Enabling High Performance Computing in Cloud Infrastructure using Virtualized GPUs

Andrew J. Younge[*] [†]
ajyounge@indiana.edu

John Paul Walters[†]
jwalters@isi.edu

Steve Crago[†]
crago@isi.edu

Geoffrey C. Fox[*]
gcf@indiana.edu

[*]Indiana University
2719 E 10th Street
Bloomington, IN 47408

[†]Information Sciences Institute
University of Southern California
3811 Fairfax Drive
Arlington, VA 22203

ABSTRACT

With the advent of virtualization and Infrastructure-as-a-Service (IaaS), the broader scientific computing community is considering the use of clouds for their technical computing needs. This is due to the relative scalability, ease of use, advanced user environment customization abilities clouds provide, as well as many novel computing paradigms available for data-intensive applications. However, there is still a notable gap that exists between the performance of IaaS when compared to typical high performance computing (HPC) resources, limiting the applicability of IaaS for many potential scientific users.

Most recently, general-purpose graphics processing units(GPGPUs or GPUs) have become commonplace within high performance supercomputers. We propose to bridge the gap between supercomputing and Clouds by providing GPU-enabled virtual machines. Specifically, the Xen hypervisor is utilized to leverage specialized hardware-assisted I/O virtualization tools in order to provide advanced HPC-centric Nvidia GPUs directly in guest VMs. We evaluate this work by measuring the performance of two Nvidia Tesla GPUs and comparing to bare-metal hardware. Results show this method of leveraging GPUs within virtual machines is a viable use case for many scientific computing workflows, and could help support high performance cloud infrastructure in the near future.

Keywords

High Performance Computing, Cloud Computing, GPU, Virtualization

1. INTRODUCTION

Cloud computing [2] has recently established itself as a prominent paradigm within the realm of Distributed Systems [31] in a very short period of time. Clouds are an internet-based solution that provide computational and data models for utilizing resources, which can be accessed directly by users on demand in a uniquely scalable way. Cloud computing functions by providing a layer of abstraction on top of base hardware to enable a new set of features that are otherwise intangible or intractable. These benefits and features include Scalability, a guaranteed Quality of Service (QoS), cost effectiveness, and direct user customization via a simplified user interface [34].

While the origin of cloud computing is based in industry through solutions such as Amazon's EC2 [1], Google's MapReduce [9], and Microsoft's Azure [3], the paradigm has since become integrated in all areas of science and technology. Most notably, there is an increasing effort within the High Performance Computing (HPC) community to leverage the utility of clouds for advanced scientific computing to solve a number of challenges still standing in the field. This can be clearly seen in large-scale efforts such as the FutureGrid project [32], the Magellan project [35], and through various other Infrastructure-as-aService projects including OpenStack [26], Nimbus [19], and Eucalyptus [24].

Within HPC, there has also been a notable movement toward dedicated accelerator cards such as general purpose graphical processing units (GPGPUs, or GPUs) to enhance scientific computation problems by an upwards of two orders of magnitude. This is accomplished through dedicated programming environments, compilers, and libraries such as CUDA [25] from Nvidia as well as the OpenCL effort [30]. When combining GPUs in an otherwise typical HPC environment or supercomputer, major gains in performance and computational ability have been reported in numerous fields [5, 22], ranging from Astrophysics to Bioinformatics. Furthermore, these gains in computational power have also reportedly come at an increased performance-per-watt [16], a metric that is increasingly important to the HPC community as we move closer to exascale computing [21] where power consumption is quickly becoming the primary constraint.

With the advent of both clouds and GPUs within the field of scientific computing, there is an immediate and ever-growing need to provide heterogeneous resources, most immediately GPUs, within a cloud environment in the same scalable, on-demand, and user-centric way that many cloud users are already accustomed to [6]. While this task alone is nontrivial, it is further complicated by the high demand for performance within HPC. As such, it is performance that is paramount to the success of deploying GPUs within cloud environments, and thus is the central focus of this work.

This manuscript is organized as follows. First, in Section 2, we discuss the related research and options are currently available for providing GPUs within a virtualized cloud environment. In Section 3, we discuss the methodology behind our proposed solution and the details of implementation. In Section 4 we outline the evaluation of our solution using two different Nvidia Tesla GPUs and compare to the best-case native application in Section 5. Then, we discuss the implications of these results in Section 6 and consider the applicability of each method within a production cloud system. Finally, we conclude with our findings and suggest directions for future research.

