International Journal on Applications in Information and Communication Engineering
Volume 1: Issue 3: March 2015, pp 35-38. www.aetsjournal.com ISSN (Online) : 2394-6237

Evaluation of Grid Computing and their Challenges

M.Sarumathi ,S.Sakthivel Balaji

***Abstract*—** The emergence of compute unified device architecture (CUDA), which relieved application developers from understanding complex graphics pipelines, made the graphics processing unit (GPU) useful not only for graphics applications but also for general applications. In this paper, we are trying to introuduced cycle sharing system named as GPU and also scientific applications. Our cycle sharing system to implement a guest application for to cooperative multitasking technique, remotely on a donated host machine without causing a significant slowdown on the host machine. Because our system has develop among the pre-CUDA era, we also tell how the evolution of GPU architectures influenced our system.

***Keywords*—** GPGPU, cooperative multitasking, cycle sharing, grid computing, volunteer computing etc

I. INTRODUCTION

The graphics processing unit (GPU) [1]–[3] is a hardware component mainly designed for acceleration of graphics tasks such as real-time rendering of three-dimensional (3D) scenes. To satisfy the demand for real-time rendering of the GPU has higher arithmetic performance and memory bandwidth than the CPU. The emergence of compute unified device architecture (CUDA) [4] allows application developers to easily utilize the GPU as an accelerator for not only graphics applications but also general applications. Using an application hotspot, CUDA can be eliminated by implementing the corresponding code as *a kernel function*, which runs on a GPU in parallel. As a result, many research studies use the GPU as an accelerator for compute- and memory-intensive applications [5]–[8].

As such a study, the Folding@home project [9], [10] employed 20,000 idle GPUs to accelerate protein folding simulations on a grid computing system. Although there are many types of grid systems, a grid system in this paper have volunteer computing system that to shares network-connected computational resources for accelerate applications. We try to denote *a host* as a user to donates a computational resource and *a guest* as a user who uses the donated resource for acceleration (Fig. 1). A host task corresponds to a local task generated by daily operations on a resource and a guest task corresponds to a grid task to be accelerated remotely on the donated resource.

Host and guest tasks can be executed simultaneously on a donated resource to shared between host .however, current GPU architectures don't support preemptive multitasking, so that a guest task can intensively occupy the resource until its completion. Thus, simultaneous execution of multiple GPU programs significantly drops the frame rate of the host machine. To make the matter worse, this performance degradation increases with kernel execution time. For example, our preliminary results [11] show that a guest task running on a donated machine causes the machine to hang and reduces its frame rate to less than 1 frame per second (fps). Accordingly, GPU-accelerated grid systems have to not only minimize host perturbation (i.e., frame rate degradation) but also maximize guest application performance.

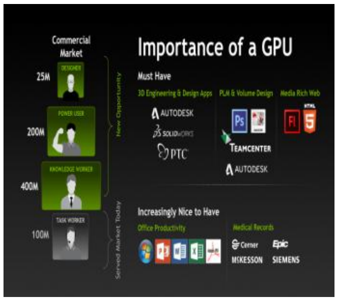


Figure 1. Overview of GPU grid.

In this paper, we introduce a GPU-accelerated grid system capable of exploiting short idle time such as hundreds of milliseconds. Our cycle sharing system extends a coopera-tive multitasking technique [12], which is useful to execute a guest application remotely on a donated host machine without causing a significant slowdown on the machine. We also present how the evolution of GPU architectures influenced our system.

II. PAST: PRE-CUDA ERA

Before the release of CUDA, the only way to implement GPU applications was to use a graphics API such as DirectX [13] or OpenGL [14]. Despite this low programmability, some

M.Sarumathi ,S.Sakthivel Balaji are UG scholars, Department of Computer Sciene and Engineering, Dhirajlal Gandhi college of Technology ,Salem.(email : saruvannan@gmail.com ,sakthivelbalajisundarewaran@gmail.com)

grid systems tried to accelerate their computation using the GPU. The Folding@home and GPUGRID.net systems [9], [15] are based on Berkeley Open Infrastructure for Network Computing (BOINC) [16], which employs a screensaver to avoid simultaneous execution of multiple GPU programs on a host machine. In system detected an of idle machine according to screensaver activation. A running guest task can be suspended (1) if the screensaver turns off due to host's activity or (2) if the host machine executes DirectX-based software with exclusive mode. The exclusive mode here is useful to avoid a significant slowdown on the host machine if both guest and host applications are implemented using DirectX.

