Evaluating GPU Passthrough in Xen for High Performance Cloud Computing

Andrew J. Younge[1], John Paul Walters[2], Stephen Crago[2], Geoffrey C. Fox[1]
[1]Pervasive Technology Institute, Indiana University
2719 E 10th St., Bloomington, IN 47408, U.S.A.
[2]Information Sciences Institute, University of Southern California
3811 Fairfax Dr. Suite 200, Arlington, VA 22203, U.S.A.
Email corresponding author: ajyounge@indiana.edu

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 concern about a performance gap that exists between the performance of IaaS when compared to typical high performance computing (HPC) resources, which could limit 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 computing. We look to bridge the gap between supercomputing and clouds by providing GPU-enabled virtual machines (VMs) and investigating their feasibility for advanced scientific computation. Specifically, the Xen hypervisor is utilized to leverage specialized hardware-assisted I/O virtualization and PCI passthrough in order to provide advanced HPC-centric Nvidia GPUs directly in guest VMs. This methodology is evaluated by measuring the performance of two Nvidia Tesla GPUs within Xen VMs and comparing to bare-metal hardware. Results show PCI passthrough of 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.

I. Introduction

Cloud computing [3] has established itself as a prominent paradigm within the realm of Distributed Systems [34] 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 [37].

While the origin of cloud computing is based in industry through solutions such as Amazon's EC2 [2], Google's MapReduce [9], and Microsoft's Azure [1], 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 [35], the Magellan project [39], and through various other Infrastructure-as-a-Service projects including OpenStack [28], Nimbus [21], and Eucalyptus [26].

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 upwards of two orders of magnitude. This is accomplished through dedicated programming environments, compilers, and libraries such as CUDA [27] from Nvidia as well as the OpenCL effort [33]. 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], [24], ranging from Astrophysics to Bioinformatics. Furthermore, these gains in computational power have also reportedly come at an increased performance-per-watt [17], a metric that is increasingly important to the HPC community as we move closer to exascale computing [23] 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 the options currently available for providing GPUs within a virtualized cloud environment. In Section 3, we discuss the methodology for providing GPUs directly within virtual machines. In Section 4 we outline the evaluation of the given methodology 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 work.

II. 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 led to an explosion of new GPU-specific applications. In some cases, GPUs outperform even the fastest multicore counterparts by an order of magnitude [29]. In addition, further research could leverage the per-node performance of GPU accelerators with the high speed, low latency interconnects commonly utilized in supercomputers and clusters to create a hybrid GPU + MPI class of applications. The number of distributed GPU applications is increasing substantially in supercomputing, usually scaling many GPUs simultaneously [22].

Since the establishment of cloud computing in industry, research groups have been evaluating its applicability to science [14]. 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 led 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 [39]. An overarching goal exists to provide HPC Infrastructure as its own service (HPCaaS) [30], aiming to classify and limit the overhead of virtualization, and reducing the bottlenecks classically found in CPU, memory, and I/O operations within hypervisors [19], [40]. 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 [36]. However, this is currently not targeted towards HPC applications.

The task of providing 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.

A. *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 [25] 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 [18], 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 [31] 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 mode using XMLRPC, as well as a sharing mode that is built on VMRPC, a dedicated remote procedure call architecture for VMM platforms. This share model can leads to 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 [15]. 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. Recent work has also enabled the dynamic sharing of GPUs by leveraging the gVirtus back-end system with relatively good results [10], however process-level GPU resource sharing is outside the scope of this manuscript.

rCUDA [12], [13], 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 all 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 when using the standard QDR InfiniBand.

B. Back-end PCI passthrough

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. This manuscript focuses on such an 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 from Intel and AMD, respectively. This mechanism, typically called PCI passthrough, uses 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 has recently become 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 nontrivial. 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 [25]. In our experimentation, we have found that the Xen hypervisor provides a good platform for performing PCI passthrough of GPU devices to VMs due to its open nature, extensive support, and high degree of reconfigurability. Work with Xen in [38] gives hints at good performance for PCI passthrough in Xen, however further evaluation with independent benchmarks is needed when looking at scientific computing with GPUs.

