An Evaluation of CUDA-enabled Virtualization Solutions

Vinaya M S
International Institute of Information Technology
Bangalore, India
Email: vinaya.ms@iiitb.org

Nagavijayalakshmi Vydyanathan and Mrugesh Gajjar
Siemens Corporate Research and Technology
Bangalore, India
Email: {nagavijayalakshmi.vydyanathan, mrugesh.gajjar}@siemens.com

***Abstract*—Virtualization, as a technology that enables easy and effective resource sharing with a low cost and energy footprint, is becoming increasingly popular not only in enterprises but also in high performance computing. Applications with stringent performance needs often make use of graphics processors for accelerating their computations. Hence virtualization solutions that support GPU acceleration are gaining importance. This paper performs a detailed evaluation of three frameworks: rCUDA, gVirtuS and Xen, which support GPU acceleration through CUDA, within a virtual machine. We describe the architectures of these three solutions and compare and contrast them in terms of their fidelity, performance, multiplexing and interposition characteristics.**

***Index Terms*—virtualization; gpgpu computing; cuda; gpu virtualization; many-core computing**

I. Introduction

Virtualization can be broadly defined as a technology that applies an abstraction layer to demarcate the computing resources from the software that runs on the resource. This abstraction allows concurrent execution of multiple, possibly different, isolated execution environments on the same physical computing resource, thereby enabling efficient resource sharing among applications of different users and maintaining high system utilization. Additionally, the abstraction from the underlying physical resources, host operating systems etc., facilitates legacy applications to run as-is on virtual machines, even when the computing system is upgraded.

Due to its support for flexible OS variety and isolated execution environments, efficient fine-grained resource sharing, enhanced productivity, reliability and availability, virtualization has been gaining popularity in both enterprises as well as high performance computing domains [1], [2]. Over the last few years, the rapid growth of this technology has resulted in a paradigm shift in computing trends towards on-demand computing, cloud computing and software-as-a-service models. Industries and institutions prefer to lease computing facilities from cloud vendors like Amazon, Microsoft, and Google, and pay only for the computing resources they consume, rather than make heavy investments on infrastructure and maintenance. Applications are hosted on virtual machines on the cloud and may leverage a 'pay-per-use' business model to increase volume of sales and generate revenue.

Vinaya M S is currently affiliated with NetApp. Inc., Bangalore, India

Classical virtualization predominantly addresses processor, network and storage virtualization. However, with the evolution of GPUs as massively parallel many-core devices that can address the growing performance needs of applications across various domains like medical imaging, bio-informatics, computational finance and scientific computing, GPU virtualization is gaining significance. The availability of hardware devices like GPUs through device emulation, para-virtualization and device dedication to the guest virtual machines in the virtualization environment has enabled different methodologies to virtualize the GPU. The hardware assistance provided by Intel's VT-d eliminates much of the virtualization overheads and thus provides a simple mechanism to access the GPU [3].

However, inspite of the above factors, GPU virtualization, especially for general purpose computing, is still a nascent area that is being explored actively by both industry and academia. NVIDIA has very recently announced the first fully virtualized GPU accelerated desktop virtualization solution using the NVIDIA VGX technology that supports a fully virtualized Kepler GPU [4]. This solution is in its beta stage of development and is yet to be released as a commercial product.

The paper presents an overview of three frameworks: a) rCUDA, b) gVirtuS and c) Xen hypervisor, and compare and contrast them with respect to their fidelity, performance, multiplexing and interposition characteristics. To the best of our knowledge, this is the first work to compare different GPGPU virtualization solutions.

II. Related Work

Though virtualization has been actively researched for more than a decade, GPU virtualization is being explored extensively only in recent years. GPU virtualization refers to the technique of exposing the features of the physical GPU to virtual machines (VMs) while preserving the illusion of a dedicated virtual GPU to each VM. As presented by Dowty and Sugerman [5], GPU virtualization can be broadly classified into front-end and back-end techniques. In front-end virtualization, the graphics driver stack resides in the host or hypervisor, and the VMs access the physical GPU through either API remoting (where the graphics calls are forwarded to the external graphics stack through remote procedure calls) or device emulation (where a virtual GPU is emulated in the VM). In back-end virtualization, the graphics driver stack resides in the VM allowing it to access the physical GPU directly. Fixed pass-through is a popular back-end virtualization technique where a physical GPU is exclusively dedicated to a VM.