Kotani *et al.* [11] also presented a screensaver-based system that monitors video memory usage in addition to host's activity. By monitoring video memory usage, the system can avoid simultaneous execution of host and guest applications though the host applications are not executed with exclusive mode. Screensaver-based systems are useful to detect long idle periods spanning over a few minutes. However, short idle periods such as a few seconds cannot be detected due to the limitation of timeout length. When every system to applied any biological application to evaluate the impact of utilizing idle GPUs in a laboratory environment [17].

Caravela [18] is a stream-based distributed computing environment that encapsulates a program to be executed in local or remote resources. our environment try to focus on the encapsulation and assumes that resources are dedicated to guests. The perturbation issue to be solved for non-dedicated systems, is not by addressed.

III. PRESENT: CUDA ERA

To detect short idle time spanning over a few seconds, Ino *et al.* [19] presented an event-based system that keyboard activities, video memory usage and CPU usage. Similar to screensaver-based systems, they assume that idle resources do not have mouse and keyboard events for one second. Furthermore, they divide guest tasks into small pieces to minimize host perturbation by completing each piece within 100 milliseconds. Owing to this task division, their system realizes the minimum frame rate of around 10 fps.

One drawback of this previous system is that the GPU is not always busy when the mouse or keyboard is operated interactively by the host. To make the matter worse, mouse and keyboard events are usually recorded at short intervals such as a few seconds. Consequently, resources can frequently done between idle and busy states. This alternation can make guest tasks be frequently cancelled immediately after their assignment, because idle host machines turn to be busy before task completion, because a state transition on a resource causes an interaction between the resource and the server.

Some research projects developed GPU virtualization technologies to realize GPU resource sharing. To the best of our knowledge, NVIDIA GRID and Gdev [20] are the only systems that virtualize a physical GPU into multiple logical GPUs and achieve a prioritization, isolation, and fairness scheme. Gdev currently supports Linux systems. Although virtualization technologies are useful to deal with the host perturbation issue, they require system modifications on host machines. We think that the host perturbation issue should be solved at the application layer to minimize modifications at the system level.

rCUDA [21] is a programming framework that enables remote execution of CUDA programs with small overhead. A runtime system and a CUDA-to-rCUDA transformation framework are provided to intercept CUDA function calls and redirect these calls to remote GPUs. Because rCUDA focuses on dedicated clusters rather than shared grids, the host perturbation issue is not solved. A similar virtualization technology was implemented as a grid-enabled programming toolkit called GridCuda [22].

vCUDA [23] allows CUDA applications executing within virtual machines to leverage hardware acceleration. Similar to rCUDA, it implements interception and redirection of CUDA function calls so that CUDA applications in virtual machines can access a graphics device of the host operating system. The host perturbation issue is not tackled.

IV. OUR CYCLE SHARING SYSTEM

Our cycle sharing system is capable of exploiting short idle time such as hundreds of milliseconds without dropping the frame rate of donated resources. To realize this, we exe-cute guest tasks using a cooperative multitasking technique [12]. Our system extends this technique to avoid mouse and keyboard monitoring. Similar to [19], our system divides guest tasks into small pieces to complete each piece within tens of milliseconds. Our extension can be summarized in two-fold: (1) a relaxed definition of an idle state and (2) two execution modes, each for partially and fully idle resources (Fig. 2).

The relaxed definition relies only on CPU and video memory usages. Consequently, there is no need to monitor mouse and keyboard activities. A resource is assumed to be busy if both CPU and video memory usages exceed 30% and 1 MB, respectively (Fig. 3). For idle resources, our system locally selects the appropriate execution mode for guest tasks. Consequently, most state transition can be processed locally, avoiding frequent communication between resources and the resource management server.

The two execution modes are as follows:

1) A periodical execution mode for partially idle re-sources. For partially idle resources, our system uses the periodical mode with tiny pieces of guest tasks. Each piece here can be processed within a few ten milliseconds, and a series of pieces are processed at regular intervals $1=F$ to keep the frame rate around F fps. In other words, F is the minimum frame rate desired by the host.
2) A continuous execution mode for fully idle resources. For fully

idle resources, on the other hand, our system

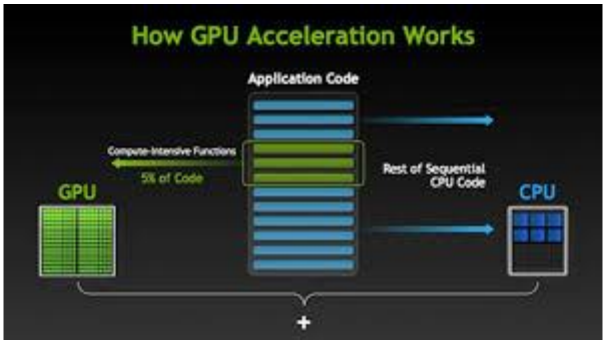


Figure 2. Our cooperative multitasking technique. (a) Periodical execution mode executes guest tasks at regular intervals $1=F$, where F is the minimum desired frame rate. (b) Continuous execution mode intensively executes guest tasks.

