On GPU Pass-Through Performance for Cloud Gaming: Experiments and Analysis

Ryan Shea
Simon Fraser University
Burnaby, Canada
Email: rws1@cs.sfu.ca

Jiangchuan Liu
Simon Fraser University
Burnaby, Canada
Email: jcliu@cs.sfu.ca

***Abstract*—Cloud Gaming renders interactive gaming applications remotely in the cloud and streams the scenes back to the local console over the Internet. Virtualization plays a key role in modern cloud computing platforms, allowing multiple users and applications to share a physical machine while maintaining isolation and performance guarantees. Yet the Graphical Processing Unit (GPU), which advanced game engines heavily rely upon, is known to be difficult to virtualize. Recent advances have enabled virtual machines to directly access physical GPUs and exploit their hardware's acceleration.**

This paper presents a experimental study on the performance of real world gaming applications as well as ray-tracing applications with GPUs. Despite the fact that the VMs are accelerated with dedicated physical GPUs, we find that the gaming applications perform poorly when virtualized, as compared to non-virtualized bare-metal base-line. For example, experiments with the Unigine gaming benchmark run at 85 FPS on our bare-metal hardware, however, when the same benchmark is run within a Xen or KVM based virtual machine the performance drops to less than 51 FPS. In contrast, ray-tracing application fares much better. Our detailed performance analysis using hardware profiling on KVM further reveals the memory bottleneck in the pass through access, particularly for real-time gaming applications.

I. Introduction

Fueled by reduced costs and unparalleled scalability, cloud computing is drastically changing the existing operation and business models of the IT industry. Specifically, it has turned the idea of *Cloud Gaming* into a reality. Cloud Gaming, in its simplest form, renders an interactive gaming application remotely in the cloud and streams the scenes as a video sequence back to the player over the Internet. This is an advantage for less powerful computational devices that are otherwise incapable of running high quality games.

Virtualization plays a key role in modern cloud computing platforms, allowing multiple users and applications to share a physical machine while maintaining isolation and performance guarantees. Today, virtualization has been well implemented for most of the computer resources, in particular, the general-purpose CPU for computation. Offloading the core tasks from the local console, cloud gaming clearly demands intensive computation in the cloud too, particularly for such high-complex operations as 3D rendering. Yet state-of-the-art game engines not only rely on the general purpose CPU for computation, but, more heavily, on the Graphical Processing Unit (GPU). A GPU is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display. In applications requiring massive vector operations, the computational power of the highly parallel pipelines in a GPU can yield several orders of magnitude higher performance than a conventional CPU. This makes many advanced gaming with ultra-realistic scenes possible. Beyond graphics, GPUs have been harnessed to accelerate such other game elements as physics engines in today's modern gaming systems, and general purpose GPUs (GPGPUs) are now empowering a wide range of matrix- or vector-based computation tasks for scientific or other applications.

While GPUs cards have been virtualized to some degree in modern virtualization systems, their performance has generally been poor [1]. This is because the high-memory bandwidth demands and difficultly in multi-tasking make a virtualized GPU a poor performer when multiple VMs attempt to share a single GPU. However, recent advances in terms of both hardware design and virtualization software design have allowed virtual machines to directly access physical GPUs and exploit the hardware's acceleration features. This access is provided using hardware, and grant a single VM a one-to-one hardware mapping between itself and the GPU. These advances have allowed the cloud platforms to offer virtual machine instances with GPU capabilities. For example, Amazon EC2 has added a new instance class know as "GPU instances", which have dedicated Nvidia Tesla GPU for GPU computing.

This paper presents a experimental study on the performance of real world gaming applications with GPUs as well as ray-tracing applications with GPGPUs. We measure the performance of these applications running in virtualized environments as well as our non-virtualized bare-metal baseline. Despite the fact that our VMs are accelerated with direct access to a dedicated GPU, we find that the gaming applications perform very poorly when run inside the VM. For example, Doom 3 by ID software achieved a frame rate of approximately 40 frames per second (FPS) when virtualized in both KVM and Xen. In contrast, our bare-metal baseline achieved over 120 FPS, when run on the same hardware. Our experiment with the Unigine gaming engine shows similar results falling from nearly 85 FPS when run on bare-metal to less then 51 when run in a virtualized system. Despite the poor performance of the virtualized gaming applications, we find that the GPGPU ray-tracing application fares much better when virtualized, achieving near identical performance

to our base-line. However, when virtualized, the ray-tracing application does consume far more systems resources such as processor cycles and cache. To help uncover the root cause of the performance degradation in these virtualized gaming systems, we perform a detailed performance analysis using hardware profiling on KVM. Our analysis reveals the memory bottleneck in the pass-through access, particularly for real-time gaming applications.

