Future Generation Computer Systems 63 (2016) 25–36

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

A user mode CPU–GPU scheduling framework for hybrid workloads

Bin Wang, Ruhui Ma, Zhengwei Qi *, Jianguo Yao, Haibing Guan

School of Software, Shanghai Jiao Tong University, Shanghai, China

HIGHLIGHTS

- We propose a user mode framework for CPU-GPU hybrid resource management.
- We analysis the drawbacks of the existing GPU and CPU scheduling systems.
- vHybrid is integrated by one GPU algorithm and two algorithms for CPU scheduling.
- The result outperforms existing SLA-aware GPU and CPU scheduling systems.

ARTICLE INFO

Article history:
Received 17 July 2015
Received in revised form
25 January 2016
Accepted 16 March 2016
Available online 11 April 2016

Keywords:
Hybrid workloads
CPU–GPU resource management
Scheduling
Switched control

ABSTRACT

Cloud platforms composed of multi-core CPU and many-core Graphics Processing Unit (GPU) have become powerful platforms to host incremental CPU–GPU workloads. In this paper, we study the problem of optimizing the CPU resource management while keeping the quality of service (QoS) of games. To this end, we propose vHybrid, a lightweight user mode runtime framework, in which we integrate a scheduling algorithm for GPU and two algorithms for CPU to efficiently utilize CPU resources with the control accuracy of QoS. vHybrid can maintain the desired QoS with low CPU utilization, while being able to guarantee better QoS performance with little overhead. Our evaluations show that vHybrid saves 37.29% of CPU utilization with satisfactory QoS for hybrid workloads, and reduces three orders of magnitude for QoS fluctuations, without any impact on GPU workloads.

1. Introduction

There is a virtually unanimous consensus in the community that Graphics Processing Unit (GPU) is one of the most important resources for cloud computing. This is because many of today's cloud graphic applications, such as cloud gaming, video rendering, and social networking, have very demanding GPU requirements, and poor GPU scheduling can directly affect application performance and degrade user experience. To provide adequate GPU resources, GPU virtualization has been widely studied. For example, the work in [1–7] leveraged GPU virtualization for general purpose computing on GPUs. In our previous work, VGRIS and vGASA [8,9], achieved a Service Level Agreement (SLA)-aware GPU resource scheduling policy. They accelerated the running times of GPU-intensive workloads and guarantee the Quality of Service (QoS) in the cloud gaming platform.

However, only the GPU scheduling mechanism is not sufficient to effectively enhance the overall computing resource utilization in a CPU–GPU hybrid platform, where GPU-intensive and CPU-intensive workloads coexist. For example, in Amazon EC2 GPU instances [10], each Virtual Machine (VM) can either run the graphic rendering applications that need GPU resources (e.g., cloud gaming), or have the CPU-intensive workloads consume the CPU resources. In addition, cloud users need to make sure their data remain intact after uploading to the remote server. Thus, a physical server runs client applications such as Apache [11]. They also require high computation in the CPU. As a result, a strategy to allocate fair-share CPU resources among CPU workloads while guaranteeing the QoS of GPU computation is deemed as a crucial demand. Unfortunately, neither CPU nor GPU resources are efficiently utilized in the cloud. One likely reason is that the workloads running in the virtualized environment are significantly affected by other workloads. The default scheduling mechanisms are usually designed and embedded by the hardware vendors who do not consider the use of cloud computing in their design nor productions. Therefore, the scheduling policies would allocate excessive CPU and GPU resources for a workload, but starve the other. On the other hand, most of the previous work only considers proposals for the scheduling policy of CPU or GPU individually. For

* Corresponding author.
E-mail addresses: binqbu2002@sjtu.edu.cn (B. Wang), ruhuima@sjtu.edu.cn (R. Ma), qizhwei@sjtu.edu.cn (Z. Qi), jianguo.yao@sjtu.edu.cn (J. Yao), hbguan@sjtu.edu.cn (H. Guan).

http://dx.doi.org/10.1016/j.future.2016.03.011
0167-739X/

example, under the use of vGASA in a hybrid platform, each GPU demand attains its basic QoS but the response time of the CPU workload still fluctuates hugely every second (Section 2).

Motivated by these problems, this paper proposes vHybrid, a user mode lightweight CPU–GPU resource management framework with new open loop and adaptive control algorithms, optimizing the CPU utilization while keeping the QoS of GPU-intensive workloads. All of the work is implemented by library API interception, without modifying either the architecture or the source code of the Operating System (OS), applications, or VMs.

vHybrid consists of two CPU control policies and a GPU scheduling policy for specific resource management. To meet SLA requirements for all the GPU resources, vHybrid borrows the SLA-aware scheduling policy from vGRIS. For CPU-intensive workloads, the first CPU scheduling policy, open-loop control allocates CPU resource using a preset model calculated by a relationship between our scheduling intensity and application performance. It can maximize the CPU resource usage and vacate as many resources as possible for the other waiting applications. The second policy is adaptive scheduling, which dynamically and instantaneously modifies the CPU resource required proportionally by analyzing SLA performance of applications. This process costs additional CPU resources, but ensures a substantial increase in control accuracy of QoS.

