PROCEEDINGS OF SCIENCE

A Fast and Iterative Migration for GPU Applications

Xinhao Xu
Shanghai Jiao Tong University
School of software, 800 Dongchuan Road, Shanghai, China, 200240
E-mail: titanxxh@sjtu.edu.cn

Peng Yang
Gansu State Grid Information & Telecommunication Co.,Ltd.
629 East Xijin Road, Qilihe, Lanzhou, Gansu Province, China, 730050
E-mail: yangpeng@gs.sgcc.com.cn

Zhicheng Ma
Gansu State Grid Information & Telecommunication Co.,Ltd.
629 East Xijin Road, Qilihe, Lanzhou, Gansu Province, China, 730050
E-mail: mazc@gs.sgcc.com.cn

Lei Zhang
Gansu State Grid Information & Telecommunication Co.,Ltd.
629 East Xijin Road, Qilihe, Lanzhou, Gansu Province, China, 730050
E-mail: zhanglei@gs.sgcc.com.cn

Graphic Process Unit (GPU) is widely used for graphics computing and general-purpose computing. Therefore, the cloud environment starts to import GPU as a part of the infrastructure to provide new services, such as cloud gaming. Migration is a core feature in the management of cloud environment. However, it is not easy to migrate the Virtual Machine (VM) with the running GPU application using the current VM migration technique. To this end, we propose FIM, an open source[1] solution which can provide a fast and iterative migration for GPU applications. Our evaluation shows that 1) the downtime of the VM migration is 220-320 ms, 2) different GPU workloads degrade only 1.5-3.5% of performance, and 3) our solution only occupies 100-180 Mbits/Sec bandwidth during execution.

ISCC 2015
18-19, December, 2015
Guangzhou, China

[1]The source code is available at https://github.com/titanxxh/xengt-ha-xen and https://github.com/titanxxh/xengt-ha-kernel.

PoS(ISCC2015)051

1.Introduction

With the evolution of Graphic Process Unit (GPU), there has been an increasing trend in importing GPU to a cloud environment. A typical cloud environment leverages virtualization technique to consolidate multiple Virtual Machines (VM) on one physical host. The GPU feature in cloud environment enables the cloud vendor to provide new services such as cloud gaming [1], high performance computing [2]. However, GPU virtualization is quite immature. For example, GPU virtualization are not integrated into the management of cloud environment well. The key functionalities of management, taking snapshot and migration, are not supported.

To solve this problem, we develop a new software solution, FIM, on Xen [3] hypervisor by saving the whole state of VM iteratively, including all GPU specific states for modern GPU to a snapshot (or checkpoint, we use the terms: snapshot and checkpoint, interchangeably hereafter). The design of this iterative saving process also enables us to migrate a VM rapidly. The challenges are how to define the context of GPU without omission and how to reduce the saving time.

Following contributions have been made in this paper:

- We study the GPU context and introduce a fast and iterative migration for an unmodified VM with full GPU virtualization support.
- We carry out detailed experiments to demonstrate the correctness of our solution, as well as the downtime and the overhead during FIM migration. Our evaluation shows the downtime ranges from 220 ms to 320 ms. The performance degrades only 1.5-3.5% and FIM only transmits 100-180 Mbits data per second on average.

2.Related Work

There are some researches in checkpointing CUDA applications or GPU applications in application-level. Laosooksathit et al. [4] introduced a mechanism, VCCP, to do checkpointing in the GPGPU environment. The work is done by adding memory copying after the *syncthreads()* function. CheCUDA [5] hooks a part of CUDA driver API calls to record the modifications in main memory. VCCP and CheCUDA are not designed for general GPU applications. Flux [6] was proposed in a recent paper to migrate an Android application to another Android device. To overcome GPU migration, one of the main obstacles, the author circumvents the difficulties of saving the complex GPU context by leveraging three types of Android mechanisms: background execution, low-memory condition, and conditional initialization. Hence, it is unlike our method in this paper. Furthermore, none of these solutions are aimed at a full GPU virtualization environment.

Intel gVirt [7] , is a product-level full GPU virtualization solution. It achieves both good performance and scalability. Figure 1 demonstrates the basic architecture of gVirt, the native GPU driver in VM directly accesses the performance critical resources, command buffer, frame buffer, etc. The privileged resources, MMIO registers and Graphics Translation Table, are trapped by Xen and emulated by the mediator. The implementation of our work is mainly based on gVirt because of its open-source code and 95% of native performance[8].