II. Background and Related Work

We first offer an overview of different classes of virtualization technologies used in today's cloud computing environment, followed by related works on device pass through, in particular, GPU pass through.

A. Virtualization Solutions

Computer virtualization can be roughly separated into two major classes, namely, Paravirtualization and Hardware Virtualization.

Paravirtualization(PVM) is one of the first adopted versions of virtualization and is still widely deployed today. It requires no special hardware to realize virtualization, instead relying on special kernels and drivers. The kernel sends privileged system calls and hardware access directly to a *hypervisor*, which in turn decides what to do with the request. The use of special kernels and drivers means a loss of flexibility in terms of choosing the operating systems. In particular, PVM must use an OS that can be modified to work with the hypervisor. Typical PVM solutions include Xen and User Mode Linux. Amazon [2], the current industry leader in cloud computing, heavily uses Xen [3] to power its EC2 platform.

Hardware Virtual Machine (HVM) is the lowest level of virtualization, requiring special hardware capabilities to trap privileged calls from guest domains. It allows a machine to be fully virtualized without the need for any special operating systems or drivers on the guest system. Most modern CPUs are built with HVM capabilities, often called *virtualization extensions*. They detect if a guest VM tries to make a privileged call to a system resource. The hardware intercepts this call and sends it to a hypervisor which decides how to handle the call. It has been noticed however that HVMs can also have the highest virtualization overhead and as such may not always be the best choice for a particular situation[4]. Yet paravirtualization I/O drivers can alleviate such overhead; one example of a paravirtualization driver package is the open source VirtIO [5]. Representative HVM solutions include VMware Server, KVM, and Virtual-Box.

B. GPU Virtualization and Pass Through

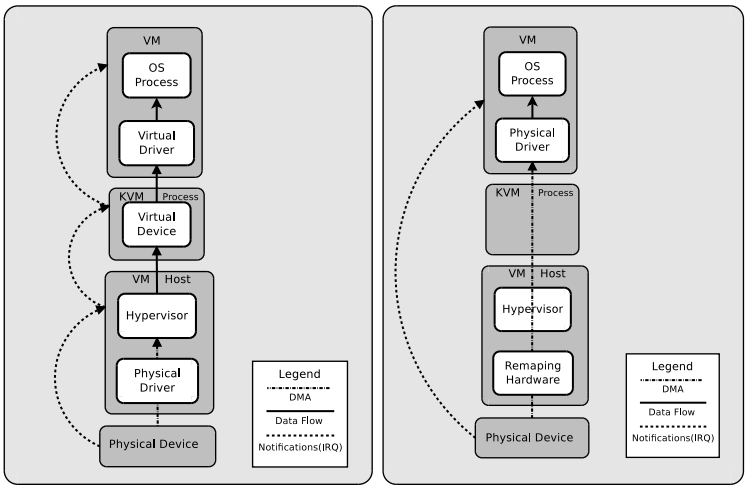
Using PVM and HVM, most of the computer resources have been well virtualized. There have also been initial attempts to virtualize GPU in VMWare [1]. Yet, full virtualization of GPU remains difficult. This difficulty is due to the fact that GPU accelerated applications require fast access and high transfer rates between the GPU and main memory, something which is difficult to implement in software. Further, unlike the CPU the GPU is notoriously bad at multi-tasking with multiple GPU intensive applications, even when run on a bare-metal system. Thus, multiple GPU intensive VMs will also have very low performance if they attempt to use the same shared GPU.

