Software compute virtualization - realizability, performance, and programmability

AMOGH AKSHINTALA

Abstract

The proposed dissertation will focus on developer effort and compatibility in software virtualization of CPU ISAs, and software virtualization of specialized compute devices (e.g., GPUs, TPUs) that are programmed through an API.

Although binary translation is a well-established software ISA virtualization technique, given the size and complexity of today's dominant ISAs, developers are routinely forced to adopt ad-hoc techniques to prioritize development effort. The proposed dissertation will present a principled approach to determine priority among different parts of the ISA. We believe this data will be useful to designers of virtual ISAs and **AA: what was the other thing?** as well.

Specialized compute accelerators, such as GPUs and TPUs, are usually controlled through a user-space API. The proposed dissertation will show that unlike with CPUs, where the ISA is the canonical interface provided to the programmer, ISA virtualization is untenable for specialized compute accelerators. Further, the proposed dissertation will present a present a novel taxonomy, *IEMTS* for cleanly understanding the design space for virtualizing compute accelerators. Based on insights from this taxonomy, the proposed dissertation will present a novel virtualization technique, *hypervisor-mediated API-remoting*, that is at once realizable and performant.

1 Introduction

Virtualization, defined broadly, is a means to an end: fair, isolated, and efficient sharing of resources among mutually distrustful entities. Virtualization is vital to high utilization of available physical resources in large computing installations. Illustratively, virtualization is foundational to the cloud computing paradigm.

Virtualization has a long and glorious history. Virtual memory was first described by German physicist Fritz-Rudolf Güntsch in his doctoral dissertation in 1956 [9] and commercialized [10] in the Cambridge University/Ferranti Inc. Atlas computer. The idea of virtualizing the entire computer was invented not longer thereafter, in 1962, by IBM [1] and commercialized as the IBM VM-370 [4] hypervisor for the IBM 370 computer. Virtualization was briefly forgotten through the 1980s and 1990s, as the mainframe computer was all but wiped out during the PC revolution. Intel's x86 Instruction Set Architecture (ISA), which came to dominate the Personal Computers that transplanted the mainframe, was not designed to be traditionally virtualizable [12].

In the mid 90s, Sun invented a new virtualization scheme with the Java language: *application virtualization*. Java

Virtualizing a compute resource, such as a CPU, typically involves mediating access to the physical resource either by exposing an interface that is identical to that of the physical resource (*full-virtualization*), or by exposing an alternative interface, operations on which are in-turn synthesized to the native interface (*para-virtualization*). The exposed interface is *virtual*, in that it is not directly exposed by the physical underlying hardware, and instead is entirely under the control of supervisory virtualization software, the *hypervisor* (also known as the *Virtual Machine Monitor*). A *virtual machine* is, thus, a machine that is entirely under the control of the hypervisor, as opposed to being a physical machine. While operations in the virtual machine may be directly executed on the physical hardware for performance reasons, as in the case of hardware-assisted virtualization schemes like AMD-V and Intel VT-x, all privileged operations still trap to the hypervisor. The interface interposed may be a hardware interface (ISA, Memory, I/O Protocols, etc.) or a software interface (Syscalls, APIs, etc.). In either case, the hypervisor is responsible for ensuring fair, isolated, efficient sharing of the virtualized resource.

1.1 CPU virtualization

Four decades of attention from both the academic community and industry has given rise to a large body of techniques that enable efficient virtualization of CPUs: software techniques such as binary translation and device emulation, are well established. While dominant ISAs, such as x86 and ARM, even provide extensions to enable low-overhead virtualization, binary translation results in lower overhead for sequences of sensitive instructions that need to be emulated [**?**].

However, when implementing a new binary translator [8] or developing a secure virtual instruction set, **AA: what was 3rd?**, the developer is left to their own devices to ascertain development priority or to understand the need to maintain compatibility. Understandably, developers typically adopt ad-hoc methodologies to get over this challenge. This dissertation will present a principled approach to discriminate between different parts of a given ISA, based on the frequency of occurrence of instructions weighted by the popularity of the applications they occur in. We will present a methodology (and the resulting data) that leverages user preference to systematically answer such questions in Chapter 1. This is completed work [2].