2. VIRTUAL GPU DIRECTIONS

Recently, GPU programming has been a primary focus for numerous scientific computing applications. Significant progress has been accomplished in many different workloads, both in science and engineering, based on parallel abilities of GPUs for floating point operations and very high on-GPU memory bandwidth. This hardware, coupled with CUDA and OpenCL programming frameworks, has lead to an explosion of new GPU-specific applications that outperform even the fastest multicore counterparts by an order of magnitude [27]. In addition, further research will leverage the per-node performance of GPU accelerators with the high speed, low latency interconnects commonly utilized in supercomputers and clusters in order to create a hybrid GPU + MPI class of applications that seek scalability to many GPUs simultaneously [20].

Since the establishment of cloud computing in industry, research groups have been evaluating its applicability to science [13]. Historically, HPC and Grids have been on similar but distinct paths within distributed systems, and have concentrated on performance, scalability, and solving complex, tightly coupled problems within science. This has lead to the development of supercomputers with many thousands of cores, high speed, low latency interconnects, and sometimes also coprocessors and FPGAs [4, 7]. Only recently have these systems been evaluated from a cloud perspective [35]. An overreaching goal exists to provide HPC Infrastructure as its own service (HPCaaS) [28], aiming to comprehend and limit the overhead of virtualization, and reducing the bottlenecks classically found in CPU, memory, and I/O operations within hypervisors [18, 36]. Furthermore, the transition from HPC to cloud computing becomes more complicated when we consider adding GPUs to the equation.

GPU availability within a cloud is a new concept that has sparked a large amount of interest within the community. The first successfully deployment of GPUs within a Cloud environment was the Amazon EC2 GPU offering. A collaboration between Nvidia and Citrix also exists to provide cloud-based gaming solutions to users using the new Kepler GPU architecture [33], although this is currently not targeted towards HPC applications.

The task of virtualizing a GPU accelerator for use in a virtualized cloud environment is one that presents a myriad of challenges. This is due to the complicated nature of virtualizing drivers, libraries, and the heterogeneous offerings of GPUs from multiple vendors. Currently, two possible techniques exist to fill the gap in providing GPUs in a cloud infrastructure: back-end I/O virtualization, which this manuscript focuses on, and Front-end remote API invocation.

2.1 Front-end Remote API invocation

One method for using GPUs within a virtualized Cloud environment is through front-end library abstractions, the most common of which is remote API invocation. Also known as API remoting or API interception, it represents a technique where API calls are intercepted and forwarded to a remote host where the actual computation occurs. The results are then returned to the front-end process that spawned the invocation, potentially within a virtual machine. The goal of this method is to provide an emulated device library where the actual computation is offloaded to another resource on a local network.

Front-end remote APIs for GPUs have been implemented by a number of different technologies for different uses. To solve the problem of graphics processing in VMs, VMWare [23] has developed a device-emulation approach that emulates the Direct3D and OpenGL calls to leverage the host OS graphics processing capabilities to provide a 3D environment within a VM. API interception through the use of wrapper binaries has also been implemented by technologies such as Chromium [17], and Blink. However these graphics processing front-end solutions are not suitable for general purpose scientific computing, as they do not expose interfaces that CUDA or OpenCL can use.

Currently, efforts are being made to provide a front-end remote API invocation solutions for the CUDA programming architecture. vCUDA [29] was the first of such technologies to enable transparent access of GPUs within VMs by API call interception and redirection of the CUDA API. vCUDA substitutes the CUDA runtime library and supports a transmission mod using XMLRPC, as well as a share mode built on VMRPC, a dedicated remote procedure call architecture for VMM platforms. The share model generally resulting in better performance, especially as the volume of data increases, although there may be limitations in VMM interoperability.

Like vCUDA, gVirtuS uses API interception to enable transparent CUDA, OpenCL, and OpenGL support for Xen, KVM, and VMWare virtual machines [14]. gVirtuS uses a front-end/back-end model to provide a VMM-independent abstraction layer to GPUs. Data transport from gVirtuS' front-end to the back-end is accomplished through a combination of shared memory, sockets, or other hypervisor-specific APIs. gVirtuS' primary disadvantage is in its decreased performance in host-to-device and device-to-host data movement due to overhead of data copies to and from its shared memory buffers.

rCUDA [11, 12], a recent popular remote CUDA framework, also provides remote API invocation to enable VMs to access remote GPU hardware by using a sockets based implementation for high-speed near-native performance of CUDA based applications. rCUDA recently added support for using InfiniBand's high speed, low latency network to increase performance for CUDA applications with large data volume requirements. rCUDA version 4.1 also supports the CUDA runtime API version 5.0, which supports peer device memory access and unified addressing. One drawback of this method is that rCUDA cannot implement the undocumented and hidden functions within the runtime framework, and therefore does not support CUDA C extensions. While rCUDA provides some support tools, native execution of CUDA programs is not possible and programs need to be recompiled or rewritten to use rCUDA. Furthermore, like gVirtuS and many other solutions, performance between host-to-device data movement is only as fast as the underlying interconnect, and in the best case with native RDMA InfiniBand, is roughly half as fast as native PCI Express usage.