Recent hardware advances have allowed virtualization systems to do a one-to-one mapping between a device and a virtual machine guest. This technology is designed to allow hardware devices that do not virtualize well to still be used by a VM. Examples of devices commonly passed in include the Network interface card (NIC) and the GPU. Both Intel and AMD have created hardware extensions, which allow this device pass through. Both Intel's implementation, named VT-D, and AMD's implementation, named AMD-Vi, work by making the processors input/output memory management unit (IOMMU) configurable by the systems hypervisor. The new extensions allow the hypervisor to reconfigure the interrupts and direct memory access (DMA) channels of a physical device so they map directly into one of the guests. In Figure 1a we show the simplified architecture enabling a Hypervisor to access a shared device to a VM. As can be seen data flows through DMA channels from the Physical device into the memory space of the VM host. The hypervisor then forwards the data to a virtual device belonging to the guest VM. The virtual devices interacts with the driver residing in the VM to deliver the data to the guest virtual memory space. Notifications are sent via interupts and follow a similar path. Figure 1b shows how a 1-1 device pass-through to a VM is achieved. As can be seen the DMA channel can now flow data directly from the physical device to the VMs memory space. Also, interrupts can be directly mapped into the VM. This feature is made possibly through the use of remapping hardware, which the hypervisor configures for the guest VM.

The performance implications and overhead of pass-through have been examined for various pass-through devices, e.g, network interface controllers [6][7][8]. The potential bottleneck for pass-through devices created by the input/output memory management unit (IOMMU) TLB was closely analyzed in [9]. There have also been recent studies on enabling multiple VMs to access CUDA enabled GPUs [10][11], as well as performance analysis of CUDA applications using a GPU pass through device in Xen [12]. Also, there has been much research on latency and QoE issues in cloud gaming systems [13][14][15]. Despite these pioneering efforts, the suitability of GPU pass-through devices for cloud gaming has seldom been explored.

III. Virtualization and Application Setup

Our test system was a modern mid-range server with an AMD Phenom II 1045t six core processor running at 2.7 Ghz. We enabled AMD-V and AMD-Vi in the BIOS as they are required for Hardware Virtual Machines (HVM) support and device pass-through. The PC was equipped with 16 GB of 1333 MHz DDR-3 SDRAM and a 500 GB 7200 RPM hard drive with 16MB cache. The physical network interface is a 1000 Mb/s Broadcom Ethernet adapter attached to the PCI-E bus. The host and Virtual Machine guests used Debian Wheezy as their operating system. Our GPU is an ATI based Sapphire HD 5830 Xtreme with 1 GB of DDR5 memory.



(a) Shared Device (b) Pass-through Device

Fig. 1: Shared vs Pass-through Device Simplified Architecture

We test 3 systems: our bare metal Linux host, KVM with its optimal paravirtualized drivers, and Xen running in HVM mode. HVM mode is required in Xen since the GPU pass-through requires hardware virtualization extensions. Next, we give detailed configuration details on our system.

A. Xen Setup

We installed the Xen 4.0 Paravirtualization Hypervisor on our test system, following closely the Debian guide. To configure networking we created a bridged adapter and attached our primary interface and Xen's virtual interfaces to it. Xen virtual machines received an IP address from the DHCP running on our gateway. For disk interface, we used a flat file attached to Xen's optimized paravirtualization disk interface. To install a base system into the image, we used the utility `xen-tools`, which automates the install procedure. We set the number of virtual CPUs (VCPU) to 6 and the amount of RAM to 8048 MB. The virtual machine host ran the 3.2.0-4-Xen kernel. We pass in the GPU using linux PCI-Stub driver and Xen's PCI-Back pass-through module.

B. KVM System Setup

We downloaded KVM 1.2.0 from the official Debian repository and installed it on our test system. Once again the virtual machine was given access to all 6 processor cores as well as 8048 MB of memory. The disk interface was configured as a flat file on the physical host's file system. Networking was configured once again as a bridge between the virtual machine's interface and the system's physical NIC. To enable best performance, we configured KVM to use the VirtIO drivers [5]. Debian kernel 3.2.0-4-amd64 was used in the virtual machine to stay consistent with the other tests. Once again we use the Linux PCI-Stub driver to pass the physical GPU into our virtual machine.

1) Non-Virtualized 'Bare metal' System Setup: Finally, as the baseline for comparison, we had a Bare-metal setup with no virtualization running, i.e., the system has direct access to the hardware. The same drivers, packages and kernel were used as in the previous setup. This configuration enabled us to calculate the minimal amount of performance degradation that our system can experience.