The main contributions of this paper are as follows:

- We propose vHybrid, a user mode framework for CPU–GPU hybrid resource management, without source code level changes in applications, guest OSes, or host OSes. Moreover, by leveraging the mature library API interposition, the guest OS and the network applications remain unmodified. This enables vHybrid to be deployed in any clusters to effectively schedule CPU and GPU resources.
- vHybrid is integrated by one existing algorithm for GPU scheduling and two new algorithms for CPU scheduling to trade off overhead and control accuracy of the CPU–GPU hybrid workloads. Open loop scheduling is a straightforward implementation that roughly minimizes the CPU resource occupation. Adaptive scheduling achieves better control accuracy of QoS for CPU-intensive workloads, with a small overhead.
- We conduct experiments of real-world workloads, showing that open-loop scheduling saves 37.29% of CPU utilization with satisfactory QoS for hybrid workloads, and adaptive scheduling achieves better QoS performance (i.e., reducing QoS fluctuations by three orders of magnitude), without compromising GPU workloads. The result outperforms existing SLA-aware GPU scheduling systems such as vGASA.

The rest of the paper is organized as follows. Section 2 gives the motivation of this paper. Section 3 introduces the architecture of vHybrid. Based on the architecture above, two CPU scheduling policies, open-loop scheduling and adaptive scheduling, are proposed in Section 4. Comprehensive experiments and performance evaluation are presented in Section 5. We discuss the related work in Section 6, followed by conclusions in Section 7.

2. Motivation

To motivate the need for both GPU and CPU scheduling policies, we start by introducing a few cases of CPU–GPU hybrid workloads for cloud computing in which QoS is desired. Subsequently, we show that both the default and SLA GPU scheduling would sometimes waste GPU and CPU resources for hybrid workloads. Finally, we conduct the experiments on our machine testbed and show how the SLA-aware GPU scheduling fails to provide good performance for CPU-intensive workloads.

2.1. CPU–GPU hybrid workloads

The CPU–GPU hybrid workloads have been employed as core roles to provide cloud services. For example, virtualization technology provides numbers of VMs with a software interface that is different from its underlying hardware counterpart in the cloud. Each VM can choose between running either CPU-intensive or GPU-intensive workloads. These workloads are sometimes mission-critical, which require high QoS of CPU and GPU resources according to SLA. Meanwhile, the hosts, which run the hypervisors, would sometimes launch other GPU and CPU computation instances. Therefore, poor resource sharing among different workloads would degrade some applications whose workloads become starved.

In this paper, our CPU–GPU hybrid workloads include three GPU-intensive instances (DiRT 3, Far Cry 2, StarCraft 2), and one CPU-intensive workload (Apache). The former three games will generate intensive graphic computational loads. During the process of gaming, the pictures usually have widely varying Frames Per Second (FPS). The FPS may continuously vary as the game scenes change, in which the QoS estimation is more complex than other GPU-intensive workloads. On the other hand, the Apache read and write [12] will continuously generate numbers of tasks for CPU response, which received 2000 requests per second and completed 200 requests simultaneously.

2.2. Native GPU scheduling for hybrid workloads

The native GPU scheduling as well as the CPU computation determines the FPS. Some CPU computation prepares the data for the GPU, e.g., calculating objects in the upcoming frame according to the game logic. The data are next uploaded to the GPU buffer, and then the GPU performs the computation. Finally, the calculation result is sent back to the main memory for the next iteration, or output to the screen. The GPU computation library depends on the application, e.g., Direct3D [13] or OpenGL [14] for gaming and rendering, or DirectCompute [15], OpenCL [16], or CUDA [17] for general-purpose GPU computation. The detailed API depends on the graphics library, e.g., `glutSwapBuffers` from OpenGL and `Present` from Direct3D.

We now present some experiments to observe the performance of a native scheduling policy (Direct3D) in our hybrid platform while guaranteeing the QoS of each workload. Each GPU workload concurrently runs in a separate VM which is configured with Windows 7[1] as the guest OS supporting the Direct3D graphics library, while the host concurrently launches the Apache service. Fig. 1(a) shows the experimental results.

Compared to their original performance under the same game configuration, the FPS of the GPU-intensive workloads is reduced dramatically due to the poor scheduling. The three games have FPS that fluctuate between 20 and 60. The most likely reason is the asynchronous and non-preemptive nature of the GPU process. For example, the default GPU scheduling mechanism in the Direct3D runtime library tends to allocate resources on a first-come first-serve manner, which results in an excessive FPS rate for low-end games and unplayable FPS rate for GPU intensive games when they are running concurrently in separate VMs. Graphics APIs also typically work in an asynchronous way to maximize hardware performance. APIs such as `DisplayBuffer` immediately return

[1] Note that many CPU scheduling policies can easily be implemented in Linux, e.g., `cgroups`, but we choose Windows as the basic OS for our implementation because today's most GPU computational applications are usually developed in the Windows platform. For example, the Direct3D framework provides many rendering functions for games in Windows.