Using PCI Pass-Through for GPU Virtualization with CUDA

Chao-Tung Yang, Hsien-Yi Wang, Yu-Tso Liu

Using PCI Pass-Through for GPU Virtualization with CUDA[1]

Chao-Tung Yang[2] Hsien-Yi Wang Yu-Tso Liu

Department of Computer Science, Tunghai University, Taichung 40704, Taiwan ROC

Email: ctyang@thu.edu.tw, wiath.wang@gmail.com, vvvv6502@hotmail.com

Abstract. Nowadays, NVIDIA's CUDA is a general purpose scalable parallel programming model for writing highly parallel applications. It provides several key abstractions – a hierarchy of thread blocks, shared memory, and barrier synchronization. This model has proven quite successful at programming multi-threaded many core GPUs and scales transparently to hundreds of cores: scientists throughout industry and academia are already using CUDA to achieve dramatic speedups on production and research codes. GPU-base clusters are likely to play an important role in future cloud data centers, because some compute-intensive applications may require both CPUs and GPUs. The goal of this paper is to develop a VM execution mechanism that could run these applications inside VMs and allow them to effectively leverage GPUs in such a way that different VMs can share GPUs without interfering with one another.

Keywords: CUDA; GPU virtualization; Cloud computing; IaaS; PCI pass-through

1 Introduction

GPUs are really "manycore" processors, with hundreds of processing elements. A graphics processing unit (GPU) is a specialized microprocessor that offloads and accelerates graphics rendering from the central (micro-) processor. Modern GPUs are very efficient at manipulating computer graphics, and their highly parallel structure makes them more effective than general-purpose CPUs for a range of complex algorithms. We know that a CPU has only 8 cores at single chip currently, but a GPU has grown to 448 cores. From the number of cores, we know that GPU is appropriately to compute the programs which are suited massive parallel processing. Although the frequency of the core on the GPU is lower than CPU's, we believe that massively parallel can conquer the problem of lower frequency. By the way, GPU has been used

[1] This study was supported in part by the National Science Council, Taiwan ROC, under grant numbers NSC 100-2218-E-029-004 and NSC 100-2622-E-029-008-CC3.

[2] Corresponding author

on supercomputers. In top 500 sites for November 2010, there are three supercomputers of the first 5 are built with NVIDIA GPU [2].

In recent years, virtualization environment on Cloud [1] become more and more popular. The balance between performance and cost is the most important point that everybody focused. For more effective to use the resource on the server, virtualization technology is the solution. Running many virtual machines on a server, the resource can be more effective to use. But the performance of virtual machines has their own limitations. Users will limit by using a lot of computing on virtual machine.

Proper use of hardware resources and computing power to each virtual machine is the Infrastructure as a Service (IaaS), which is one of the architecture of Cloud Computing. But the virtual machine has its limitations, system virtual environment does not support CUDA [4], CUDA high performance computing unable to GPU virtualization. In this paper, by using PCI pass-through technology, making the virtual machines in a virtual environment able to use the NVIDIA graphics card, and then we can use the CUDA high performance computing. Finally, we will compare the two open source virtualization environment hypervisor, whether it will be after PCI pass-through CUDA performance differences. Through the experiment, we will be able to know which environment will be the best in a virtual environment using CUDA.

Therefore, there is a new topic that let the virtual machines using the physical GPGPU (General-Purpose computing on Graphics Processing Units) in the real machine to help compute. Because GPU is really many-core processors, the computing power of virtual machines will increase by using GPU. In this paper, we introduce some of the hypervisor environment for virtualization and different virtualization types on cloud system. And we also introduce some types of hardware virtualization. This paper implements a system with virtualization environment and using PCI pass-through technology let the virtual machines on the system can use GPU accelerator to increase the computing power. Final, we discuss the performance of GPU between virtual machines and native machine.

2 Background

Virtualization [5] technology is a technology that creation of a virtual version of something, such as a hardware platform, operation system, a storage device or network resources. The goal of virtualization is to centralize administrative tasks while improving scalability and overall hardware-resource utilization. By using virtualization, several operating systems can be run on a single powerful server without glitch in parallel.