C. Gaming Applications and Benchmark Setup

We chose three different complex applications to test our GPU performance. We selected two game engines, both of which have cross platform implementation, and run natively on our Debian Linux machine. The first is *Doom 3*, which is a popular game released in 2005, and Utilizes OpenGL to provide high quality graphics. Next, we chose to test the *Unigine Sanctuary* benchmark, which is an advanced benchmarking tool that runs on both Windows and Linux. The Unigine engine uses state-of-the-art OpenGL hardware features to provide rich graphics that are critical to many state-of-the-art games. Finally, we use the *LuxMark* ray-tracing benchmark, which is to test applications written in OpenCl to see if there were any performance difference between games and GPGPU applications such as ray tracing.

IV. Experimental Results with GPU Pass Through

For each of the following experiments, we run each benchmark 3 times and graph the average. For Doom3 and Sanctuary we give the results in frames per second (FPS) and for Luxmark we provide the results in terms of number of rays and samples calculated per second. We also provide amount of CPU consumed by each system, which we measure through the Linux top command for KVM and the bare-metal system. For Xen we use the equivalent Xen Top command. Top reports data in terms of percentage of CPU used. For example, 80% means the equivalent to 0.8 cores is being consumed by the system and 200% means 2 cores are currently being occupied.

A. Doom 3 Results

In Figure 2 we describe the results of the time demo Doom 3 benchmark, measured in Frames Per Second (FPS),

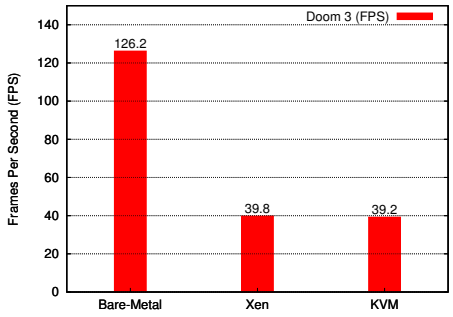


Fig. 2: Doom 3 Performance

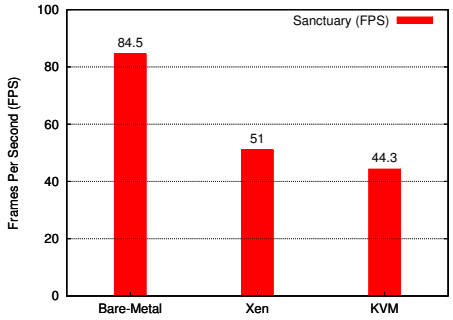


Fig. 3: Unigine Sanctuary Performance

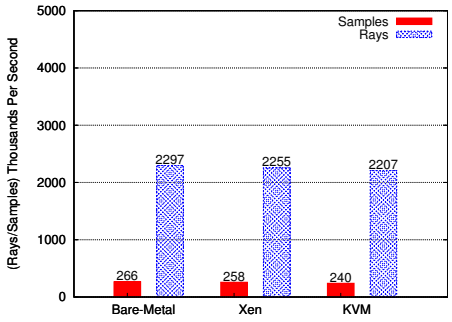


Fig. 4: LuxMark Ray Tracing Benchmark

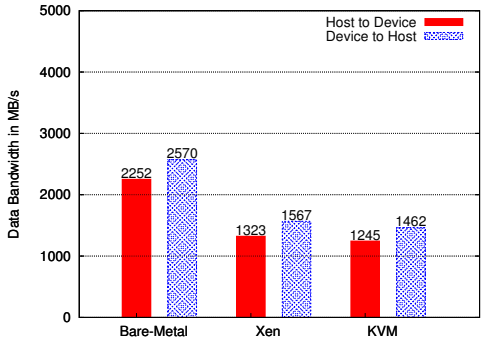


Fig. 5: Memory Bandwidth by System

running on our Bare metal system as well as on the virtualized systems Xen and KVM. The Bare metal system performs at 126.2 FPS, while Xen and KVM perform at 39.8 FPS and 39.2 FPS respectively. In terms of CPU consumption the KVM system consumed 200% CPU, Xen consumed 208% and finally KVM consumed 207%. It is interesting to note that although all systems consumed nearly identical CPU resources both Virtualized systems provided far less than half the performance of the bare-metal system. It is clear that for Doom 3 the tested virtual machines are very poor performers.