Today's GPUs can provide a variety of frameworks for application programmers to use. Two common solutions are CUDA and OpenCL. CUDA, or the Compute Unified Device Architecture, is a 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 GPUs and CPUs. While OpenCL provides a more robust and portable solution, many HPC applications utilize the CUDA framework. As such, we focus only on Nvidia based CUDA-capable GPUs as they offer the best support for a wide array of programs, although this work is not strictly limited to Nvidia GPUs.

III. Implementation

In this manuscript we use a specific host environment to enable PCI passthrough. First, we start with 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 and therefore able to be assigned to DomU VMs.

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.

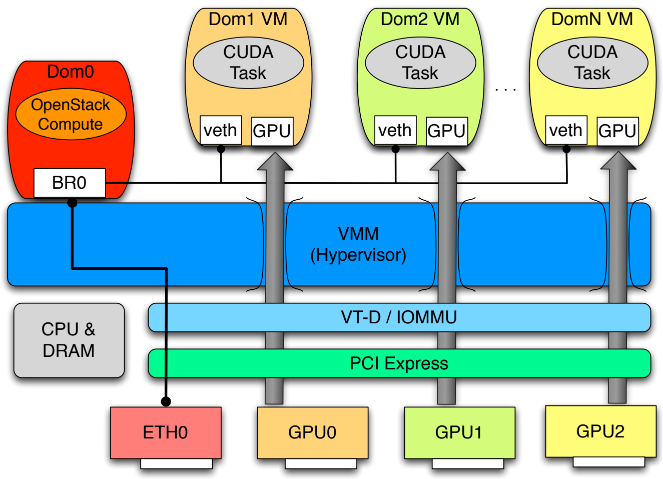

Fig. 1. GPU PCI passthrough within the Xen Hypervisor

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 as the Xen VMM, and is managed by VT-d or IOMMU to access the PCI-Express bus which the GPUs utilize. This is in contrast to other common virtual device uses, where hardware is emulated by the host and shared across 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, unlike the bridged model shown in the figure. 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 in many contexts.

A. Feature Comparison

Using the GPU PCI passthrough technique described previously has a number of advantages compared to front-end API implementations. First, it allows for an operating environment that more 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 recompile or modify their code, as the GPUs are essentially local to the data. Further comparing to remote API implementations, using PCI passthrough within a VM allows users to leverage any GPU framework available, OpenCL or CUDA, and any higher level programming frameworks such as within 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.

IV. Experimental Setup

In this manuscript back-end GPU PCI passthrough to virtual machines using the Xen hypervisor is detailed, however proper evaluation of the performance of such method needs to be properly considered. As such, we ran an array of benchmarks that evaluate the performance of this method compared to the same hardware running native bare-metal GPU code without any virtualization. We focus our tests on single-node performance to best understand low level overhead.

To evaluate the effectiveness of GPU-enabled VMs within Xen, two different machines were used to represent two generations of 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 dual-socket Intel Xeon E5-2670 8-core CPUs at 2.6Ghz with 48GB DDR3 RAM, 3 600GB SAS disk drives, but with the latest 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 installed with standard Centos 6.4 with a stock 2.6.32-279 Linux kernel. Xen host systems were still loaded with Centos 6.4 but with a custom 3.4.53-8 Linux kernel and Xen 4.2.22. Guest VMs were also Centos 6.4 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 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 platform. The SHOC benchmarks were chosen because they provide a higher level of evaluation regarding GPU performance than the sample applications provided in the Nvidia SDK, and can also evaluate OpenCL performance in similar detail. The benchmarks were 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.

V. Results

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

A. Floating Point Performance

Flops, or floating point operations per second, are a common measure of computational performance, especially within scientific computing. The Top 500 list [11] 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 GPUs identical to those used in our experimentation, understanding the performance relative to this metric is imperative.