2.1 Full-Virtualization

Unlike the traditional way that put the operation system kernel to Ring 0 level, full-virtualization use hypervisor instead of that. Hypervisor manage all instructions to Ring 0 from Guest OS. Full-virtualization [6] uses the Binary Translation technology to translate all instructions to Ring 0 from Guest OS and then send the requirement to hardware. Hypervisor virtualized all hardware until, Guest OS access the hardware just like a real machine. It has highly independence. But Binary Translation technology reduces the performance of virtual machine. The architecture of full-virtualization is shown in Figure 1.

2.2 Para-Virtualization

In para-virtualization [7], does not virtualize all hardware until. There is a unique Host OS called Domain0. Domain0 is parallel with other Guest OS and use Native Operating System to manage hardware driver. Guest OS accessing the real hardware by calling the driver in Domain0 through hypervisor. The requirement sent by Guest OS to hypervisor is called Hypercall. To make the Guest OS sending the hypercall instead of sending requirement directly to hardware, the Guest OS's kernel needs to rewrite, so that some non-open-sourced operation systems cannot support.

In para-virtualization, does not virtualize all hardware until. There is a unique Host OS called Domain0. Domain0 is parallel with other Guest OS and use Native Operating System to manage hardware driver. Guest OS accessing the real hardware by calling the driver in Domain0 through hypervisor. The requirement sent by Guest OS to hypervisor is called Hypercall. To make the Guest OS sending the hypercall instead of sending requirement directly to hardware, the Guest OS's kernel needs to rewrite, so that some non-open-sourced operation systems cannot support. The architecture of para-virtualization is shown in Figure 2.

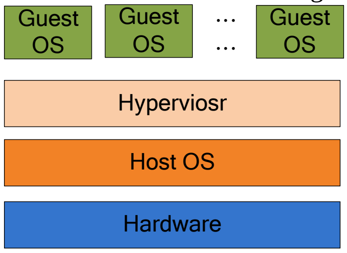


Fig. 1. Full-Virtualization

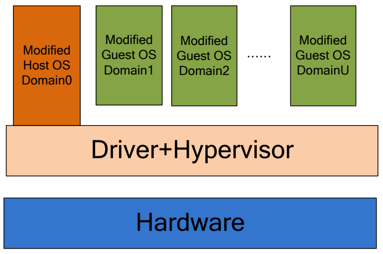

Modified Host OS Domain0
Modified Guest OS Domain1
Modified Guest OS Domain2
……
Modified Guest OS DomainU
Driver+Hypervisor
Hardware

Fig. 2. Para-Virtualization

2.3 Xen

The VM Layer of Host OS type deploys on Host OS, such like Windows or Linux, and then installs other operation system on top of VM Layer. The operation systems on top the VM Layer are called Guest OS. Xen's hypervisor is installed directly in the host, and the other operation systems we want deploy are on top of it. It is easier to manage CPU, Memory, Network, Storage and other resource. The main purpose of Xen [3] uses hypervisor type and its VMM (Virtual Machine Monitor) is more efficient and safety to control the host CPU, Memory and other resource.

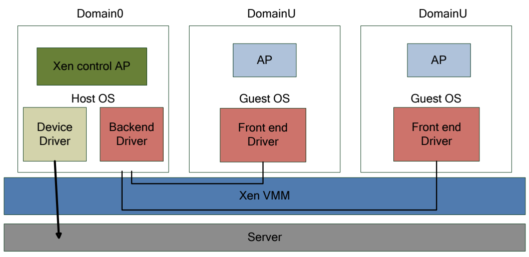
Xen uses a unit called Domain to manage virtual machines. Domain is divided into two types as shown in Figure 3, one of them is called Domain0, played like Host OS, has control AP of Xen, used for management. Another type called DomainU is a field that Guest OS installed on it. When using physical resource, DomainU cannot call the hardware driver directly, it must be through Domain0 to deal with.

2.4 Related Work

There are some approaches that pursue the virtualization of the CUDA Runtime API for VMs such as rCUDA [12], vCUDA, GViM [13] and gVirtuS [14]. The solutions feature a distributed middleware comprised of two parts, the front-end and the back-end. Figure 4 shows that the front-end middleware is installed in the virtual machine, and the back-end middleware with direct access to the acceleration hardware, is running by host OS with executing the VMM.

rCUDA using Sockets API to let the client and server have communication with each other. And client can use the GPU on server through that. It is a production-ready framework to run CUDA applications from VMs, based in a recent CUDA API version. We can use this middleware to make a customized communications protocol and is independent.

VMGL [16] is the OpenGL hardware 3D acceleration for virtual machines, OpenGL apps can run inside a virtual machine through VMGL. VMGL can be used on VMware guests, Xen HVM domains (depending on hardware virtualization extensions) and Xen paravirtual domains, using XVnc or the virtual frame buffer. VMGL is available for X11-based guest OS's: Linux, FreeBSD and OpenSolaris. Finally, VMGL is GPU-independent: we support ATI, nVidia and Intel GPUs.



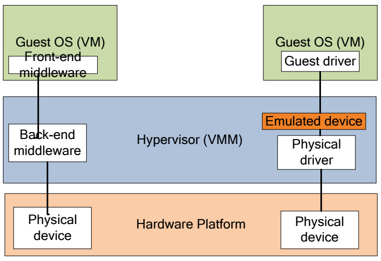


Fig. 3. Domain0 and DomainU

Fig. 4. Front-End and Back-End

The key idea in vCUDA is: API call interception and redirection. With API interception and redirection, applications in VMs can access graphics hardware device and achieve high performance computing applications. It allows the application executing within virtual machines to leverage hardware acceleration. They explained how to access graphics hardware in VMs transparently by API call interception and redirection. Their evaluation showed that GPU acceleration for HPC applications in VMs is feasible and competitive with those running in a native, non-virtualized environment [8].

GViM is a system designed for virtualization and managing the resources of a general purpose system accelerated by graphics processors. GViM uses Xen-specific mechanisms for the communication between front-end and back-end middleware. The GViM virtualization infrastructure for a GPGPU platform enables the sharing and consolidation of graphics processors. Their experimental measurements of a Xen-based GViM implementation on a multi-core platform with multiple attached NVIDIA graphics accelerators demonstrate small performance penalties for virtualized vs. non-virtualized settings, coupled with substantial improvements concerning fairness in accelerator use by multiple VMs.

In J. Duato's work, he uses remote GPU for virtual machine. Although his virtualization technique noticeably increases execution time when using a 1 Gbps Ethernet network, it performs almost as efficiently as a local GPU when higher performance interconnects are used. Therefore, the small overhead incurred by our proposal because of remote use of GPUs is worth the savings that a cluster configuration with less GPUs than nodes reports [11].

Atsushi Kawai and Kenji Yasuoka proposed DS-CUDA, a middleware to virtualize a GPU cluster as a distributed shared GPU system. It simplifies development of a code that uses multiple GPUs distributed on a network. Results with good scalability were shown in their paper. Also the usefulness of the redundant calculation mechanism is confirmed.

3 System Implementation

To use GPU accelerator on virtual machines, this paper plans using PCI pass-through to implement the system for better performance. For performance, near-native performance can be achieved using device pass-through. This technology is perfect for networking applications or those that have high disk I/O or like using hardware accelerator that have not adopted virtualization because of contention and performance degradation through the hypervisor. But assigning devices to specific guests is also useful when those devices cannot be shared. For example, if a system included multiple video adapters, those adapters could be passed through to unique guest domains.

VT-d Pass-Through is a technique to give a DomU exclusive access to a PCI function using the IOMMU[17] provided by VT-d. It is primarily targeted at HVM (fully virtualized) guests because Para-Virtualized pass-through does not require VT-d .There is an important thing that your hardware must support that. In addition to the motherboard chipset and BIOS also your CPU must have support for IOMMU IO virtualization (VT-d). VT-d is disabled by default, to enable it, need "iommu" parameter to enable it. The system architecture and user point of view are shown in Figures 5 and 6, respectively.

4 Experimental Results

Previously, we have conducted the design principle and implementation methods. We present here several experiment conducts on two machines. The node's hardware specification is listed in Table 1. In Table 1, we use a machine with the Xeon E5506, 12GB memory, CentOS 6.2 with Xen. Quadro NVS 295 is using for primary graphics card. Tesla C1060 is the one we using for computing and passing through to virtual machine.