B. Unigine Sanctuary Results

In Figure 3 we describe the results of the Unigine benchmark on our systems, the advanced 3D game engine benchmark. Our Bare metal system performs at 84.5 FPS, while Xen performs at 51 FPS and KVM performs at 44.3 FPS. Xen achieves only slightly better than half the performance of the Bare metal system, while KVM achieves slightly worse than half the performance. The CPU consumption on this test was 252% for the bare metal system, approximately 200% for Xen and 223% for KVM. In this experiment the both virtualized systems consume considerably less CPU then the previous Doom 3 benchmark. However, this should perhaps not be too surprising as they are processing frames at nearly half the rate of the Bare-metal machine. Interestingly, although performance of the VMs is still far below the bare-metal machine, they is a much smaller decrease when compared to the Doom 3 benchmark. We conjecture that this may be due to the fact that unlike the Doom 3 time demo benchmark, which simply renders a preset number of frames as fast as it can, the Unigine Sanctuary demo not only renders frames but also process some game logic such as in-game physics. This means that this benchmark is not simply trying to move data for processing on the GPU as fast as possible, as it must also coordinate game logic with the CPU.

C. LuxMark Results

The results of our systems running the LuxMark Ray Tracing benchmark are shown in Figure 4. This application uses the graphical technique ray tracing to draw a scene. The Bare metal system performs at 266 thousand samples per second and 2297 thousand rays per second. The virtualized system Xen performs at 258K samples per second and 2255k rays per second, and KVM performs at 240k samples per second and 2207k rays per second. Interestingly, for this application the Bare metal and the two virtualized systems perform almost identically. However, in terms of CPU consumption the virtualized systems consumed far more resources when compared to the bare-metal system. The bare-metal system consumed 78%, Xen consumed 146% and KVM 108%. When we compare the LuxMark's near native performance while virtualized to our Doom 3 and Unigine Sanctuary benchmarks abysmal virtualized performance it is obvious that there is some difference in the requirements of these different applications. Next, we discuss a major contributing factor, which leads to much poorer performance for our gaming applications when virtualized.

D. Discussion and Memory Bandwidth

We conjecture the reason why the performance remains high during the LuxMark benchmark for our virtualized system is because unlike the two game benchmark's, LuxMark moves much less data across the PCI-E bus from main memory to the GPU. This is because once the ray-tracing scene is transfered to the GPU only the resultant picture and status updates are sent back to the VM CPU and main memory. Unlike Doom 3 and Unigine Sanctuary benchmarks that are constantly moving data on to the GPU for processing. To motivate this conjecture we ran a simple memory bandwidth experiment written in OpenCl by Nvidia (Code is available in the Nvidia OCL SDK at: http://developer.nvidia.com/opencl). We tested three different copy operations, from host's main memory (DDR3) to the GPU device's global memory (GDDR5), from the devices global memory to the host's main memory and finally from the devices global memory to another location on the devices global memory. We give the results for host to device and device to host experiments in Figure 5. As can be seen the bare-metal system far out performs the VMs in terms of memory bandwidth across the PCI-E bus to the GPU. In terms of Host to Device performance both KVM and Xen achieve less than 59% of the performance of the bare-metal system, which is using the same hardware. Also, in terms of Device to Host memory transfer both virtualized systems only achieve 61% of the bandwidth of the non-virtualized bare-metal host. Interestingly, all systems achieved a near identical device to device copy throughput of just over 66400 MB/s. This indicates that once data is transfered from the virtual machine to the GPU, commands that operate exclusively on the GPU are much less susceptible to overhead created by the virtualization system. Next, we took a deeper look into the performance of these systems using software profiling techniques.

V. Detailed Profiling

Based on the performance variation experienced by our virtualized systems, we decided to use advanced hardware profiling techniques to further explore the systems and discover the route cause of the performance degradation. KVM was chosen as our representative hypervisor-based virtualization system to profile. The reason for this is that KVM runs on the unmodified Linux kernel, and is compatible with the standard performance profiling tools; Xen on the other hand, has a specialized kernel, and may have compatibility problems with these tools.

We use the Linux hardware performance analysis tool `Perf` and hardware counters to collect system level statistics such as processor cycles consumed, cache references, and processor context switches. For each experiment, we instruct `Perf` to collect samples during the duration of the experiment and then average them.