GPU virtualization traditionally was targeted at virtualizing graphics rendering but recently the focus has shifted to include virtualization of GPGPU computing. A few recent frameworks have been proposed by both industry and academia for CUDA enabled virtualization (CUDA is a programming model proposed by NVIDIA that exposes the NVIDIA GPU hardware for GPGPU computing). Some of these are front-end virtualization solutions that adopt API remoting or call forwarding techniques - examples of these are vCUDA [6], rCUDA [7], [8], gVirtuS [9] and GViM [10]. Others such as Citrix XenServer [11], Xen [12] and VMware's vSphere [13] , make use of fixed pass-through mechanisms that use the hardware support of Intel's VT-d [14] . The above mentioned fixed pass-through solutions officially supports only desktop virtualization for graphics rendering and not for general purpose computations. In our study we tried Xen PCI-passthrough mechanism to accomplish the access of GPU for general purpose computations by doing certain modifications which is discussed later in Section III C. We failed to accomplish the same using VMware's vSphere server.

The front-end virtualization solutions (vCUDA, GViM, rCUDA and gVirtuS) support multiplexing of the graphics card among different VMs. In addition, vCUDA supports features like suspend and resume that enables client sessions to migrate across computers while retaining hardware acceleration capabilities [6], while GViM proposes techniques for co-ordinated sharing of GPU resources amongst VMs [10]. Vmware ESX, which employs back-end virtualization, supports GPU computing by dedicating the physical GPU device to a VM by a feature called VMDirectPath I/O [15]. This provides performance very close to native execution at the cost of lack of support for multiplexing. Commercially, GPU virtualization is used by certain cloud providers like Amazon Elastic Compute Cloud (Amazon EC2) [16] to provide GPU computing services including GPGPU computing. These currently make use of back-end virtualization with fixed pass-through mechanisms. Amazon Cluster GPU instances make use of Xen pci-passthrough [12] on xen fully virtualized environment (Xen HVM). Peer1's hosted GPU private cloud [17] makes use of pass-through while Zillianss V-GPU [18] uses API remoting.

III. CUDA-enabled Virtualization Frameworks

This section gives a system overview of rCUDA, gVirtuS and Xen PCI pass-through.

A. rCUDA

rCUDA [7], [8] is a framework intended to enable remote use of CUDA runtime APIs on GPU devices installed in a high performance cluster. This has been developed in collaboration between the Parallel Architectures Group at the Universitat Politecnica de Valencia and the High Performance Computing and Architectures Group at the Universitat Jaume I.

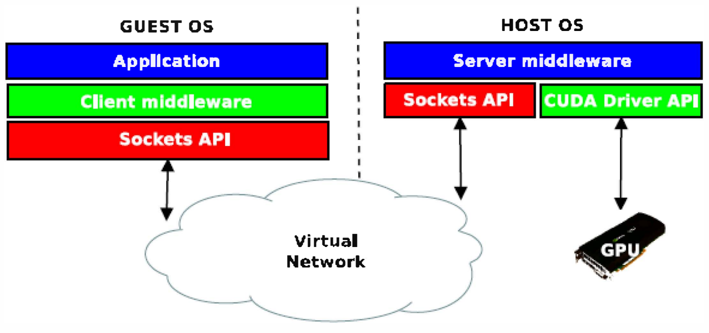

Fig. 1. rCUDA System Architecture. (Courtesy: Duato et al. [7])