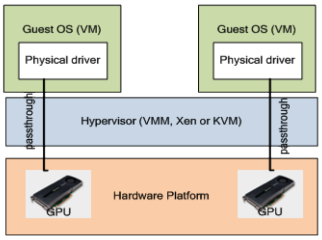


Fig. 5. System architecture

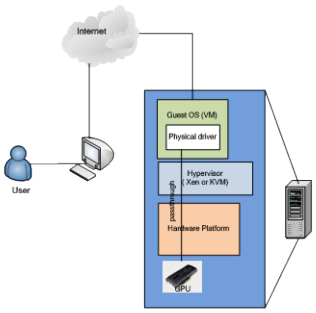


Fig. 6. User point of view

Table 2 is the hardware/software specification of virtual machines. We want discuss that the CPU number will or not affect the performance of virtual machine in PCI pass-through. So we will use 1, 2 or 4 CPUs in our virtual machine to see the difference between each other. These of virtual machines' virtualization type are full, because we found out that PCI pass-through is not working in para-virtualization in our research.

Table 1. Hardware/Software Specification

Hardware/Software Specification						
	CPU	Memory	Disk	OS	Hypervisor	GPU
Node	Xeon E5506	12GB	1TB	CentOS 6.2	Xen	Quadro NVS 295/ Tesla C1060

Table 2. Hardware/Software Specification of Virtual Machines

Hardware/Software Specification of Virtual Machine							
	CPU	Memory	Disk	OS	Hypervisor	GPU	Virtualization
VM1	1	1024MB	12GB	CentOS 6.2	Xen	Tesla C1060	Full
VM2	2	1024MB	12GB	CentOS 6.2	Xen	Tesla C1060	Full
VM3	4	1024MB	12GB	CentOS 6.2	Xen	Tesla C1060	Full

We use seven benchmarks for performance comparison: alignedTypes, asyncAPI, BlackScholes, clock, convolutionSeparable, fastWalshTransform and matrixMul. These seven benchmarks are part of CUDA SDK. From many benchmarks in the suite, we select 7 representative SDK benchmarks of varying computation loads and data size which use different CUDA feature. These benchmarks are executed with the default options. And we also test the bandwidth between GPU and virtual machine. We want to see that the PCI pass-through technology will affect the bandwidth or not.

The first experiment is performance comparison between native and virtual machine. We will present how much GPU performance will be reduced with PCI pass-through. The second experimental result is performance comparison among three virtual machines with one CPU, two CPUs and four CPUs, respectively. We will demonstrate the number of CPU used in virtual machines will affect the GPU performance or not. All SDK benchmarks' execution time is measured by the Linux command 'time' in the CentOS 6.2.

We first analyze the performance of the CUDA SDK benchmarks running in a VM using PCI pass-through, and compare their execution times with those of a native environment —i.e., using the regular CUDA Runtime library in a non-virtualized environment. Figure 10 shows that the execution time on processing the SDK benchmark in native and virtual machine using one CPU on Xen. We can see the real time of these benchmarks on virtual machine is less than native machine.

Figure 11 shows that the different execution time between virtual machines which one has one CPU, another has two CPUs. In this figure, we can see that the number of CPUs does not affect the user time, which means the GPU computing time is not changed when the number of CPU changed one to two. Figure 12 shows that the execution time between two CPUs and four CPUs in virtual machines which based on Xen. In this figure, we can see more clearly about the number of CPU does not affect the performance of GPU. In Figures 10, 11 and 12, we can demonstrate that the execution time is very close. Only the system time is significantly different.