1.2 Accelerator virtualization

Compute heavy and data parallel workloads such as graph processing and machine learning have precipitated a Cambrian explosion of specialized processors. These emerging compute devices (e.g., GPGPUs, TPUs, IPUs, IO accelerators), however, pose a challenge to virtualization developers, who once again find themselves balancing the essential characteristics of a virtualization scheme—compatibility, interposition, sharing, isolation—with the need to preserve the raw performance these processors provide. Virtualization techniques developed for I/O devices, such as

NICs, are untenable for specialized compute devices because they all sacrifice one or more of the essential characteristics listed above. Full-virtualization based schemes, such as GPUvm [14], suffer from massive overheads that essentially negate the speedup that makes the specialized compute unit attractive in the first place. Para-virtual systems, such as SVGA [5] that interpose on low-level interfaces, such as the kernel driver, introduce much lower overhead than full-virtualization based schemes but have poor compatibility, i.e., the introduction of an artificial abstract interface constructed expressly for the purpose of interposition necessitates massive engineering effort to support new hardware in the host and new software frameworks in the guest. User-space API-remoting solutions [15, 7, 13] interpose on the user-space API in the guest and forward the interposed operation to the host as an RPC. This approach introduces very low overhead and can evolve with the hardware easily, but has traditionally eschewed hypervisor interposition, thereby making it difficult to enforce safety and isolation among guests.

Virtualizing a Graphics Processing Unit (GPU) for the purposes of graphics rendering is a well studied problem, with existing commercial solutions, e.g., VMware's SVGA [6]. Over the last decade, GPUs have been re-purposed for parallel general purpose compute (commonly known as GPGPU). Chapter 2 will present our findings from attempting to extend the SVGA model of GPU virtualization to cover GPGPU virtualization as well. We find that the tight coupling between ISA virtualization and device virtualization in SVGA leads to poor performance for GPGPU compute. We propose a new virtualization scheme, Trillium, that doesn't rely on ISA virtualization and show that Trillium outperforms all other traditional virtualization schemes while retaining hypervisor interposition. Material presented in Chapter 2 will be drawn from a published paper [3].

Specialized compute units (e.g., Google TPU, Intel QAT, etc.) are typically exposed to developers via a user-space API. The API is typically implemented by a combination of proprietary software that interacts with the hardware through opaque interfaces. Chapters 3, 4 and 5 will explore the performance implications of virtualizing the user-space API for specialized compute accelerators. Chapter 3 will present an overview of AvA, a framework that enables automated virtualization of accelerator APIs. Chapter 4 will focus on the performance implications of API-remoting based virtualization of a single specialized accelerator. Chapter 5 will explore performance issues that arise when an application uses multiple API-remoted virtual accelerators in a pipelined fashion. Chapters 3 and 4 will draw on material that appeared in a HotOS workshop paper [16] and a full paper that is currently under submission. Chapter 5 is proposed work.

Virtualization schemes are traditionally taxonomized according to the core techniques employed (e.g. emulation, full- or para-virtualization, API remoting, etc.), and evaluated in a property trade-off space comprising performance, compatibility, interposition, and isolation. We argue that both the de facto taxonomy and the property trade-off space are illustrative but not informative for GPGPU virtualization: there is a large body of research that has had little influence on practice. We suggest an alternative framework called `IEMTS` that teases apart design axes that are implicitly and unnecessarily intertwined in much of the literature. By focusing on the **I**nterface interposed, the interposition **E**ndpoints, the **M**echanism of interposition, the **T**ransport used to move the interposed operations between the guest and the host, and the mechanism used to **S**ynthesize the interposed interface, `IEMTS` enables a clearer understanding of trade-offs in prior designs and provides a model for comparison of alternative designs. `IEMTS` will be presented in Chapter 6, along with analysis of traditional virtualization techniques in the context of GPGPUs.