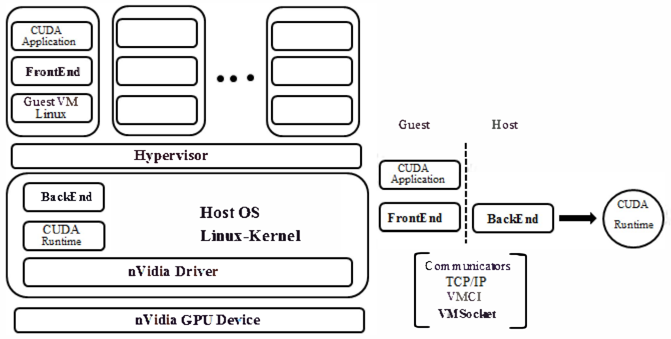

Fig. 2. gVirtuS System Architecture (left) and Components (right). (Courtesy: Giunta et al. [9])

Figure 1 depicts the rCUDA system architecture in VMM environment. rCUDA framework is a client-server distributed architecture consisting of two software modules, the client middleware and the server middleware. The client middleware consists of rCUDA wrappers to the CUDA runtime. These wrappers are responsible for forwarding the CUDA API calls made by client applications to the server middleware, retrieving the results and reporting the same back to the application, thus enabling remote access to GPUs. The server middleware is run on a machine that hosts one or more GPUs. The server is responsible for execution of CUDA API calls made from clients by employing different GPU contexts, thereby multiplexing the GPU across different clients. The client server communication is designed over TCP.

Though fundamentally rCUDA is independent of virtualization, it can also be used to enable access to the host GPU device from the virtual machines. The rCUDA server middleware daemons are run on the host that has direct access to the GPU. The client wrappers are invoked from the virtual machines to enable access to the host GPU. This technique is an example of API remoting virtualization.

B. gVirtuS

The GPU Virtualisation Service (gVirtuS) [9] is a framework that enables access to GPUs from virtual machines for general purpose computing in a transparent and hypervisor independent way.

Figure 2 shows the gVirtuS system architecture and components. Like rCUDA, gVirtuS employs API remoting virtualization technique having front-end and a back-end module. The front-end resides on the virtual machine (guest). A wrapper CUDA library residing in the guest intercepts the CUDA calls made by an application and forwards them to the front-end module. The front-end packs the library call and sends it to the back-end module. The communication between the front-end and back-end is established through a pluggable communicator component as shown in Figure 2. The back-end unpacks the library function, maps memory pointers and executes the operation on the installed GPU and returns the results to the front-end. The gVirtuS architecture is similar to rCUDA except that it is designed to accommodate a pluggable communication component independent of the hypervisor and channel and hence aims to use efficient communicators to reduce the overheads in remote execution of CUDA calls.

C. Xen PCI pass-through

Xen is a popular open source virtualization solution developed by the University of Cambridge [19]. Xen, named after neXt gENeration virtualization, is a low-overhead system that supports multiple virtual machines at close to native performance. The Xen hypervisor is a bare-metal software layer that runs directly on the physical hardware replacing the operating system, thereby controlling all the hardware requests such as CPU, memory, I/O and disk requests from one or more guest operating systems (virtual machines) running above it.

Every running instance of a VM over the xen hypervisor is called as a guest. A guest that has the privileges to access all hardware resource and manage other guests is called a privileged domain or a Domain 0 (Dom0) guest. The unprivileged guests that do not have direct access to the physical resources and are managed by the Dom0 guest are called Domain U (unprivileged) guests. A Xen installation has a single Dom0 guest and several DomU guests. In addition, Xen also allows DomU guests to access certain physical devices directly through a mechanism called PCI pass-through.

The DomU guests can be of two types: para-virtualized (PVM) guests which run modified operating systems that invoke the hypervisor; and hardware-virtualized (HVM) guests that run un-modified operating systems that are unaware of the hypervisor. HVM leverages virtualization support in the hardware such as Intel VT-x technology. Para-virtualized guests run more efficiently since they do not require device emulation. HVM guests on Xen use Qemu to emulate the computer hardware and hence it is slower in comparison. To enhance the performance of HVM guests, para-virtual device drivers called PV-on-HVM drivers [20] can be used to bypass the emulation for certain devices. For our study we have made use of HVM guests and PV-on-HVM drivers on the Dom0 kernel to get better performance.