Figure 2 shows the raw peak FLOPs available using each GPU in both native and virtualized modes. First, we observe the 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 roughly a doubling of performance with the new GPUs. However its most interesting that there is less than a 1% overhead when using GPU VMs compared to native case for pure floating point operations. The K20m-enabled VM was able to provide over 3 Teraflops, a notable computational feat for any single node.

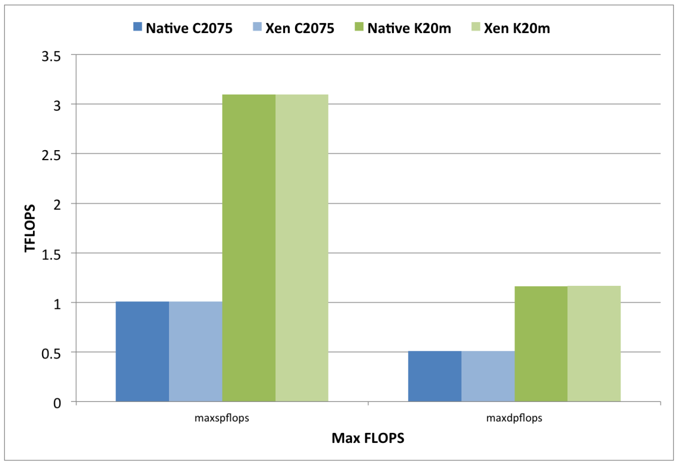

Fig. 2. GPU Floating Point Operations per Second

Figures 3 and 4 show Fast Fourier Transform (FFT) and the Matrix Multiplication implementations across both test platforms. For all benchmarks that do not take into account the PCI-Express (pcie) bus transfer time, we again see near-native performance using the Kepler GPUs and Fermi GPUs when compared to bare metal cases. Interestingly, overhead within Xen VMs is consistently less than 1%, confirming the synthetic MaxFlops benchmark above. However, we do see some performance impact when calculating the total FLOPs with the pcie bus in the equation. This performance decrease ranges significantly for the C2075-series GPU, roughly about a 15% impact for FFT and a 5% impact for Matrix Multiplication. This overhead in pcie runs is not as pronounced for the Kepler K20m test environment, with near-native performance in all cases (less than 1%).