Concretely, the proposed dissertation will evaluate the following hypotheses:

H 1: Priority among instructions in an ISA, in the context of binary translation, can be automatically inferred from user preferences. (Chapter 1)

H 2: ISA virtualization is untenable for performant virtualization of compute accelerators. (Chapter 2)

H 3: Hypervisor-mediated API-remoting is a low-overhead virtualization scheme for API-controlled compute accelerators. (Chapters 3, 4, and 5)

H 4: The characteristics of a virtualization technique can be succinctly described by a scheme that explicitly captures the *Interface* interposed, the *Endpoints* interposed on, the *Mechanism* of interposition, the *Transport* used to connect the interposed endpoints, and the mechanism used to *Synthesize* the interposed operation on the host. (Chapter 6)

2 CPU virtualization

CPU virtualization as a technique was first considered for the IBM 360 in 1970 [11]. The idea then, as it is today is, was to provide each user with the illusion of having a dedicated machine to themselves, by simultaneously running multiple operating systems on the same machine. The IBM 370 was specifically designed to be *virtualizable* [12].

The PC revolution, in the 1980s, saw new ISAs establish their dominance in computing. These new ISAs were not explicitly designed to be virtualizable; on the contrary, Intel x86 is considered classically unvirtualizable. Intel's CTO at the time, Pat Gelsinger is

Task	Deadline
Dissertation proposal	Nov. 2019
Implement vTask in AvA	15 Jan. 2020
Dissertation draft to committee	1 Mar. 2020
Dissertation defense	late Mar. 2020
Submit dissertation to graduate school	12 Apr. 2020

Table 1: *Proposed timeline.*

3 GPU ISA virtualization

4 Hypervisor-mediated API-remoting

5 IEMTS

6 vTask

7 Plan of Work

References

[1] R.J. Adair. *A Virtual Machine System for the 360/40*. IBM Cambridge Scientific Center report. International Business Machines Corporation, Cambridge Scientific Center, 1966.

[2] Amogh Akshintala, Bhushan Jain, Chia-Che Tsai, Michael Ferdman, and Donald E Porter. x86-64 instruction usage among c/c++ applications. In *Proceedings of the 12th ACM International Conference on Systems and Storage*, pages 68–79. ACM, 2019.

[3] Amogh Akshintala, Hangchen Yu, Arthur Peters, and Christopher J Rossbach. Trillium: The code is the ir. In *The Second Special Session on Virtualization in High Performance Computing and Simulation (VIRT 2019), Dublin, Ireland*, 2019.

[4] R. J. Creasy. The origin of the vm/370 time-sharing system. *IBM J. Res. Dev.*, 25(5):483–490, September 1981.

[5] Micah Dowty and Jeremy Sugerman. Gpu virtualization on vmware's hosted i/o architecture. *SIGOPS Oper. Syst. Rev.*, 43(3):73–82, July 2009.

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

[7] 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.

[8] Alex Fishman, Mike Rapoport, Evgeny Budilovsky, and Izik Eidus. HVX: Virtualizing the cloud. In *Presented as part of the 5th USENIX Workshop on Hot Topics in Cloud Computing*, San Jose, CA, 2013. USENIX.

[9] Fritz-Rudolf Güntsch. *Logical Design of a Digital Computer with Multiple Asynchronous Rotating Drums and Automatic High Speed Memory Operation*. Doctoral dissertation, Technische Universität Berlin, 1956.

[10] Tom Kilburn, David BG Edwards, Michael J Lanigan, and Frank H Sumner. One-level storage system. *IRE Transactions on Electronic Computers*, (2):223–235, 1962.

[11] R. A. Meyer and L. H. Seawright. A virtual machine time-sharing system. *IBM Systems Journal*, 9(3):199–218, Sep 1970.

[12] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, July 1974.

[13] 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.

[14] Yusuke Suzuki, Shinpei Kato, Hiroshi Yamada, and Kenji Kono. Gpuvm: Why not virtualizing gpus at the hypervisor? In *USENIX Annual Technical Conference*, pages 109–120, 2014.

[15] 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.

[16] Hangchen Yu, Arthur M Peters, Amogh Akshintala, and Christopher J Rossbach. Automatic virtualization of accelerators. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, pages 58–65. ACM, 2019.