To provide access to the GPU device from DomU, a mechanism called PCI pass-through is used. This requires IOMMU support provided by Intel VT-d [14]. This PCI pass-through mechanism employs fixed pass-through virtualization technique where a GPU device is dedicated to a virtual machine. Thus, the device cannot be shared amongst different guest and multiplexing is not supported. Figure 3 depicts the architecture of Xen PCI pass-through. The GPU driver stack runs in each DomU guest and allows the guest to directly access the GPU device. A GPU device is dedicated to each DomU guest.

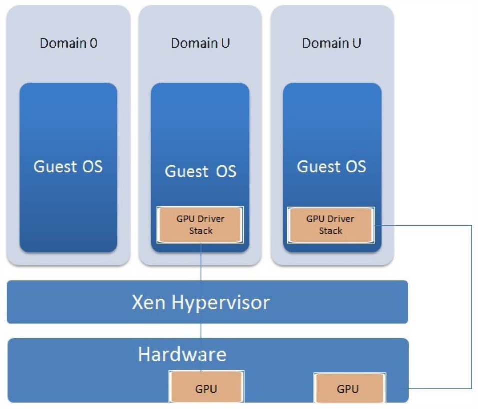

Fig. 3. Xen PCI pass-through System Architecture.

IV. Experimental Evaluation

Our experimental test-bed is a HP Z800 workstation consisting of an Intel Xeon Quad-Core Processor E5506 clocked at 2.13 GHz, 4 GB main memory and two NVIDIA GPU cards - NVIDIA Tesla C2050 for computations and NVIDIA Quadro FX 570 for graphics rendering.

Both the host and guest are installed with Ubuntu 11.04 64-bit OS. In case of rCUDA and gVirtuS, the CUDA drivers are installed in the host whereas in Xen, they are installed in the VMs. The experiments are carried out with the CUDA 4.2 toolkit and SDK and driver version 295.41 for rCUDA and Xen. For gVirtuS, CUDA toolkit and SDK version 3.2 along with driver version 260.19.21 is used since gVirtuS is not upgraded to support newer CUDA toolkits.

A. Installation Details

For the rCUDA evaluations, we installed the current release, rCUDA framework 3.2 (rCUDA 3.2) and the Open Source VMM: Qemu/KVM. The rCUDA server daemon, rCUDAd, runs on the host machine and the rCUDA clients run on the VMs. The communication between the guests (VMs) and the host is handled through simple Network Address Translation (NAT) mechanism provided by Qemu/KVM. rCUDA clients and server communicates through a customised communication protocol that makes use of sockets API. This protocol helps in optimal data exchange. [21].

To setup the gVirtuS environment, we installed the Beta 3 version of gVirtuS along with the gVirtuS-cudart module. Like in rCUDA, Qemu/KVM was used as the VMM. To enable communication between the front-end and the back-end, the communicator argument in the gvirtus.properties file residing in /usr/local/gvirtus/etc is edited with the IP address and port at which the back-end module has been started. By initializing the plugins argument in the properties file with cudart, the gvirtus.cudart module is enabled for compilation of cuda code inside gVirtuS.The back-end module is started on the host. To compile and run CUDA programs on the guest, the LD_LIBRARY_PATH environment variable is set to the path of the front-end library and PATH is set to the CUDA install path.

For the Xen installation, we used Xen hypervisor 4.1.0 [22] along with paravirt_ops(pv-ops) Dom0 kernel 2.6.32.40 downloaded from xen git [23] which is a piece of Linux kernel infrastructure which runs as a para-virtualized kernel on hypervisor. The device module of the hypervisor, called qemu-dm, is built with libpci installed so as to enable pass-through support. Dom0 is configured with the pci_stub module or pciback module so that the pass-through devices (i.e., GPU in our case) using VT-d are bound to it. The devices bound by this module can be used in any one of the guest or on the host at a time. We have used Xen HVM guests which makes it possible to use unmodified Ubuntu 11.04 OS. PV-on-HVM drivers are used on Dom0 kernel that ensures better performance. In the configuration file of the HVM guest we provide the device identity of the intended pass-through device (i.e., BDF notation of the device) to the pci argument. This will bind the device to the xen-pciback module and in turn to the guest when the guest is created using xm create command. We can even bind the device at runtime by making use of device hot-plug and hot-unplug to attach and detach the device to a HVM guest. [12]. In case of Xen, the GPU device drivers are installed in the guest.