3.Design and Implementation

FIM creates a complete snapshot for the unmodified VM, featured with full GPU virtualization, iteratively and fast. We can utilize it to migrate a VM to a remote machine in a short downtime. The challenges, however, exist in two areas: 1) how to define the context of a GPU, and 2) how to cut down the downtime without causing significant performance degradation.

3.1Architecture of FIM

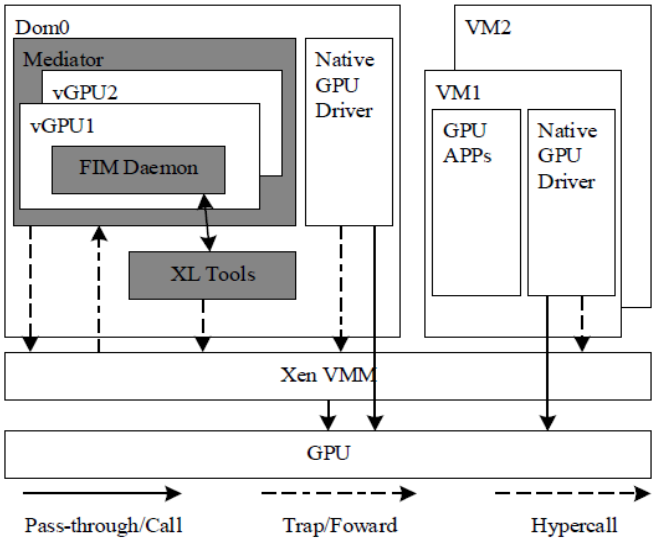

Figure 1 : The architecture of FIM.

Figure 1 shows the architecture of our design. Since our work is based on Intel gVirt, FIM reuses the architecture of gVirt and add/modify a few components (the gray parts) to achieve our goal. In this section, we only explain the changes made in detail since the other parts are demonstrated in Section 2.

Mediator: Mediator is implemented as a kernel module in Dom0 to manage vGPUs, as well as to emulate the access to privileged resources. To fit our demands, a replica for vGPU is created and updated when the vGPU is scheduled out.

FIM Daemon: FIM daemon is a daemon thread for each vGPU in the mediator. The daemon receives the requests from XL tools to save and restore the vGPU states.

XL Tools: XL tool is a user-level toolstack provided by Xen to manage the VMs (domains). In order to enable the domain saving thread to communicate with the FIM daemon, the original saving routine is tailored to fulfill our needs.

3.2GPU Context

The GPU context can be defined as the combination of following parts.

1. **All MMIO registers**, such as head/tail of a ring buffer, are the privileged resources.
2. **Graphics Translation Table (GTT)** translates the Graphics Memory Address to Guest Physical Address.
3. **Graphics Memory (GM)** is part of the main memory in Intel GPU. However, to support iterative pre-copying, each active page in Graphics Translation Table (GTT) should be saved in the stop-copying phase to guarantee the consistency.

4. **vGPU structure** is constructed by gVirt to manage the vGPU instance.

Based on the fact that these parts above are brought online when a vGPU is scheduled, we claim that these 4 parts together make up the abstraction for the physical GPU.

3.3Memory Copying

The method proposed in this paper is pre-copying plus stop-copying. Although the downtime is not the shortest (compared with post-copying), it is acceptable. The benefit of pre-copying is the negligible performance degradation. It is essential to point out that it consumes more network bandwidth in migration because each page is possible to be transmitted multiple times. Another crucial reason we choose this method instead of post-copying is the lack of page fault mechanism for Intel GPU.

3.4Workflow of FIM

The workflow of FIM can be divided into 3 stages: 1) Memory Pre-copying, 2) Suspension, and 3) Memory Stop-copying. The data copied out are immediately transmitted to the remote machine. The memory pages are applied on the remote machine as soon as received.

Stage 1: The log-dirty mode is toggled on for recording dirty pages. In this stage, the memory is copied out iteratively while the VM is still running. During the first iteration, all memory pages are copied out once. To expedite up the copying in this stage, a batch, 1024 by default, of pages are copied out per iteration. Those dirty pages in the previous iteration will be copied out in the next iteration. In our implementation, if the count of dirty pages does not converge during 30 iterations, the suspension stage will be forced to start.

Stage 2: To ensure to capture a consistent snapshot of the VM, the vCPU and vGPU are both scheduled out and the device model emulated by QEMU is also suspended at this time.

Stage 3: The whole system states are static after the suspension, so it is safe to copy out the dirty memory in last iteration. It usually is a very small part compared to the whole memory space. The GPU context in Section 3.2 also is saved at this stage. Eventually, the full snapshot is saved and transmitted. The migration process ends after the VM on the remote machine resumes.