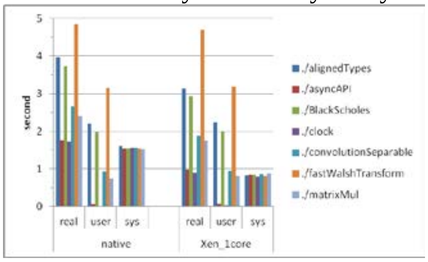


Fig. 7. Execution Time between Native and Virtual Machine

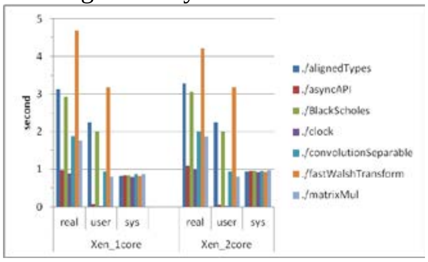


Fig. 8. Execution Time between 1 Core and 2 Core Virtual Machine

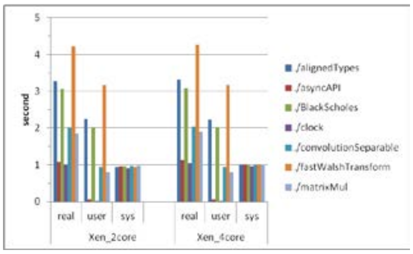


Fig. 9. Execution Time between 2-Core and 4-Core Virtual Machines

5 Conclusions

As we know, the method of CUDA is famous in parallel programming. The performance could get better after the serial codes translated to the codes which using this method. Virtualization is also a famous thing in recent year. Through the virtualization technology, we can use the resource on the serve more efficient. Using the PCI pass-through to implement that computing with GPU accelerator in virtual machines is very helpful. In our work, we can see the performance of GPU is the same between native and virtual machine. No matter how many CPU in virtual machine, the GPU provide the same performance by PCI pass-through. Even we using virtual machine, the system time are less than real machine. And the system time of the virtual machine using one CPU is less than the four. The inner communication in virtual machine is not through the real hardware but just in the memory on the real machine. Using the PCI pass-through to implement that computing with GPU accelerator in virtual machines is saving more resource, and has the same performance with real machine in computing overall.

References

1. Cloud computing, http://en.wikipedia.org/wiki/Cloud_computing
2. NVIDIA, http://www.nvidia.com.tw
3. Xen, http://www.xen.org/
4. CUDA, http://en.wikipedia.org/wiki/CUDA
5. Virtualization, http://en.wikipedia.org/wiki/Virtualization
6. Full Virtualization, http://en.wikipedia.org/wiki/Full_virtualization
7. Para Virtualization, http://en.wikipedia.org/wiki/Paravirtualization
8. L. Shi, H. Chen, and J. Sun, "vCUDA: GPU accelerated high performance computing in virtual machines," in IEEE International Symposium on Parallel & Distributed Processing (IPDPS'09), 2009
9. J. Duato, F. D. Igual, R. Mayo, A. J. Pena, E. S. Quintana-Ort, and F. Silla, "An efficient implementation of GPU virtualization in high performance clusters," in Euro-Par 2009, Parallel Processing — Workshops, ser. LNCS, vol. 6043. Springer-Verlag, 2010, pp. 385–394
10. J. Duato, A. J. Pena, F. Silla, R. Mayo, and E. S. Quintana-Ort, "rCUDA: Reducing the number of GPU based accelerators in high performance clusters," in Proceedings of the 2010 International Conference on High Performance Computing & Simulation (HPCS 2010), Jun. 2010, pp. 224–231
11. J. Duato, A. J. Pe˜na, F. Silla, R. Mayo, and E. S. Quintana-Orti, "Performance of CUDA virtualized remote GPUs in high performance clusters," in International Conference on Parallel Processing (ICPP), Sep. 2011
12. D. Inf. de Sist. y Comput., Univ. Politec, "Enabling CUDA acceleration within virtual machines using rCUDA," High Performance Computing (HiPC), 2011 18th International Conference , Dec 2011
13. V. Gupta, A. Gavrilovska, K. Schwan, H. Kharche, N. Tolia, V. Talwar, and P. Ranganathan, "GViM: GPU-accelerated virtual machines," in 3rd Workshop on System-level Virtualization for High Performance Computing. NY, USA: ACM, 2009, pp. 17–24
14. G. Giunta, R. Montella, G. Agrillo, and G. Coviello, "A GPGPU transparent virtualization component for high performance computing clouds," in Euro-Par 2010 - Parallel Processing, ser. LNCS, P. D Ambra, M. Guarracino. D. Talia, Eds. Springer Berlin / Heidelberg, 2010, vol. 6271, pp. 379–391
15. C.T. Yang, C.L. Huang and C.F. Lin, "Hybrid CUDA, OpenMP, and MPI Parallel Programming on Multicore GPU Clusters", Computer Physics Communications, Vol. 182, Issue 1, pp. 266-269, June 25, 2010
16. VMGL, http://sysweb.cs.toronto.edu/vmgl
17. Nadav Amit, Muli Ben-Yehuda and Ben-Ami Yassour, "IOMMU: Strategies for Mitigating the IOTLB Bottleneck," Lecture Notes in Computer Science, 2012, Volume 6161, Computer Architecture, Pages 256-274