B. Evaluation Benchmarks

To compare and contrast between the three virtualization frameworks, we use a subset of the CUDA SDK examples. Care is taken to select examples from the SDK that provide a comprehensive coverage of the CUDA APIs and range from simple to complex benchmarks spread across several domains. Table I gives a list of the benchmarks selected for our evaluations and the categories of CUDA APIs they cover (detailed description of the benchmarks is not included due to space restrictions, this can be found at the NVIDIA website). Apart from the CUDA API types mentioned in Table I, device and error handling APIs are used in most of these benchmarks. For the evaluations on Xen and gVirtuS, the selected CUDA SDK benchmarks are run as-is. However, as rCUDA does not support the C for CUDA extensions, the SDK benchmarks need to be re-written.

C. Evaluation Metrics and Analysis

We evaluate the three virtualization solutions through four evaluation metrics proposed by Dowty et. al [5]: fidelity, performance, multiplexing and interposition. Fidelity, in the context of CUDA-enabled virtualization, measures the breadth and quality of coverage of the CUDA GPGPU features by a virtualization solution. Performance measures the speed of operation of CUDA APIs while multiplexing refers to the

Benchmark Name	CUDA API Category
Aligned Types (AT)	memory
Async API (AS)	event, memory
Simple Texture (ST)	texture reference, memory
Bandwidth Test (BT)	memory
Binomial Option Pricing (BiP)	memory
Black-Scholes Option Pricing (BsP)	memory
Monte Carlo Option Pricing (McP)	memory
Concurrent Kernels (CK)	stream, memory
Fast Walsh Transform (FW)	memory
Simple Multi-copy and Compute (MC)	stream, memory
Simple Surface Write (SW)	surface reference, memory

TABLE I
DETAILS ON THE BENCHMARKS SELECTED FROM THE CUDA SDK
(COURTESY: NVIDI A.COM).

CUDA API Category	rCUDA	gVirtuS	Xen-PCI
device mgmt	Y	Y	Y
error handling	Y	Y	Y
version mgmt	Y	Y	Y
stream mgmt	Y	N	Y
event mgmt	Y	Y	Y
memory mgmt	Y	Y	Y
texture ref	Y	N	Y
surface ref	Y	N	Y
graphics interop	N	Y	Y

TABLE II
FIDELITY CHARACTERISTICS OF RCUDA, GVIRTUS AND XEN-PCI.

ability for sharing a physical GPU across multiple VMs. Interposition refers to the ability of the virtualization software to mediate access between a VM and the physical hardware (similar to device emulations).

1) Fidelity: To measure the degree of fidelity, we run the selected benchmarks (mentioned in Table I) in all the three frameworks. These results are reported in Table II. We have not evaluated the availability of graphics interoperability features since our focus is on using GPGPU capabilities over virtualization. However, we capture this in Table II based on the limitations observed.

Apart from the results reported in Table II, rCUDA in its current release, has a few limitations (as reported in the rCUDA 3.1.2 user guide [21]). It is compatible with only CUDA Toolkit version 4.0 or greater and does not support zero-copy capabilities. The rCUDA library is not thread-safe and hence multiple GPU devices need to be managed from different processes. The device and host code need to be kept in separate files and compiled separately. Also, as mentioned before, rCUDA does not support CUDA C extensions and hence existing CUDA applications need to be re-written with plain C APIs. For example, kernel launches need to be done through a sequence of API calls - cudaConfigureCall() (to configure threads and blocks), cudaSetupArgument() and cudaLaunch(), instead of the C extensions.gVirtuS on the other hand, does not support streams (cudaStreamWaitEvent()), texture reference and surface reference features of CUDA whereas Xen-PCI, supports all CUDA features.

2) Performance: To evaluate the performance characteristics of the three solutions, we record the kernel compute times, memory throughput and texture rates, as relevant, for

CUDA benchmark	Host	rCUDA	Xen-PCI
AS	33.2	288.9	40.3
BiP	29.5	43.5	30.0
BsP	1.5	20.2	1.5
McP	2.6	40.4	2.8
CK	10	273	10
FW	18.6	467.6	24.5

TABLE III
KERNEL COMPUTE TIMES IN MS FOR HOST (NON-VIRTUALIZED), RCUDA AND XEN-PCI USING CUDA 4.2.

CUDA benchmark	Host	rCUDA	Xen-PCI
AT	29.5	1.2	27.4
BT (h2d)	2.8	0.22	2
BT (d2h)	2.2	0.21	1.6
BT (d2d)	103.4	17.7	59.7
MC	5.33	0.3	5

TABLE IV
MEMORY THROUGHPUT IN GB/S FOR HOST (NON-VIRTUALIZED), RCUDA AND XEN-PCI USING CUDA 4.2.(H2D, D2H AND D2D REFERS TO HOST-TO-DEVICE, DEVICE-TO-HOST AND DEVICE-TO-DEVICE BANDWID THS)

CUDA benchmark	Host	rCUDA	Xen-PCI
ST	1464.5	6.4	919.8
SW	1310.7	6.5	907.1

TABLE V
TEXTURE RATE IN MPIXELS/S FOR HOST (NON-VIRTUALIZED), RCUDA AND XEN-PCI USING CUDA 4.2.

CUDA benchmark	Host	gVirtuS
AS	33.2	844.2
BiP	27.4	28.8
BsP	4.5	4.5
McP	2.3	3.6
CK	10	NA
FW	18	22

TABLE VI
KERNEL COMPUTE TIMES IN MS FOR HOST (NON-VIRTUALIZED) AND GVIRTUS USING CUDA 3.2.

the selected CUDA SDK benchmarks. For example, for the bandwidth test and the aligned types, we record the memory bandwidth, while for the Black Scholes Options Pricing, we record the kernel compute times. Since the evaluation of rCUDA and Xen was done on CUDA 4.2 (as rCUDA is compatible only with CUDA version 4.0 or greater), and the gVirtuS evaluations were done on CUDA 3.2 (as gVirtuS is not compatible with newer cuda toolkits), we depict these results in separate tables. In order to assess the overheads introduced by these virtualization solutions over native execution of the programs, we also show performance results on the host machine without using any of these virtualization frameworks.

Table III shows the kernel compute times for rCUDA and Xen-PCI in comparison with execution on the native host. We see that rCUDA is between 1.5x to 27x slower than native computations. This is largely due to the overheads of indirect access of the GPU device through a client-server architecture. In addition, since rCUDA does not support CUDA C extensions, kernel launches incur more overheads as these involve a number of C style API calls to configure the threads/blocks/grid and setup kernel arguments before making the kernel launch. Hence, benchmarks that have repeated kernel launches like CK and FW suffer a huge performance penalty. On the other hand, Xen-PCI has only an average of 10% slowdown in comparison to native code. This is expected as in fixed pass-through, the GPU device is dedicated to a VM and is directly accessed by the drivers installed in the VM.

In terms of memory bandwidth, we see a similar trend (see Table IV) where rCUDA shows an order of magnitude lower bandwidth in comparison to Xen-PCI which suffers an average of about 20% degradation in bandwidth. rCUDA does not support zero copy capabilities, while Xen-PCI which has direct access to the GPU, can exploit all the device capabilities. A similar trend is observed in texture performance as seen in Table V.

Table VI and Table VII shows the performance results for gVirtuS in comparison to native execution using CUDA 3.2. gVirtuS, though also based on API remoting and a client-server architecture, seems to be more efficient and suffers only about 20% degradation in compute performance in comparison to native execution. In terms of memory bandwidth, in gVirtuS, host-to-device and device-to-host bandwidths suffer by an order of magnitude similar to rCUDA. However, device-to-device bandwidths are better. Since gVirtuS does not support texture references, we do not report those results.

Next, since rCUDA and gVirtuS are client-server architectures that are independent of the virtualization framework, we compared the performance of the benchmarks on rCUDA and gVirtuS over a virtualized environment to that on rCUDA and gVirtuS on a native host. This will be an indicator of the overheads in rCUDA and gVirtuS introduced by the virtualization layer. Our experiments indicated that rCUDA over Qemu/KVM virtualization was on an average about 21% worse in performance than rCUDA on the native host. gVirtuS over Qemu/KVM virtualization was on an average about 20% worse than gVirtuS over the native host.