4.Evaluation

4.1Configuration

CPU	Intel i5-4570 (4 cores, 3.2 GHz)
Graphics	Intel HD Graphics 4600
Memory	8 GB
NIC	Intel I217-LM 1Gbps
Hypervisor	Xen 4.5
gVirt	2015Q1 released
Dom0 & Guest OS	Ubuntu 14.04
Dom0 & Guest Kernel	Linux 3.18.0-rc7

Table 1 : Experimental environment.

All experiments in Section 4 are run on DELL Optiplex 9020-MT configured as Table 1. It should be noted that the guest OS kernel remains unmodified except a graphics driver patch that gVirt needs. The migration tests are deployed on two identical physical hosts mentioned above. The guest VM is configured with 4 vCPUs, 1 GB system memory and 512 MB global graphics memory.

4.2Correctness Verification

We test the migration process between a pair of hosts under different workloads. The migration starts with a number of iterations of pre-copying before the final stop-copying stage begins. The VM on the remote machine resumes after all the data received. The GPU applications all behave normally after the whole migration process ends.

4.3Downtime of FIM

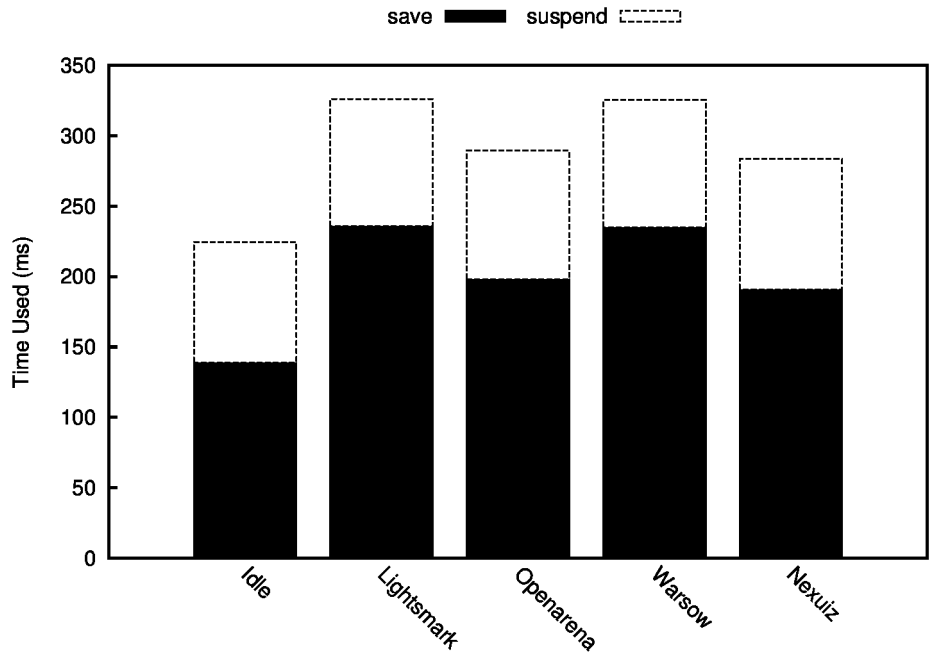

Figure 2 : Downtime of different benchmarks.

Figure 2 shows the migration downtime (time spent at Stage 3, which is mentioned in Section 3.4) of an idle VM and VMs running workloads. The workloads are light rendering benchmark and game benchmarks. The downtime mainly consists of two parts, suspension (80-90 ms) and saving (140-230 ms). From our observation, the total downtime ranges from 220 ms to 320 ms depending on different workloads. The diverge results from the saving time, which depends on the amount of dirty pages.