3. DESIGN AND IMPLEMENTATION

Another approach to using a GPU in a virtualized environment is to provide a VM with direct access to the GPU itself, instead of relying on a remote API. Our solution in this manuscript focuses on this approach. Devices on a host's PCI-express bus are virtualized using directed I/O virtualization technologies recently implemented by chip manufacturers, and then direct access is relinquished upon request to a guest VM. This can be accomplished using the VT-d and IOMMU instruction sets form Intel and AMD, respectively. This mechanism, typically called PCI passthrough, implements a memory management unit (MMU) to handle direct memory access (DMA) coordination and interrupt remapping directly to the guest VM, thus bypassing the host entirely. With host involvement being nearly non-existent, near-native performance of the PCI device within the guest VM can be achieved, which is an important characteristic for using a GPU within a cloud infrastructure.

PCI Passthrough itself is a standard technique for many other I/O systems such as storage or network controllers. However, GPUs (even from the same vendor) have additional legacy VGA compatibility issues and non-standard low-level interface DMA interactions that make direct PCI Passthrough difficult. As such, most common hypervisors such as KVM or VirtualBox do not provide support for GPU PCI Passthrough. VMWare has started use of a vDGA system for hardware GPU utilization, however it remains in tech preview and only documentation for Windows VMs is present [23]. In our experimentation, we have found that the Xen hypervisor is an ideal virtualization solution for providing PCI Passthrough of GPU devices due to its open nature, extensive support, and high degree of reconfigurability.

Today's GPUs can provide a variety of frameworks for application programmers to use. Two common solutions mainly used, CUDA and OpenCL. CUDA, or the Compute Unified Device Architecture, is framework for creating and running parallel applications on Nvidia GPUs. OpenCL provides a more generic and open framework for parallel computation on CPUs and GPUs, and is available for a number of Nvidia, AMD, and Intel based GPUs and CPUs. While OpenCL provides a more mobust and portalbe solution, most HPC applications utilizing GPUs are based on the faster CUDA framework. As such, we focus only on Nvidia based CUDA-capable GPUs as they can offer the best support for a wide array of programs, although this work is not strictly limited to Nvidia GPUs.

To enable PCI Passthrough, a very specific host environment must be implemented. In this work we utilize the Xen 4.2.2 hypervisor on Centos 6.4 and a custom 3.4.50-8 Linux kernel with Dom0 Xen support. Within the Xen hypervisor, GPU devices are seized upon boot and assigned to the xen-pciback kernel module. This process blocks the host devices form loading the Nvidia or nouveau drivers, keeping the GPUs uninitialized.

Xen, like other hypervisors, provides a standard method of passing through PCI devices to guest VMs upon creation. When assigning a GPU to a new VM, Xen loads a specific VGA BIOS to properly initialize the device enabling DMA and interrupts to be assigned to the guest VM. Xen also relinquishes control of the GPU via the xen-pciback module. From there, the Linux Nvidia drivers are loaded and the device is able to be used as expected within the guest. Upon VM termination, the xen-pciback module re-seizes the GPU and the devices can be re-assigned to new VMs in the future.

This mechanism of PCI-passthrough for GPUs can be implemented using multiple devices per host, as illustrated in Figure 1. Here, we see how the device's connection to the VM totally bypasses the Dom0 host as well ass the Xen VMM, and managed by VT-d or IOMMU for accessing the PCI-Express bus which the GPUs utilize. This is in contrast almost all other virtual device usage, where hardware is emulated by the host and shared accross all guests. This is the common usage for Ethernet controllers and input devices to enable users to interact with VMs as they would with native hosts.

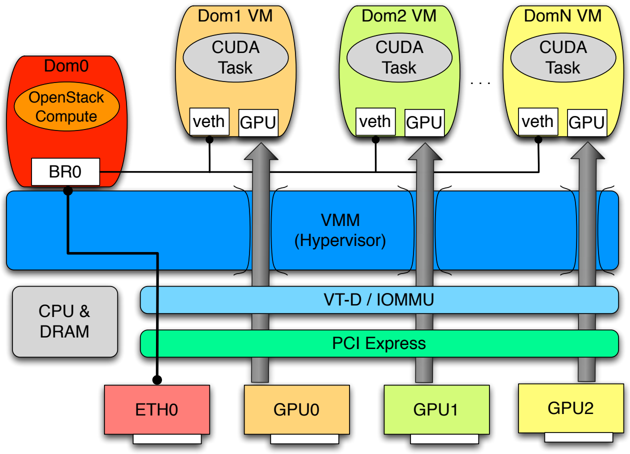

Figure 1: GPU PCI Passthrough within the Xen Hypervisor

The potential downside of this method is there can be a 1:1 or 1:N mapping of VMs to GPUs only. A M:1 mapping where multiple VMs use a GPU is not possible. However, almost all scientific applications environments using GPUs generally do not share GPUs between processes or other nodes, as doing so would cause unpredictable and serious performance degradation. As such, this GPU isolation within a VM can be considered an advantage, especially when IaaS VM scheduling mechanisms can proactively schedule heterogeneous hardware.

3.1 Feature Comparison

This proposed solution has a number of advantages compared to front-end API implementations. First, it allows for an operating environment that most closely relates to native bare-metal usage of GPUS. Essentially, a VM provides a nearly identical infrastructure to clusters and supercomputers with integrated GPUs. This lowers the learning curve for many researchers, and even enables researchers to potentially use other tools within a VM that might not be supported within a supercomputer or cluster. Unlike with remote API implementations, users don't need to decompile or modify their code, as the GPUs are essentially local to the computation. Further comparing to remote API implementations, using PCI passthrough within a VM allows users to leverage any GPU framework available, either OpenCL to CUDA, and any higher level programming frameworks such as Matlab or Python.

Through the use of advanced scheduling techniques within Cloud infrastructure, we can also take advantage of pci-passthrough implementation for efficiency purposes. Users could request VMs with GPUs which get scheduled for creation on machines that provide such resources, but can also request normal VMs as well. The scheduler can correctly map VM requirement requests to heterogeneous hardware. This enables large scale resources to support a wide array of scientific computing applications without the added cost of putting GPUs in all compute nodes.

4. EXPERIMENTAL SETUP

In this manuscript back-end GPU PCI Passthrough to virtual machines using the Xen hypervisor is described. While we can confirm this method as working though multiple Tesla based GPUs, proper evaluation of the performance of our method needs to be properly considered. As such, we run an array of benchmarks that evaluate the performance of our method compared to the same hardware running native bare-metal GPU code without any virtualization. As our method could be scaled up to many cloud compute nodes, we focus our tests on single-node performance to best understand low level overhead.

To evaluate our design, two different machines were used to represent present and upcoming Nvidia GPUs. The first system at Indiana University consists of dual-socket Intel Xeon X5660 6-core CPUs at 2.8Ghz with 192GB DDR3 RAM, 8TB RAID5 array, and two Nvidia Tesla "Fermi" C2075 GPUs. The system at USC/ISI uses a similar dual-socket Intel Xeon E5-2670 8-core CPUs at 2.6Ghz with 48GB DDR3 RAM, 3 600GB SAS disk drives, but with a prototype Nvidia Tesla "Kepler" K20m GPU supplied by Nvidia. These machines represent the present Fermi series GPUs along with the recently release Kepler series GPUs, providing a well-rounded experimental environment. Native systems were standard RHEL 6.4 with a stock 2.6.32-279 Linux kernel, whereas Xen host systems were still loaded with RHEL6.4 but with a custom 3.4.53-8 Linux kernel and Xen 4.2.22. Guest VMs were also RHEL6.4 with N-1 processors and 24GB of memory and 1 GPU passed through in HVM full virtualization mode. Using both IU and USC/ISI machine configurations both in native and VM modes represent the 4 test cases for our work.

In order to evaluate the performance, the SHOC Benchmark suite [8] was used to extensively evaluate performance across each test case. This was compiled using the NVCC compiler in CUDA5 driver and library, along with OpenMPI 1.4.2 and GCC 4.4. Each benchmark was run a total of 20 times, with the results given as an average of all runs along with 95% confidence intervals as error bars.

5. RESULTS

Results of all benchmarks are compressed into three subsections; Flops, device bandwidth and pci bus performance. Each represents a different level of evaluation for GPU-neabled VMs compared to bare-metal native GPU usage.

5.1 Flops

Flops, or floating point operations per second, are a common measure of computational performance, especially within scientific computing. The Top 500 list [10] has been the relative gold standard for evaluating supercomputing performance for more than two decades. With the advent of GPUs in some of the fastest supercomputers today, including the exact same GPUs ued in our experimentation, understanding virtualized performance relative to this metric is imperative to its relative sucess.