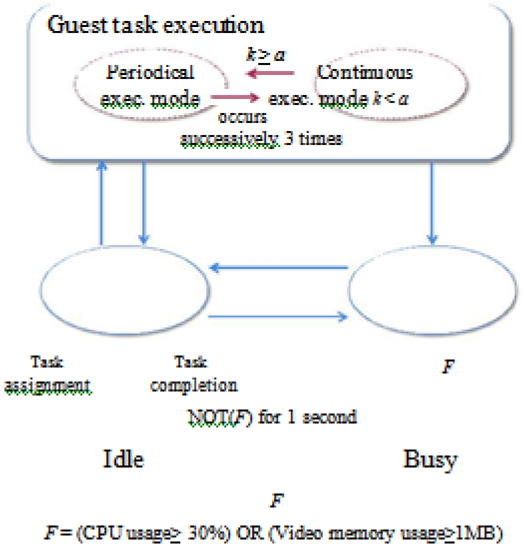


Figure 3. State transition diagram for cooperative multitasking.

switches its execution mode to the continuous mode with small pieces of guest tasks. A series of pieces is continuously processed on the GPU. The continuous execution mode allows guests to execute their tasks on lightly-loaded resources that are interactively operated by hosts.

In order to determine whether a resource is partially idle or fully idle, our system estimates GPU workload with keeping the frame rate as possible as we can. To realize such a low-overhead estimation, our system executes a null kernel before guest task execution and measures its execution time k. A null kernel is a device function that immediately returns after its function call. The measured time k is then compared to the pre-measured time obtained by dedicated execution on the same resource. We assume that the resource is partially idle if k and is fully idle if k < occurs successively three times.

False positive and false negative cases can occur when

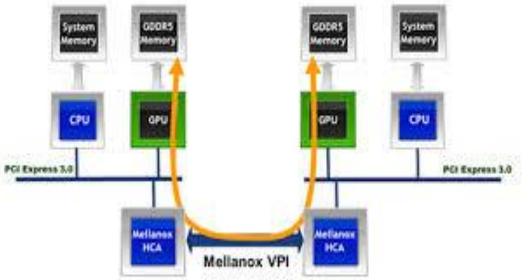
Table I
SPECIFICATION OF EXPERIMENTAL MACHINES.

Item	Specification
OS	Windows 7 Professional 64 bit
CPU	Intel Core i7-3770K (3.5 GHz)
Main memory	16 GB
GPU	NVIDIA GTX 680
CUDA	5.0
Video driver	310.90

Table II
SYSTEM UPTIME IN HOUR.

Host machine	#1	#2	#3	#4
Uptime	135.1	15.9	197.0	81.6

switching to the continuous execution mode. The former leads to excessive execution of guest tasks, failing to keep the original frame rate obtained withoug guest task execu-tion. On the other hand, the latter fails to maximize guest task throughput, but frame has kept. We think that the latter issue is not critical for our system, because our first priority is minimization of host perturbation. In contrast, we prevent the former case by confirming k < three times, which avoids immediate transition to the continuous execution mode.



V.CONCLUSION

We have introduced a GPU-accelerated grid system ca-pable of utilizing short idle time spanning over hundreds of milliseconds. Our cooperative multitasking technique realizes concurrent execution of host and minimizing host perturbation. Our technique eliminates the mouse and keyboard monitoring process required in previous systems. Our monitoring process checks only CPU and video memory usages, according to a relaxed definition of an idle resource. It relaxation can not reduces only the number of state transitions but also that of communication messages between resources and the resource management server.

We performed case study in which our system is applied to four desktop machines of our laboratory. Compared to a previous screensaver-based system, our cooperative sys-tem detected 1.7 times longer idle time. Consequently, our system achieved a 91% higher guest throughput, realizing efficient utilization of idle resources. Furthermore, our sys-tem reduced the server workload by reducing the number of state transitions by 96%.

Future work includes detailed evaluation using more practical applications in a large-scale environment. We plan to apply our system to a homology search problem [8]. NVIDIA has announced that their next-generation GPU architectures, Maxwell and Volta, will support preemption and unified virtual memory. Such preemptive architectures will require a task scheduler to find the best tradeoff point between the frame rate of host machines and the throughput of guest tasks

REFERENCES

[1] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "NVIDIA Tesla: A unified graphics and computing architec-ture," *IEEE Micro*, vol. 28, no. 2, pp. 39–55, Mar. 2008.