4.4Overhead of FIM

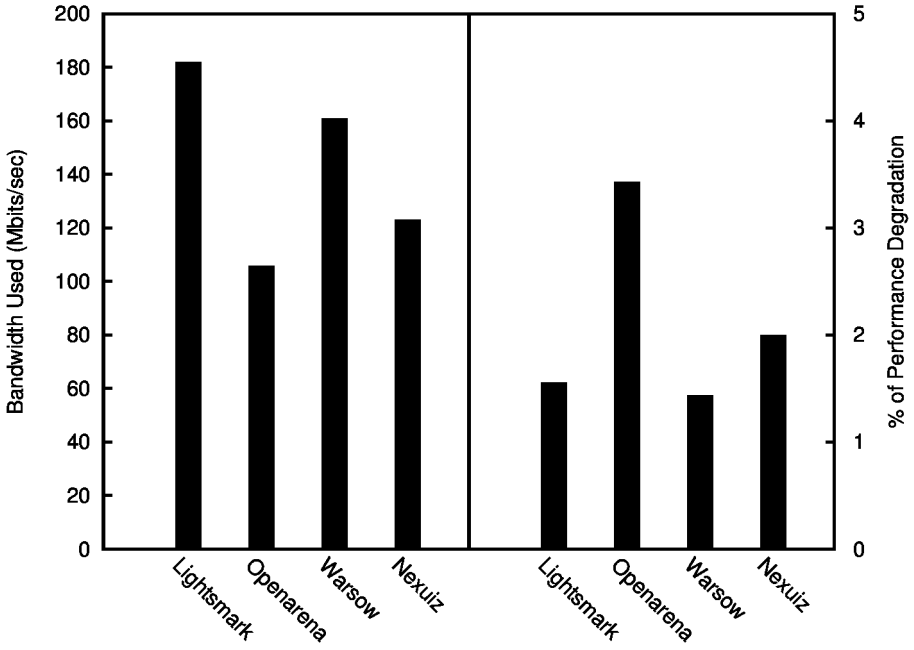

Figure 3 : Bandwidth and performance overhead.

The continuous pre-copying incurs overhead, so the performance degradation and the bandwidth overhead are demonstrated as Figure 3. During the pre-copying stage, the performance degrades 1.5% to 3.5% and the extra bandwidth consumption is 100-180 Mbits/sec depending on different workloads. Lightsmark and Warsow consumes more bandwidth because the amount of dirty pages is larger. Openarena suffers more performance degradation because this game is the most CPU-intensive one due to its complicated game logic. Since the iterations of pre-copying do not last too long, we believe the overhead is acceptable for our migration solution.

4.5Discussion

Although we have achieved iterative migration for the VM with full GPU virtualization, some limitations still exist in our implementation. In the current implementation of gVirt, the allocation strategy of the GM of a VM uses memory ballooning, which implies that each VM uses a part of GM of the physical GM. Restrained by the ballooning strategy of gVirt, the GM of the secondary VM allocated from the host GM should be at the same place as the primary VM. Fortunately, this restriction will disappear soon after the pre-released dynamic GM allocation strategy in gVirt is officially released.

5.Conclusion and Future Work

FIM is the first solution providing migration support for the VM with GPU virtualization. FIM saves and transmits the GPU context to the remote machine during the process of migration to keep the GPU applications alive. Additionally, the iterative memory pre-copying reduce the downtime caused by migration to only 220-320 ms. Different benchmarks degrade

only 1.5-3.5% of performance and the extra bandwidth is only 80-170 Mbits/s. Lastly, our work is open source and convenient for future collaboration.

As for future work, we will focus on further reducing the downtime by various means, including calculating the GPU dirty memory and utilizing new hardware features. Additionally, our implementation can be transplanted to other hypervisors like KVM .

References

[1] R. Shea, J. Liu, E. Ngai, Y. Cui. *Cloud gaming: architecture and performance*[J]. Network, IEEE 27(4), 16–21 (2013)

[2] A.J. Younge, J.P. Walters, S.P. Crago, G.C. Fox. *Supporting high performance molecular dynamics in virtualized clusters using IOMMU, SR-IOV, and GPUDirect*[C]. In: Proceedings of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. ACM, USA. pp, 31–38(2015)

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield. *Xen and the art of virtualization*[J]. ACM SIGOPS Operating Systems Review, ACM 37(5), 164–177 (2003)

[4] S. Laosooksathit, N. Naksinehaboon, C. Leangsuksan, A. Dhungana, C. Chandler, K. Chanchio, A. Farbin. *Lightweight checkpoint mechanism and modeling in GPGPU environment*[C]. Computing (HPC Syst). ACM, USA. pp, 13–20 (2010)

[5] H. Takizawa, K. Sato, K. Komatsu, H. Kobayashi. Checuda: *A checkpoint/restart tool for CUDA applications*[C]. In: Parallel and Distributed Computing, Applications and Technologies, 2009 International Conference. IEEE, USA. pp. 408–413(2009)

[6] A. Vant Hof, H. Jamjoom, J. Nieh, D. Williams. *Flux: multi-surface computing in Android*[C]. In: Proceedings of the Tenth European Conference on Computer Systems. ACM, USA. pp, 1-24(2015)

[7] K. Tian, Y. Dong, D. Cowperthwaite. *A full GPU virtualization solution with mediated pass-through*[C]. In: Proc. USENIX ATC 2014. USENIX, USA. pp, 121-132(2014)

[8] F. Michael, M.A. Murphy, and S. Goasguen. A study of a KVM-based cluster for grid computing[C]. In Proceedings of the 47th Annual Southeast Regional Conferenc. ACM, USA. pp, 34(2009)