Figure 2 describes raw peak FLOPs available using each GPU in both native and virtualized modes. First, we the obvious advantage of using the Kepler series GPUs over Fermi, with peak single precision speeds tripling in each case. Even for double precision flops, there is rougly a doubling of performance with the new GPUs. For our research, however, its most interesting that there between a 0 and 2.9% overhead when using our GPU VMs compare to normal. Furthermore, the K20m-enabled VM was still able to provide over 3 Teraflops within a single node, a notable computational feat for any single node.

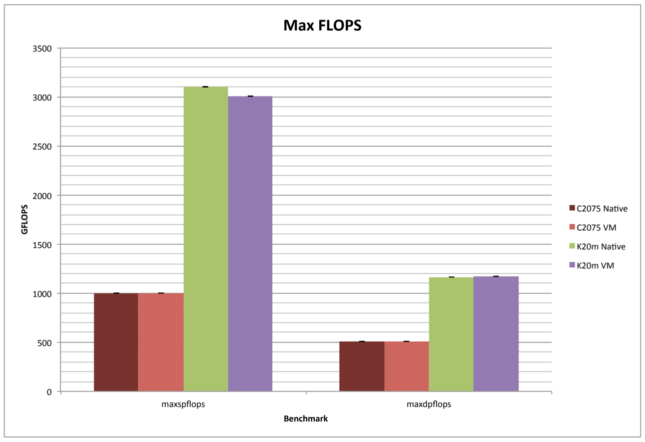

Figure 2: GPU Floating Point Operations per Second

Next, we investigate Figure 3 which exams less synthetic benchmarks of Fast Fourier Transform (FFT), and the traditionally HPC-centric Matrix Multiplication implementations. For all benchmarks that do not take into account the PCI-express bus transfer time, we again see notable speed-ups when using the Kepler GPUs. Interestingly, we do see some performance impact when using both GPU-enabled VMs compared to native. This performance decrease ranges significantly, from 0% in some cases to over 30% with single precision FFT using the K20m, with most of the FFT benchmarks seeing some overhead. The Matrix Multiplication based benchmarks performed relatively well with both Tesla GPUs, seeing a 0.1% to 7.4% performance impact, with the average overehead of just 2.9% compared to the native implementation.

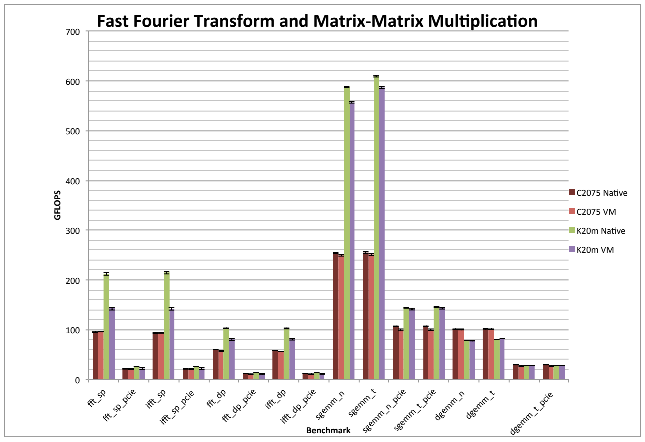

Figure 3: GPU Fast Fourier Transform and Matrix Multiplication

Other FLOP-based benchmarks emulate higher level applications. Stencil represents a 9-point stencil operation applied to a 2D data set, and the S3dD benchmark is a computationally-intensive kernel from the S3D turbulent combustion simulation program [15]. From the results in Figure 4, we notice the Fermi series GPU-enabled VMs perform roughly at near-native performance across all runs. The Kepler VMs see slightly more overhead when compared to native performance, but overhead is at most 5.8% for the Stencil benchmark, and still outperforms a Kepler series native GPU.