[2] "CUDA C Programming Guide Version 5.5," Jul. 2013, http://docs.nvidia.com/cuda/pdf/CUDA C Programming Guide.pdf.

[3] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, Kruger,¨ A. E. Lefohn, and T. J. Purcell, "A survey of general-purpose computation on graphics hardware," *Com-puter Graphics Forum*, vol. 26, no. 1, pp. 80–113, Mar. 2007.

[6] M. Garland, S. L. Grand, J. Nickolls, J. Anderson, J. Hard-wick, S. Morton, E. Phillips, Y. Zhang, and V. Volkov, "Parallel computing experiences with CUDA," *IEEE Micro*, vol. 28, no. 4, pp. 13–27, Jul. 2008.

[7] Y. Okitsu, F. Ino, and K. Hagihara, "High-performance cone beam reconstruction using CUDA compatible GPUs," *Parallel Computing*, vol. 36, no. 2/3, pp. 129–141, Feb. 2010.

[8] Y. Munekawa, F. Ino, and K. Hagihara, "Accelerating Smith-Waterman algorithm for biological database search on CUDA-compatible GPUs," *IEICE Trans. Information and Systems*, vol. E93-D, no. 6, pp. 1479–1488, Jun. 2010.

[9] The Folding@Home Project, "Folding@home distributed computing," 2010, http://folding.stanford.edu/.

[10] A. L. Beberg, D. L. Ensign, G. Jayachandran, S. Khaliq, and S. Pande, "Folding@home: Lessons from eight years of volunteer distributed computing," in *Proc. 26th IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS'09)*, Apr. 2009, 8 pages (CD-ROM).

[11] Y. Kotani, F. Ino, and K. Hagihara, "A resource selection system for cycle stealing in GPU grids," *J. Grid Computing*, vol. 6, no. 4, pp. 399–416, Dec. 2008.

[12] F. Ino, A. Ogita, K. Oita, and K. Hagihara, "Cooperative multitasking for GPU-accelerated grid systems," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 1, pp. 96–107, Jan. 2012.

[13] D. Blythe, "The Direct3D 10 system," *ACM Trans. Graphics*, vol. 25, no. 3, pp. 724–734, Jul. 2006.

[14] D. Shreiner, M. Woo, J. Neider, and T. Davis, *OpenGL Programming Guide*, 5th ed. Reading, MA: Addison-Wesley, Aug. 2005.

[15] GPUGRID.net, 2010, http://www.gpugrid.net/.

[16] D. P. Anderson, "BOINC: A system for public-resource com-puting and storage," in *Proc. 5th IEEE/ACM Int'l Workshop Grid Computing (GRID'04)*, Nov. 2004, pp. 4–10.

[17] F. Ino, Y. Kotani, Y. Munekawa, and K. Hagihara, "Harness-ing the power of idle GPUs for acceleration of biological sequence alignment," *Parallel Processing Letters*, vol. 19, no. 4, pp. 513–533, Dec. 2009.

[18] S. Yamagiwa and L. Sousa, "Caravela: A novel stream-based distributed computing," *IEEE Computer*, vol. 40, no. 5, pp. 70–77, May 2007.

[19] F. Ino, Y. Munekawa, and K. Hagihara, "Sequence homology search using fine grained cycle sharing of idle GPUs," *IEEE Trans. Parallel and Distributed Systems*, vol. 23, no. 4, pp. 751–759, Apr. 2012.

[20] S. Kato, M. McThrow, C. Maltzahn, and S. Brandt, "Gdev: First-class GPU resource management in the operating sys-tem," in *Proc. 2012 USENIX Ann. Technical Conf. (ATC'12)*, Jun. 2012, 12 pages (CD-ROM).

[21] C. Reano,˜ A. J. Pena,˜ F. Silla, J. Duato, R. Mayo, and E. S. Quintana-Ort´ı, "CU2rCU: towards the complete rCUDA remote GPU virtualization and sharing solution," in *Proc. 19th Int'l Conf. High Performance Computing (HiPC'12)*, Dec. 2012, 10 pages (CD-ROM).

[22] T.-Y. Liang, Y.-W. Chang, and H.-F. Li, "A CUDA program-ming toolkit on grids," *Int'l J. Grid and Utility Computing*, vol. 3, no. 2/3, pp. 97–111, May 2012.

[23] L. Shi, H. Chen, and J. Sun, "vCUDA: GPU-accelerated high-performance computing in virtual machines," *IEEE Trans. Computers*, vol. 61, no. 6, pp. 804–816, Jun. 2012.