strategies for direct access to virtualized i/o devices. In *USENIX 2008 Annual Technical Conference*, ATC'08, pages 15–28, Berkeley, CA, USA, 2008. USENIX Association.

[71] Jingyue Wu, Artem Belevich, Eli Bendersky, Mark Heffernan, Chris Leary, Jacques Pienaar, Bjarke Roune, Rob Springer, Xuetian Weng, and Robert Hundt. Gpucc: An open-source gpgpu compiler. In *Proceedings of the 2016 International Symposium on Code Generation and Optimization*, CGO '16, pages 105–116, New York, NY, USA, 2016. ACM.

[72] Dong Yu, Adam Eversole, Mike Seltzer, Kaisheng Yao, Zhiheng Huang, Brian Guenter, Oleksii Kuchaiev, Yu Zhang, Frank Seide, Huaming Wang, et al. An introduction to computational networks and the computational network toolkit. *Microsoft Technical Report MSR-TR-2014–112*, 2014.

[73] Hangchen Yu and Christopher J Rossbach. Full virtualization for gpus reconsidered. In *14th Workshop on Duplicating, Deconstructing, and Debunking (WDDD), ISCA*, 2017.