We present information on the performance of these platforms by calculating the Instructions Per Cycle (IPC), Stalled Cycles per Instructions, Cache-Miss percentage, and finally processor context switches. Before moving onto the results we will briefly provide a high-level description of each of the calculated metrics. *Instructions Per Cycle* (IPC) is a metric that is often used to compare the efficiency of an application/algorithm implementation on a specific architecture. Higher IPC is generally considered better as it implies that each clock cycle is processing more instructions and not waiting for memory or a shared device such as the processor's ALU. *Stalled Cycles per Instruction* indicate how many cycles the average instruction is stalled for while it is being pipelined through the processor. Stalls are often due to waiting on memory, or shared device contention. Next, we provide *Cache-Miss* percentage, which is the percentage of cache references which fail to find the data in the processor L2 and L3 cache. Finally, we measure *Context Switches*, which happen anytime a running thread voluntarily releases, or is forced to relinquish, control of a processing core so another thread can run.

	Bare-Metal	KVM
Instructions/Cycle (IPC)	0.84	0.64
Stalled Cycles/Instruction	0.47	0.82
Cache Miss Percentage	1.5%	3.71%
Context Switches	586/Sec	1588/Sec

TABLE I: Doom 3 Perf Results

	Bare-Metal	KVM
Instructions/Cycle (IPC)	1.03	0.54
Stalled Cycles/Instruction	0.26	1.24
Cache Miss Percentage	1.60%	2.85%
Context Switches	7059.3/Sec	12028/Sec

TABLE II: Sanctuary Perf Results

Using the same system setup and applications given in Section III we profiled both the bare-metal system and our KVM virtual machine. The results and our analysis are given below.

A. Profiling Results

Table I shows the results of the Doom 3 Profiling, which yielded some interesting results with the Virtualed KVM system being far worse in every measured metric. The IPC performance for KVM fell by over 20% and its Stalled Cycles/instruction nearly doubled. This is likely due to the fact that KVM's virtual CPU is having to wait many extra cycles to process the Doom 3 game data before sending it across the PCI-E bus to the GPU. The KVM system also suffers from over double the Cache Miss percentage. Further, the number of context switches experienced by KVM are over 2.5 times higher. The large increase in context switches are likely due

	Bare-Metal	KVM
Instructions/Cycle (IPC)	0.59	0.57
Stalled Cycles/Instruction	0.57	0.96
Cache Miss Percentage	3.68%	11.45%
Context Switches	16863.6/Sec	12028/Sec

TABLE III: LuxMark Perf Results

to the fact that virtualized system such as Xen and KVM must switch on the hypervisor to complete some privileged requests. It is likely that the increase in Context switches comes from memory access requests that the VMs currently do not have the mapping for. Also, some of the context switches may be caused by the fact that, due to a security issue in the x86 architecture, the hypervisor must be involved in answering some hardware interrupts [7].

Next, we focus on Unigine Sanctuary benchmark profiling, which can be seen in Table II. Similar to Doom 3 the Sanctuary game engine when run inside the KVM VM performed much worse in every metric when compared to the bare-metal hardware. The IPC for the benchmark now stumbles over 47% from 1.04 to 0.54. An even more profound increase can be found in the stalled cycles per instruction, which increase from the bare-metal baseline of 0.26 to 1.24, an increase of over four times for our KVM host. As stated before the decrease in IPC and increase in stalled cycles is likely due to the games logic being starved of needed data to render the scene. Like our Doom 3 profiling we also see a large increase in terms of both cache misses and context switches. Both are likely due to the impact of the hypervisor being constantly switched on to help with privileged requests from the benchmark and the VMs operating system.

Finally, we look at the LuxMark profiling results, which can be seen in Table III. In terms of IPC, the KVM benchmark did much better with the ray tracing application than the games, achieving only slightly worse performance than the bare-metal system. Stalled cycles increase from the bare metal baseline of 0.57 to 0.96 and increase of over 68%. Cache misses in KVM have the most significant increase yet, increasing from 3.68% to 11.45%. It is interesting to note that despite the decrease in performance due to both stalled cycles and poorer cache performance, we saw in Section IV-C that KVM performed nearly as well as the bare-metal system in this benchmark. Although this may seem surprising we conjecture that this is not unexpected, since our benchmark actually calculates performance based on the number of samples and rays processed by the GPU per second. The CPU has very little to do with this benchmark after the data is sent from main memory to the GPU. So although the operations performed in main memory and on the CPU may be degraded, they likely make up only a small amount of the work done in this benchmark.