3) Multiplexing: Xen-PCI does not allow multiplexing of the GPU device among different VMs as it follows the fixed pass-through virtualization technique that dedicates a GPU device to a VM. rCUDA and gVirtuS on the other hand, being a client-server model that embraces the API remoting technique, naturally support multiplexing of the GPU among VMs.

To evaluate the multiplexing feature of rCUDA and gVirtuS, we ran the AS, BiP, BsP, McP and FW benchmarks on increasing number of concurrent VMs. The single Tesla C2050

CUDA benchmark	Host	gVirtuS
AT	30.8	29.3
BT (h2d)	2.8	0.29
BT (d2h)	2.4	0.2
BT (d2d)	84.2	82.1
MC	5.44	0.14

TABLE VII
MEMORY THROUGHPUT IN GB/S FOR HOST (NON-VIRTUALIZED) AND GVIRTUS USING CUDA 3.2.

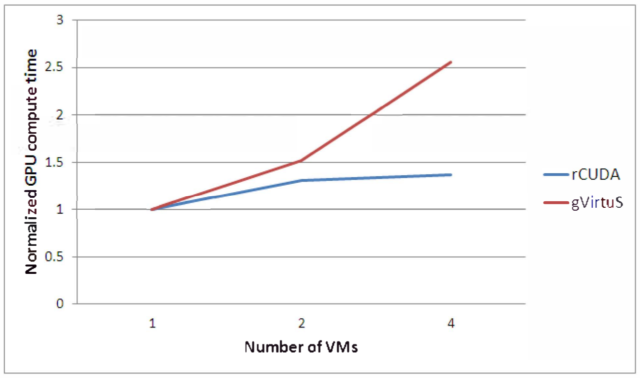

Fig. 4. Multiplexing results for rCUDA and gVirtuS.

GPU was multiplexed among the different VMs. Figure 4 plots the normalized GPU compute time as we increase the number of concurrent VMs. Please note that the GPU compute time is the maximum time taken to execute the concurrent instances. This time is normalized to the time taken on a single VM. Interestingly, we notice that rCUDA scales much better than gVirtuS. We need to investigate this further to understand the reasons for this behavior.

4) Interposition: Interposition, as mentioned earlier, refers to the ability of the virtualization software to mediate access between a virtual machine and the physical hardware. This extra level of abstraction enables support for advanced features such as live migration of virtual machines, check pointing and re-start etc. None of the virtualization solutions discussed provides this feature. A virtualization solution that provides for complete interposition may suffer in terms of performance.

V. Summary and Conclusions

This paper presents a detailed study on three virtualization solutions: rCUDA, gVirtuS and Xen PCI pass-through, that support CUDA acceleration within virtual machines. It compares and contrasts the solutions based on their fidelity, performance, multiplexing and interposition characteristics. Xen PCI pass-through, supports all the GPU features in the VMs as in the host. Thus, it provides for maximum fidelity and performance in comparison to the other two solutions. However, as a GPU device is dedicated to a VM in the case of Xen-PCI, we could say that it does not support GPU virtualization in its truest sense. rCUDA and gVirtuS, which are front-end virtualization frameworks based on API remoting, support multiplexing of the GPU device among the VMs. rCUDA displays greater fidelity in comparison to gVirtuS, but shows lower performance. However, a next enhanced version of rCUDA with advanced features including pipelined data transfers is expected shortly. None of these solutions support interposition.

Based on our study, we opine that true CUDA-enabled GPU virtualization that provides for highest fidelity, performance, multiplexing and interposition, would require virtualization support in the GPU hardware. The NVIDIA VGX technology that is expected to support full GPU virtualization at the hardware, once released, may mark a disruptive break-through innovation in this domain.

Acknowledgements: We would like to thank rCUDA and gVirtuS development team for providing access to their respective softwares as well as for the support in installation and troubleshooting. We would also like to thank all the people replied for our queries in gvirtus and xen mailing lists.