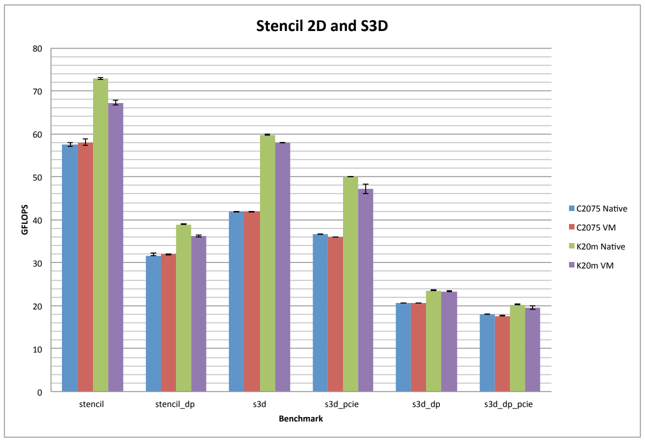

Figure 4: GPU Stencil and S3d

5.2 Device Speed

While floating point operations allow for the proposed solution to relate to many traditional HPC applications, they are just one facet of GPU performance within scientific computing. Device speed, measured in both raw bandwidth and additional 3rd part benchmarks, provide a new perspective towards evaluating our solution. Figure 5 illustrates device level memory access of various GPU device memory. With both Nvidia GPUs, we see our solution of virtualizing by and large has very little to no impact on the performance of inter-device memory bandwidth. While MD, Reduction and Scan benchmarks in Figure 6 have some variance accross runs, generally there is minimal or no significant performance impact working within our GPU-enabled VM.

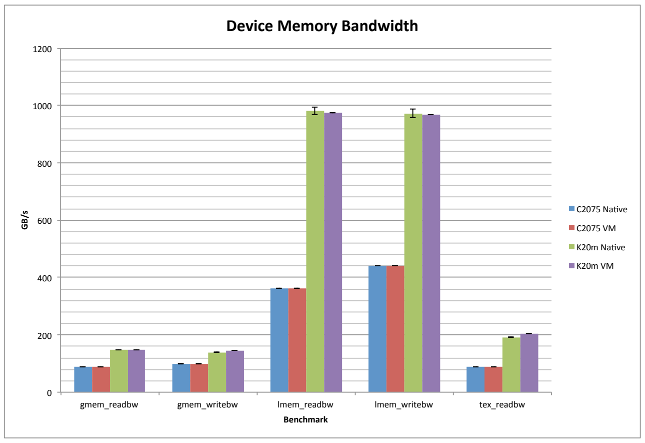

Figure 5: GPU Device Bandwidth

5.3 PCI Express Bus

Upon evaluating our solution for using GPUs in Clouds, its apparent that the PCI express bus was subject to the greatest potential for overhead. This is because the VT-d and IOMMU chip instruction sets interface directly with the PCI bus to provide operational and security related mechanisms for each PCI device, thereby ensuring proper function in a multi-guest environment. As such, its imperative to investigate any and all overhead at the PCI Express bus.

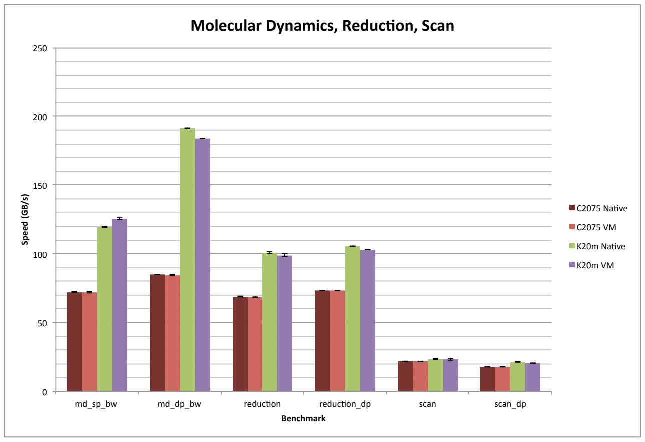

Figure 6: GPU Molecular Dynamics, Reduction, and Scan

Figure 7 looks at maximum PCI bus speeds for each experimental implementation. While we can generally see the newer Kepler's better utilize the PCI-Express 2.0 bus lanes, both cards are approaching the maximum available bandwidth provided by PCI Express. More interestingly, we see a consistent but very small overhead in our implementation. In the Fermi chipsets, we see roughly 5.6% overall impact in the bus speeds between native and VM operations. With the newer Kepler series GPUs, that overhead is significantly decreased to just .2% for combined download and readback.

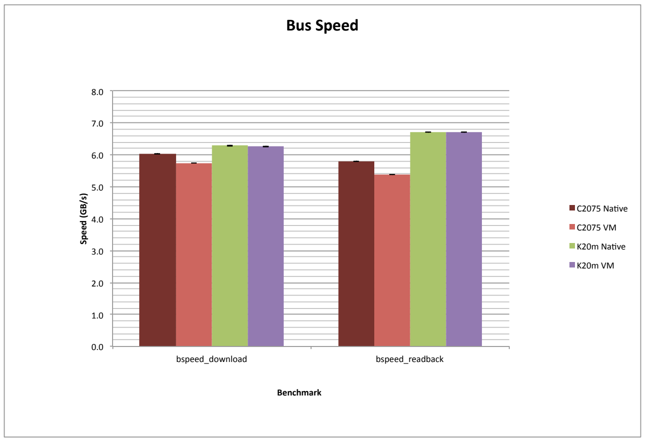

Figure 7: GPU PCI Express Bus Speed

6. DISCUSSION

This manuscript sets forth a new solution for providing advanced general purpose GPUs within Cloud computing infrastructure, primarily targeted towards advanced scientific computing. Our solution employs direct passthrough of a PCI-based GPUs directly to a guest virtual machine running on the Xen hypervisor. Evaluating the results in the previous section, a number of things become clear.