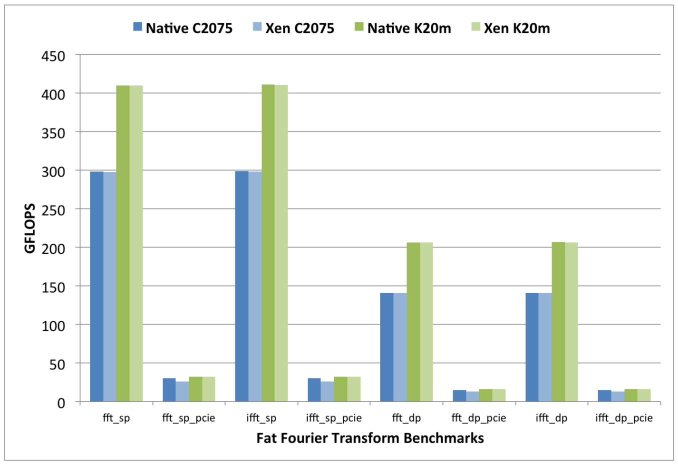

Fig. 3. GPU Fast Fourier Transform

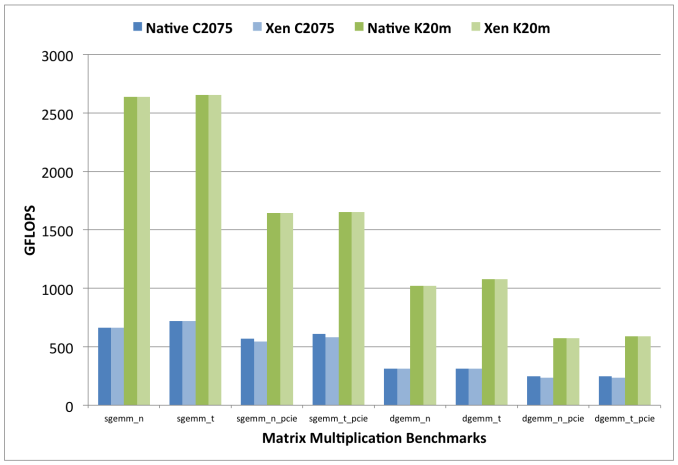

Fig. 4. GPU Matrix Multiplication

Other FLOP-based benchmarks are used to emulate higher level applications. Stencil represents a 9-point stencil operation applied to a 2D data set, and the S3D benchmark is a computationally-intensive kernel from the S3D turbulent combustion simulation program [16]. In Figure 5, we see that both the Fermi C2075 and Kepler K20m GPUs performing well compared to the native base case, showing the overhead of virtualization is low. The C2075-enabled VMs experience slightly more overhead when compared to native performance again for pcie runs, but overhead is at most 7% for the S3D benchmark.

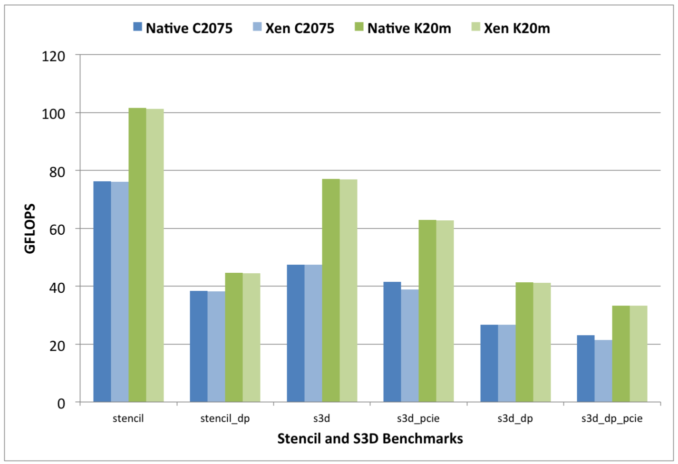

Fig. 5. GPU Stencil and S3D

B. 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 benchmarks, provides a different perspective towards evaluating GPU PCI passthrough in Xen. Figure 6 illustrates device level memory access of various GPU device memory structures. With both Nvidia GPUs, virtualization has little to no impact on the performance of inter-device memory bandwidth. As expected the Kepler K20m outperformed the C2075 VMs and there was a higher variance between runs with both native and VM cases. Molecular Dynamics and Reduction benchmarks in Figure 7 perform again at near-native performance without the pcie bus taken into account. However the overhead observed increases to 10-15% when the PCI-Express bus is considered when looking at the Fermi C2075 VMs.

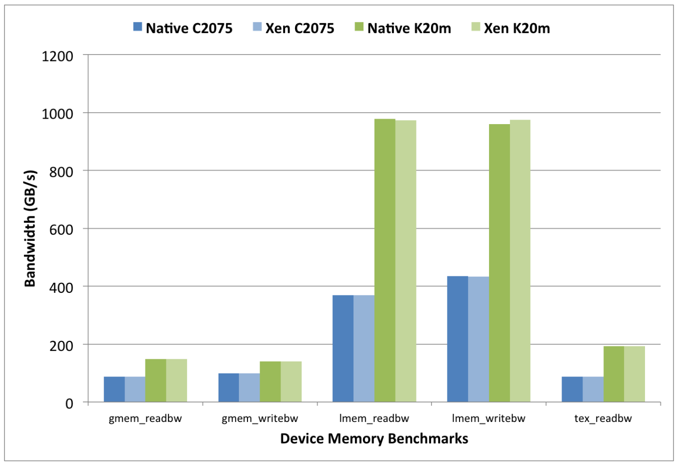

Fig. 6. GPU Device Memory Bandwidth

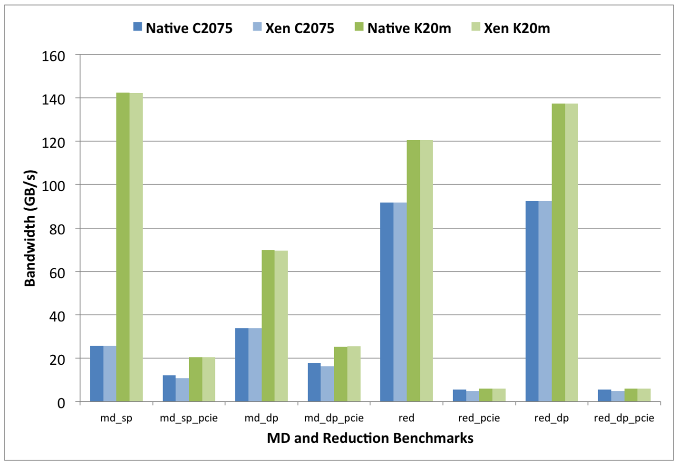

Fig. 7. GPU Molecular Dynamics and Reduction

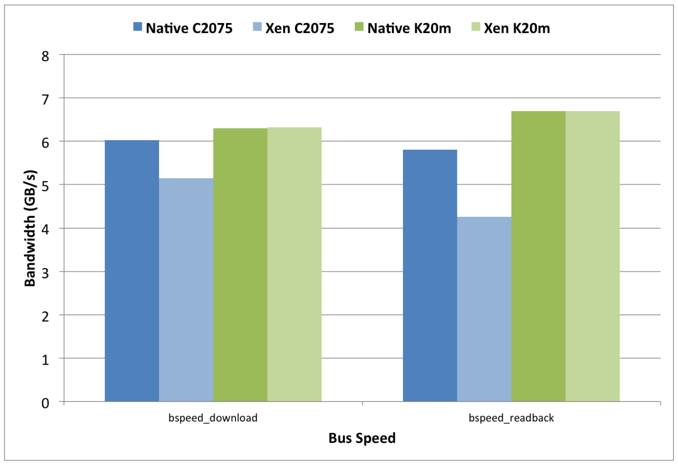

Fig. 8. GPU PCI Express Bus Speed

C. PCI Express Bus

Upon evaluating PCI passthrough of GPUs, it is apparent that the PCI express bus is subject to the greatest potential for overhead, as was observed in the Fermi C2075 benchmarks. 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 but potentially introducing some overhead. As such, it is imperative to investigate any and all overhead at the PCI Express bus.

Figure 8 looks at maximum PCI bus speeds for each experimental implementation. First, we see a consistent overhead in the Fermi C2075 VMs, with a 14.6% performance impact for download (to-device) and a 26.7% impact in readback (from-device). However, the Kepler K20 VMs do not experience the same overhead in the PCI-Express bus, and instead perform at near-native performance. These results indicate that the overhead is minimal in the actual VT-d mechanisms, and instead due to other factors.

Upon further examination, we have identified the performance degradation within the Fermi-based VM experimental setup is due to the testing environment's NUMA node configuration with the Westmere-based CPUs. With the Fermi nodes, the GPUs are connected through the PCI-Express bus from CPU Socket #1, yet the experimental GPU VM was run using CPU Socket #0. This meant that the VM's PCI-Express communication involved more host involvement with Socket #1, leading to an notable decrease in read and write speeds across the PCI-Express Bus. Such architectural limitations seem to be relieved in the next generation with a Sandy-Bridge CPU architecture and the Kepler K20m experimental setup. In summary, GPU PCI-Express performance in a VM is sensitive to the host CPU's NUMA architecture and care is needed to mitigate the impact, either by leveraging new architectures or by proper usage of Xen's VM core assignment features. Furthermore, the overhead in this system diminishes significantly when using the new Kepler GPUs by Nvidia.