References

[1] M. F. Mergen, V. Uhlig, O. Krieger, and J. Xenidis, “Virtualization for high-performance computing,” *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 2, pp. 8–11, Apr. 2006.
[2] “Trends in enterprise virtualization technologies,” http://www.f5.com/pdf/reports/enterprise-virtualization.pdf.
[3] B. H. Ng, B. Lau, and A. Prakash, “Direct access to graphics card leveraging vt-d,” University of Michigan - Computer Science and Engineering Division, Tech. Rep., July 2009.
[4] “Nvidia vgx technology,” http://www.nvidia.com/object/vdi-desktop-virtualization.html.
[5] M. Dowty and J. Sugerman, “Gpu virtualization on vmware’s hosted i/o architecture,” *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 3, pp. 73–82, Jul. 2009.
[6] L. Shi, H. Chen, J. Sun, and K. Li, “vcuda: Gpu-accelerated high-performance computing in virtual machines,” *IEEE Transactions on Computers*, vol. 61, pp. 804–816, 2012.
[7] J. Duato, A. J. Peña, F. Silla, J. C. Fernández, R. Mayo, and E. S. Quintana-Ortí, “Enabling cuda acceleration within virtual machines using rcuda,” in *International Conference on High Performance Computing (HiPC)*, 2011, pp. 1–10.
[8] J. Duato, A. J. Pena, F. Silla, R. Mayo, and E. S. Quintana-Orti, “Performance of cuda virtualized remote gpus in high performance clusters,” in *Proceedings of the 2011 International Conference on Parallel Processing*, ser. ICPP ’11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 365–374.
[9] G. Giunta, R. Montella, G. Agrillo, and G. Coviello, “A gpgpu transparent virtualization component for high performance computing clouds,” in *Proceedings of the 16th international Euro-Par conference on Parallel processing: Part I*, ser. EuroPar’10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 379–391.
[10] V. Gupta, A. Gavrilovska, K. Schwan, H. Kharche, N. Tolia, V. Talwar, and P. Ranganathan, “Gvim: Gpu-accelerated virtual machines,” in *Proceedings of the 3rd ACM Workshop on System-level Virtualization for High Performance Computing*, ser. HPCVirt ’09. New York, NY, USA: ACM, 2009, pp. 17–24.
[11] “Citrix xenserver 6,” http://www.citrix.com/english/ps2/products/feature.asp?contentid=2316933.
[12] “Xen pci pass-through,” http://wiki.xen.org/wiki/Xen_PCI_Passthrough.
[13] “Vmware vroom! blog: Gpgpu computing in a vm,” http://blogs.vmware.com/performance/2011/10/gpgpu-computing-in-a-vm.html.
[14] “Intel virtualization technology,” http://www.intel.com/technology/itj/2006/v10i3/2-io/1-abstract.htm.
[15] “Vmware vmdirectpath i/o,” http://communities.vmware.com/docs/DOC-11089.
[16] “Cluster gpu instances for amazon ec2,” http://aws.amazon.com/about-aws/whats-new/2010/11/15/announcing-cluster-gpu-instances-for-amazon-ec2/.
[17] “Peer1 hosting gpu cloud,” http://www.peer1.com/cloud-hosting/high-performance-cloud-computing.
[18] “Zillians v-gpu,” http://www.zillians.com/how-it-works-2/.
[19] U. o. C. Computer Laboratory, “The xen virtual machine monitor,” http://www.cl.cam.ac.uk/research/srg/netos/xen/, 2008.
[20] “Xen pvhvm drivers for linux hvm guests,” http://wiki.xen.org/xenwiki/XenLinuxPVonHVMdrivers.
[21] “rcuda user guide version 3.1.2,” http://www.rcuda.net/pub/rCUDA_guide.pdf.
[22] “Xen hypervisor roadmap and archives,” http://xen.org/products/xen_archives.html.
[23] “git.kernel.org,” http://git.kernel.org/?p=linux/kernel/git/jeremy/xen.git;a=commit;h=2b494f184d3337d10d59226e3632af56ea66629a.