First, we can see that there is a small but notable overhead in using PCI Passthrough of GPUs within VMs, compared to native bare-metal usage, which represents the best possible use case. As with all abstraction layers, some overhead is usually inevitable as a necessary trade-off to added feature sets and improved usability. With GPUs in VMs the same is true. For all SHOC benchmarks tested, the performance impact for using a Kepler series K20m is still significantly less than the just cost of using the previous generation Fermi series C2075 GPU natively. While performance varies depending on application, there is largely a constant yet small overhead at the PCI Express bus, as well as the benchmark's natural performance impact of running within a VM itself. It is the author's hypothesis that the small overhead will largely go unnoticed by most mid-level scientific computing applications.

Our method also has the ability to perform better than other front-end API solutions applicable within VMs, such as gVirtus or rCUDA. While not tested directly, previous research has shown that such solutions have a much larger overhead compared to bare-metal use. For instance with rCUDA's optimal configuration using QDR InfiniBand, the maximum theoretical bus speed is 40Gbs. This is considerably less the roughly 54.4Gps real-world performance measured between host-to-device transfers with our GPU-enabled Kepler VMs.

7. CONCLUSION AND FUTURE WORK

The ability to use GPUs within virtual machines represents a leap forward for supporting advanced scientific computing within Cloud infrastructure. Our method of direct PCI-Passthrough of Nvidia GPUs using the Xen hypervisor offers a clean, reproducible solution that can be implemented within many Infrastructure-as-a-Service (IaaS) deployments. Performance measurements indicate that the overhead of virtualizing a GPU within Xen is minimal compared to the best-case native use, with the average performance degradation of just 2.8% for Fermi GPUs and 4.9% for Kepler GPUs.

Current and future work with this technology revolves around enabling GPU-based PCI Passthrough within the OpenStack nova IaaS framework. This will enable research laboratories and institutions to create new private Cloud offerings that have the ability to support a plethora of new scientific computing. Furthermore, we hope to integrate this work with advanced interconnects and other heterogeneous hardware to provide a parallel high performance cloud infrastructure.

Acknowledgment

This document was developed with support from the National Science Foundation (NSF) under Grant No. 0910812 to Indiana University for "FutureGrid: An Experimental, High Performance Grid Test-bed" Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF. Andrew J. Younge also greatly acknowledges support from The Persistent Systems Fellowship of the School of Informatics and Computing at Indiana University.

8. REFERENCES

[1] Amazon. Elastic Compute Cloud.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical report, University of California at Berkeley, February 2009.

[3] Windows azure platform. [Online]. `http://www.microsoft.com/azure/`.

[4] K. J. Barker, K. Davis, A. Hoisie, D. J. Kerbyson, M. Lang, S. Pakin, and J. C. Sancho. Entering the petaflop era: the architecture and performance of roadrunner. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, page 1. IEEE Press, 2008.

[5] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, and K. Skadron. A performance study of general-purpose applications on graphics processors using cuda. *Journal of Parallel and Distributed Computing*, 68(10):1370 – 1380, 2008. K-means implementation on CUDA with 72x speedup. Compares to 4-threaded CPU version with 30x speedup.

[6] S. Crago, K. Dunn, P. Eads, L. Hochstein, D.-I. Kang, M. Kang, D. Modium, K. Singh, J. Suh, and J. P. Walters. Heterogeneous cloud computing. In *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*, pages 378–385. IEEE, 2011.

[7] S. Craven and P. Athanas. Examining the viability of fpga supercomputing. *EURASIP Journal on Embedded systems*, 2007(1):13–13, 2007.

[8] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. 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.

[9] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[10] J. Dongarra, H. Meuer, and E. Strohmaier. Top 500 supercomputers. website, November 2013.

[11] J. Duato, A. J. Pena, F. Silla, J. C. Fernández, R. Mayo, and E. S. Quintana-Orti. Enabling cuda acceleration within virtual machines using rcuda. In *High Performance Computing (HiPC), 2011 18th International Conference on*, pages 1–10. IEEE, 2011.

[12] J. Duato, A. J. Pena, F. Silla, R. Mayo, and E. S. Quintana-Orti. rcuda: Reducing the number of gpu-based accelerators in high performance clusters. In *High Performance Computing and Simulation (HPCS), 2010 International Conference on*, pages 224–231. IEEE, 2010.

[13] I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud Computing and Grid Computing 360-Degree Compared. In *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10, 2008.

[14] G. Giunta, R. Montella, G. Agrillo, and G. Coviello. A gpgpu transparent virtualization component for high performance computing clouds. In P. DâĂŹAmbra, M. Guarracino, and D. Talia, editors, *Euro-Par 2010 - Parallel Processing*, volume 6271 of *Lecture Notes in Computer Science*, pages 379–391. Springer Berlin Heidelberg, 2010.

[15] E. R. Hawkes, R. Sankaran, J. C. Sutherland, and J. H. Chen. Direct numerical simulation of turbulent combustion: fundamental insights towards predictive models. In *Journal of Physics: Conference Series*, volume 16, page 65. IOP Publishing, 2005.

[16] S. Hong and H. Kim. An integrated gpu power and performance model. In *ACM SIGARCH Computer Architecture News*, volume 38, pages 280–289. ACM, 2010.

[17] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and J. T. Klosowski. Chromium: a stream-processing framework for interactive rendering on clusters. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 693–702. ACM, 2002.

[18] K. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. Wasserman, and N. Wright.

Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud. In *2nd IEEE International Conference on Cloud Computing Technology and Science*, pages 159–168. IEEE, 2010.

[19] K. Keahey, I. Foster, T. Freeman, and X. Zhang. Virtual workspaces: Achieving quality of service and quality of life in the grid. *Scientific Programming Journal*, 13(4):265–276, 2005.

[20] V. V. Kindratenko, J. J. Enos, G. Shi, M. T. Showerman, G. W. Arnold, J. E. Stone, J. C. Phillips, and W.-m. Hwu. Gpu clusters for high-performance computing. In *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*, pages 1–8. IEEE, 2009.

[21] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, et al. Exascale computing study: Technology challenges in achieving exascale systems. 2008.

[22] Z. Liu and W. Ma. Exploiting computing power on graphics processing unit. volume 2, pages 1062–1065, Dec. 2008.

[23] S. Long. Virtual machine graphics acceleration deployment guide. Technical report, VMWare, 2013.

[24] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The eucalyptus open-source cloud-computing system. In *Proceedings of Cloud Computing and Its Applications*, October 2008.

[25] C. Nvidia. Programming guide, 2008.

[26] L. OpenStack. Openstack compute administration manual, 2013.

[27] J. Sanders and E. Kandrot. *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.

[28] G. Shainer, T. Liu, J. Layton, and J. Mora. Scheduling strategies for hpc as a service (hpcaas). In *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*, pages 1–6. IEEE, 2009.

[29] L. Shi, H. Chen, J. Sun, and K. Li. vcuda: Gpu-accelerated high-performance computing in virtual machines. *Computers, IEEE Transactions on*, 61(6):804–816, 2012.

[30] J. E. Stone, D. Gohara, and G. Shi. Opencl: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering*, 12(3):66, 2010.

[31] A. S. Tanenbaum and M. Van Steen. *Distributed systems*, volume 2. Prentice Hall, 2002.

[32] G. von Laszewski, G. C. Fox, F. Wang, A. J. Younge, A. Kulshrestha, G. G. Pike, W. Smith, J. Vockler, R. J. Figueiredo, J. Fortes, et al. Design of the futuregrid experiment management framework. In *Gateway Computing Environments Workshop (GCE), 2010*, pages 1–10. IEEE, 2010.

[33] W. Wade. How nvidia and citrix are driving the future of virtualized visual computing. [Online]. `http://blogs.nvidia.com/blog/2013/05/22/synergy/`.

[34] L. Wang, G. von Laszewski, A. J. Younge, X. He, M. Kunze, and J. Tao. Cloud Computing: a Perspective Study. *New Generation Computing*, 28:63–69, Mar 2010.

[35] K. Yelick, S. Coghlan, B. Draney, and R. S. Canon. The Magellan Report on Cloud Computing for Science. Technical report, U.S. Department of Energy Office of Science Office of Advanced Scientific Computing Research (ASCR), Dec. 2011.

[36] A. J. Younge, R. Henschel, J. T. Brown, G. von Laszewski, J. Qiu, and G. C. Fox. Analysis of Virtualization Technologies for High Performance Computing Environments. In *Proceedings of the 4th International Conference on Cloud Computing (CLOUD 2011)*, Washington, DC, July 2011. IEEE.