VI. Discussion

This manuscript evaluates the use of general purpose GPUs within cloud computing infrastructure, primarily targeted towards advanced scientific computing. The method of PCI passthrough of GPUs directly to a guest virtual machine running on a tuned Xen hypervisor shows initial promise for an ubiquitous solution in cloud infrastructure. In evaluating the results in the previous section, a number of points become clear.

First, we can see that there is a small 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. The same is true for GPUs within Xen VMs. The Kepler K20m GPU-enabled VMs operated at near-native performance for all runs, with a 1.2% reduction at worst in performance. The Fermi based C2075 VMs experience more overhead due to the NUMA configuration that impacted PCI-Express performance. However, when the overhead of the PCI-Express bus is not considered, the C2075 VMs perform at near-native speeds.

GPU PCI passthrough also has the potential to perform better than other front-end API solutions that are applicable within VMs, such as gVirtus or rCUDA. This is because such solutions are designed to communicate via a network interconnect such as 10Gb Ethernet or QDR InfiniBand [32], which introduces an inherent bottleneck. Even with the theoretically optimal configuration of rCUDA using QDR InfiniBand, the maximum theoretical bus speed is 40Gbs, which is comparably less than the measured 54.4Gps real-world performance measured between host-to-device transfers with GPU-enabled Kepler VMs.

Overall, it is our hypothesis that the overhead in using GPUs within Xen VMs as described in this manuscript will largely go unnoticed by most mid-level scientific computing applications. This is especially true when using the latest Sandy-Bridge CPUs with the Kepler series GPUs. We expect many mid-tier scientific computing groups to benefit the most from the ability to use GPUs in a scientific cloud infrastructure. Already this has been confirmed in [20], where similar a methodology has been leveraged specifically for Bioinformatics applications in the cloud.

VII. Conclusion and Future Work

The ability to use GPUs within virtual machines represents a leap forward for supporting advanced scientific computing within cloud infrastructure. The 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 providing a GPU within Xen is minimal compared to the best-case native use, however NUMA inconsistencies can impact performance. The New Kepler-based GPUs operate with a much lower overhead, making those GPUs an ideal choice when designing a new GPU IaaS system.

Next steps for this work could involve providing GPU-based PCI passthrough within the OpenStack nova IaaS framework. This will enable research laboratories and institutions to create new private or national-scale cloud infrastructure that have the ability to support new scientific computing challenges. Other hypervisors could also leverage GPU PCI passthrough techniques and warrant in-depth evaluation in the future. Furthermore, we hope to integrate this work with advanced interconnects and other heterogeneous hardware and provide a parallel high performance cloud infrastructure to enable mid-tier scientific computing.

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 acknowledges support from The Persistent Systems Fellowship of the School of Informatics and Computing at Indiana University.

References

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

[2] Amazon. Elastic Compute Cloud.

[3] 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.

[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 *Proceedings of the Workshop on Parallel Programming on Accelerator Clusters (PPAC). 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] K. Diab, M. Rafique, and M. Hefeeda. Dynamic sharing of GPUs in cloud systems. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International*, pages 947–954, 2013.

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

[12] 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.

[13] 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.

[14] 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.

[15] G. Giunta, R. Montella, G. Agrillo, and G. Coviello. A GPGPU transparent virtualization component for high performance computing clouds. In P. DAmbra, 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] H. Jo, J. Jeong, M. Lee, and D. H. Choi. Exploiting GPUs in virtual machine for biocloud. *BioMed research international*, 2013, 2013.

[21] 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.

[22] 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.

[23] 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.

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

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

[26] 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.

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

[28] OpenStack. Openstack compute administration manual, 2013.

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

[30] 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.

[31] 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.

[32] F. Silla. rCUDA: share and aggregate GPUs in your cluster. Mellanox Booth Presaentation, Nov. 2012.

[33] 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.

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

[35] 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.

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

[37] 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.

[38] C.-T. Yang, H.-Y. Wang, and Y.-T. Liu. Using pci pass-through for gpu virtualization with cuda. In *Network and Parallel Computing*, pages 445–452. Springer, 2012.

[39] 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.

[40] 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.