VI. Further Discussion and Conclusion

We are actively working on research into Cloud gaming, and plan on exploring strategies to help improve the understanding of GPU accelerated systems and ways to improve performance. For a future work we are working on comparing the Intel VT-D and AMD-Vi implementation of the IOMMU device pass-through hardware. Also, a comparison of different operating systems and GPU features may reveal interesting insights such as Windows and its Direct X graphic standards. Further, the recent addition of cloud gaming enabled GPUs such as the Geforce Grid still need to be explored [16]. This paper primarily focused on the performance of the cloud based GPU, which is a critical component of a cloud gaming system. However, in a future work other critical systems such as video encoding and real-time streaming methods need to be explored in terms of suitability and performance for virtualized cloud gaming applications.

In this paper we investigated the possibility of using virtual machines with GPU pass-through devices to power cloud gaming platforms. However, our analysis shows that there are significant performance issues still to be addressed and explored before the widespread adoption of this architecture.

Acknowledgment

This research is supported by a Canada NSERC Discovery Grant, an NSERC Strategic Project Grant, a Swedish STINT Initial Grant, and a China NSFC Major Program of International Cooperation Grant (61120106008).

References

[1] M. Dowty and J. Sugerman, "Gpu virtualization on vmware's hosted i/o architecture," *ACM SIGOPS Operating Systems Review*, vol. 43, no. 3, pp. 73–82, 2009.

[2] Amazon Elastic Compute Cloud, http://aws.amazon.com/ec2/.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*. ACM, 2003, pp. 164–177.

[4] K. Ye, X. Jiang, S. Chen, D. Huang, and B. Wang, "Analyzing and modeling the performance in xen-based virtual cluster environment," in *2010 12th IEEE International Conference on High Performance Computing and Communications*, 2010, pp. 273–280.

[5] R. Russell, "virtio: towards a de-facto standard for virtual i/o devices," *SIGOPS Oper. Syst. Rev.*, vol. 42, pp. 95–103, July 2008.

[6] J. Liu, "Evaluating standard-based self-virtualizing devices: A performance study on 10 gbe nics with sr-iov support," in *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 1–12.

[7] A. Gordon, N. Amit, N. Har'El, M. Ben-Yehuda, A. Landau, A. Schuster, and D. Tsafrir, "Eli: bare-metal performance for i/o virtualization," *ACM SIGARCH Computer Architecture News*, vol. 40, no. 1, pp. 411–422, 2012.

[8] Y. Dong, X. Yang, J. Li, G. Liao, K. Tian, and H. Guan, "High performance network virtualization with sr-iov," *Journal of Parallel and Distributed Computing*, vol. 72, no. 11, pp. 1471–1480, 2012.

[9] N. Amit, M. Ben-Yehuda, and B.-A. Yassour, "Iommu: Strategies for mitigating the iotlb bottleneck," in *Computer Architecture*. Springer, 2012, pp. 256–274.

[10] L. Shi, H. Chen, J. Sun, and K. Li, "vcuda: Gpu-accelerated high-performance computing in virtual machines," *Computers, IEEE Transactions on*, vol. 61, no. 6, pp. 804–816, 2012.

[11] C. Reaño, A. J. Peña, F. Silla, J. Duato, R. Mayo, and E. S. Quintana-Orti, "Cu2rcu: Towards the complete rcuda remote gpu virtualization and sharing solution," in *High Performance Computing (HiPC), 2012 19th International Conference on*. IEEE, 2012, pp. 1–10.

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

[13] M. Jarschel, D. Schlosser, S. Scheuring, and T. Hoßfeld, "An evaluation of qoe in cloud gaming based on subjective tests," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2011 Fifth International Conference on*. IEEE, 2011, pp. 330–335.

[14] K.-T. Chen, Y.-C. Chang, P.-H. Tseng, C.-Y. Huang, and C.-L. Lei, "Measuring the latency of cloud gaming systems," in *Proceedings of the 19th ACM international conference on Multimedia*. ACM, 2011, pp. 1269–1272.

[15] M. Claypool, D. Finkel, A. Grant, and M. Solano, "Thin to win? network performance analysis of the onlive thin client game system," in *Network and Systems Support for Games (NetGames), 2012 11th Annual Workshop on*. IEEE, 2012, pp. 1–6.

[16] Nvidia Grid White Paper, http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf.