iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 2 **Issue:** IX **Month of publication:** September 2014

DOI:

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

ISSN: 2321-9653

INTERNATIONAL JOURNAL FOR RESEARCH IN APPLIED SCIENCE AND ENGINEERING TECHNOLOGY (IJRASET)

Operating System Challenges for GPU Resource Management

Jyoti Chandel[1], Neha Gupta[2], Jyoti Yadav[3]

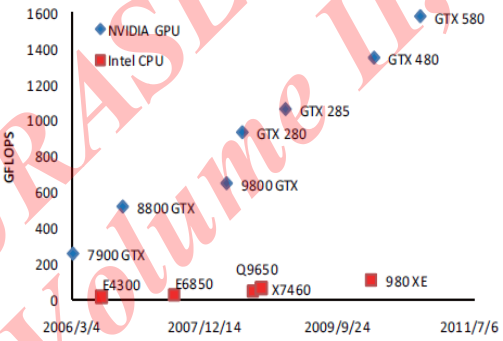
Dronacharya College of Engineering,
Gurgaon, HR

Abstract: ***The graphics processing units(GPU) have rapidly evolved to become high performance accelerators for data-parallel computing. For accelerating graphics and data-parallel compute-intensive applications the GPU is a most popular platform. As technology stands today , the GPU is the most well-suited platform. GPUs can accelerate the processing speeds significantly. It significantly outperforms traditional multi-core processor in performance and energy efficiency. The efforts of GPU resource management for multitasking environments are lacking models, designs and implementation.***

This paper identifies a GPU resource management model to provide a basis for operating systems research using GPU technology .We present design concepts for GPU resource management. In this paper, we discuss about the list of operating system challenges for GPU resource management. The specific idea of GPU scheduling is for real-time system. Our primary evaluation demonstrate that the performance of open-source software is competitive with that of proprietary software , and hence operating systems research can start investigating GPU resource management.

I. INTRODUCTION

The major concerns for today's computer systems are performance and energy . GPUs have become a very powerful platform embracing a concept of heterogeneous many-core processors. Since the 1960s processors have shown an exponential improvement in performance fueled by advances in semiconductor technology , allowing to increase the clock speed and architecture innovations increasing the amount of workdone per cycle. In the early 2000s, chip manufacture had competed on processor clock rate to continue the performance improvements in their product lines .In 2002 the Intel Pentium 4 processors was the first commercial product that exceeded a clock rate of 3GHz. Since about 2005 this exponential growth of single-core performance has significantly flattened out. This paradigm



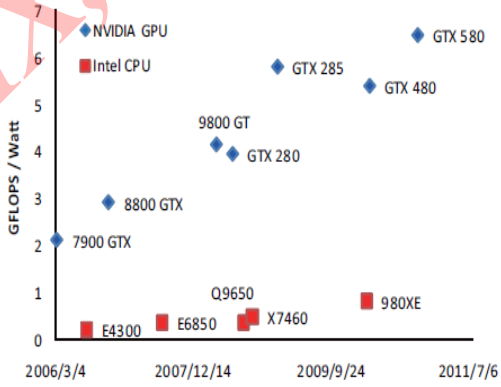


Figure 1.Performance trends on the Well-known GPU and CPU Architectures.

IJRASET: ISSN: 2321-9653
Volume II, Issue IX, September 2014

ISSN: 2321-9653

shift was a break through to achieve high performance with low energy .Figure 1 illustrates the performance trends on the well-known GPU and CPU architecture from NVIDIA and Intel[1]. Latest GPUs contains 100s of microprocessor, ability of achieving up to one TFLOPS for single-precision arithmetic and over eighty GFLOPS for dual-precision calculations. The GPU is the most well-suited platform for today's technology .In the future product lines of BMW[11] vehicles will used NVIDIA GPUs for infotainment systems.

The advantage of GPU is non-trivial performance which comes from hundreds of processing core integrated on a chip. The CPU is also less preferable than the GPU in performance per watt. Specifically, the CPU is about 7times less energy-efficient than the GPU today.



GeForce 6600GT (NV43) GPU

The TOP500[12] supercomputing sites declared in Nov ,2011 that three of the top five supercomputers comprise GPUs Clusters and also provide performance improvement for scientific applications. The GPU also provide the benefit of large-scale storage systems. The ASUS Eee Pad Transformer Prime ,also leverage embedded GPUs to expand the Performance under power constraints. In the implant system domain, a latest design of Camegie Mellon's autonomous vehicle[8] provides four NVIDIA's GPUs to raise its computing power required for autonomous driving tasks, including vision-based perception and motion planning. A number of parallel algorithms such as vehicle planning, sensor fusion, computer vision and graphics sub-systems must operate and coordinate in real-time so that vehicles can become semi autonomous and fully autonomous eventually. A case study from Stanford disclose that the GPU can speed up computer vision applications for autonomous driving by 40 times compared to CPU execution. The GPGPU[10] is known as a rapid growth of general-purpose computing on GPUs which is supported by recent advances in programming technology authorize the GPU to be used easily for general "compute" problems.

GPU resource management get very limited support from Operating System in commodity software of operating systems. Some of the multi-tasking, such as fairness, prioritization and isolation are not supported at all. The research society has implement various approaches to GPU resources management recently. Our research paper identifies various ways towards operating systems challenges for GPU resources management .A critical issue for GPU resource management is to explore system design and implementation with missing information in open-source software. We present potential solutions and initial ideas to these open problems in this paper.

II. OUR SYSTEM ASSUMPTION

This paper contemplate on a system composed of a GPU and a multi-core CPU. Various GPU architecture are present today . While many established microprocessor format is based on the X86 CPU architecture suitable across many years. To change GPU architecture designer need lots of year. Fermi architecture was released by NVIDIA[3] in 2011 which support both computer and graphics program. Fermi architecture is the main focus of this paper but the concept is also applicable to other architecture . On-board GPU is also assumed.

A new X86 -based architecture is provided by Intel, which integrate the GPU on a chip, GPUs on a board are still more popular today . The CPU and GPU operate asynchronously that we assume in on-board GPU model . Or we can say that GPU contexts and CPU contexts are separately processed. The piece of code is offloaded from the CPU when the user programs launch that code onto the GPU to get accelerated.

IJRASET: ISSN: 2321-9653 Volume II, Issue IX, September 2014

ISSN: 2321-9653

III. PROGRAMMING MODEL

To accelerate a particular program code we use the GPU device rather than CPU device. The user program start the execution on the CPU and forward the piece of code to GPU kernels and after that it get accelerated. The user program take at least three major steps to accelerate on the GPU.

1. Memory Allocation: The memory allocation of user programs are required for computation. The GPU has several types of memory are shared ,local , global, constant and heap.

2. Data Copy: The GPU Kernel starts on the GPU after the input data copy from the host to the device memory . To return the computed result to user programs the output data is copy back from the device to the host memory.

3. Kernel Launch: The GPU itself is not a control unit as GPU-accelerated program code must be launched from the CPU to the GPU at runtime.

The memory-allocation phases must manage the device memory address regions available for each request. We need to access the GPU to move data between the host and the device memory and Launch the GPU program code. A brief execution flow of GPU-

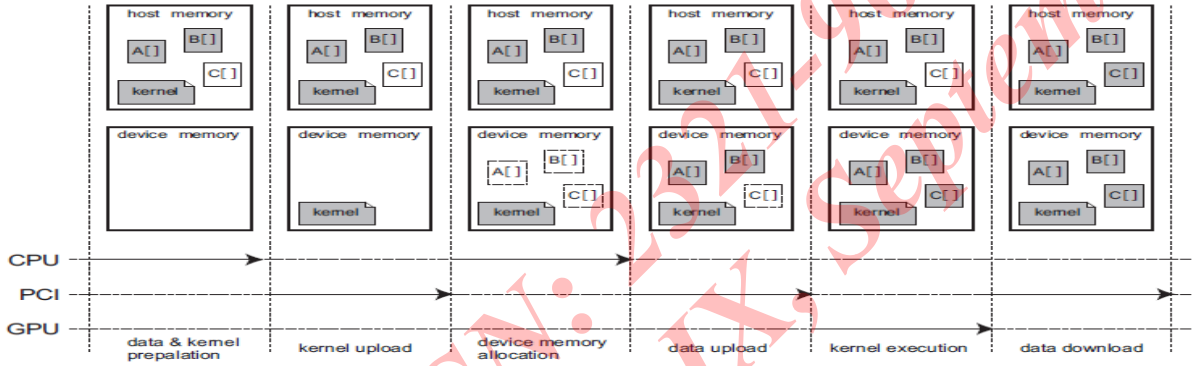


Figure 2. Example of an execution flow of matrix multiplication $A[] \times B[] = C[]$.

accelerated matrix multiplication shows in figure2 example, i.e., *A*[ ] × *B*[ ] = *C*[ ] . The GPU program code image must be loaded on the main memory. An output buffer C[ ] may be empty, while two input buffers A[ ] and B[ ] hold the valid values for computation. At the beginning , usually uploaded the kernel image. The device memory used to allocate data in it and the GPU uses this for data access. The PCI bus is used to copy the input buffers onto these allocated data spaces on the device memory. The GPU kernel starts execution, after the input data was ready on the device memory. Usually the output data are copied back onto the main memory. This is a generic flow to accelerate the program on the GPU.

IV. RESOURCE MANAGEMENT MODEL

In this part, we show a primary level model of GPU resource management, especially along with the Linux system stack, accommodating Path scale's open source driver, NVIDIA proprietary driver and Linux is open-source driver. NVIDIA can share about 85% of code between Linux and Windows, since Windows Display Driver Model(WDDM) also applicable to our model. The NVIDIA's Fermi architecture is also conceptually applicable to most of today's GPU architecture.4.1 System stack A set of GPU commands to enable the device driver and the user-space runtime engine to control

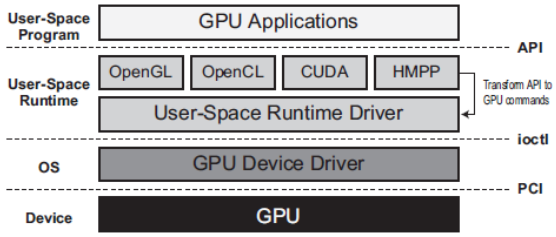


Figure 3. System stack for GPU processing.

IJRASET: ISSN: 2321-9653 Volume II Issue IX, September 2014

data copies and kernel launches is defined by GPU architecture. The device driver supply primitives for user-space programs to send GPU commands to the GPU and the user-space runtime engine issue a specific API to write user program, abstracting the low-level primitives at the device driver. The device driver and run-time engine use ioctl system call to interface between them. The system stack in our GPU resource management model is illustrates in figure 3. The runtime engine provides applications call API library functions.

4.2 GPU Channel Management

The GPU channels are managed by the device driver.

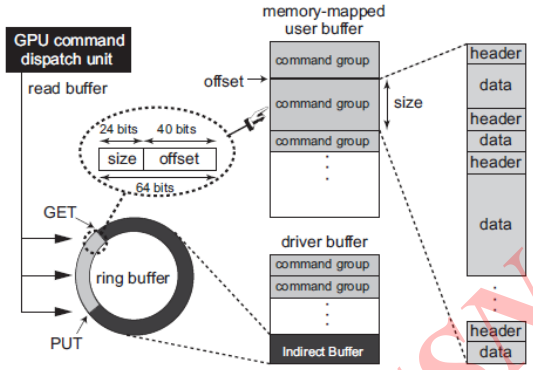


Figure 4. GPU command submissions.

An interface that bridges across the CPU and GPU contexts is GPU channel which is used to send the GPU commands from the CPU and the GPU. The support of 128 channels is present in the NVIDIA's Fermi[3] architecture for an instance. How to submit GPU commands to the GPU within a channel is illustrate in figure4. To store GPU commands the GPU channel uses two types of buffers in the operating-system address space. The device driver directly use the buffer to send specific GPU commands that control the GPU, such as initialization, channel synchronization and mode setting. The GPU command groups occur when the GPU commands are grouped into multiple units.

4.3 GPU Context Management GPU contexts consist of two register are memory-mapped I/O registers and Hardware registers which must be initialized by the device driver at the beginning. The hardware register need GPU sub-units to be read and write, but the memory-mapped I/O registers can be directly read and write by the device driver through the PCI bus. For accessing the

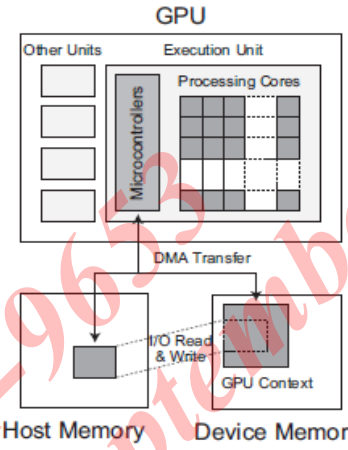


Figure 5. GPU context management model.

context values there are multiple ways. Several hardware units are provided by the GPU to transfer data among the main memory, the device memory and GPU registers in a burst manner. A conceptual model of how to manage the GPU context is illustrates in figure5. The device memory store the GPU context but the main memory store the one-time accessed data. The microcontroller also contain memory space where some accessed data by GPU context are stored. To manage such data DMA transfer is needed.

4.4 Memory Management

At least three address spaces are associated for GPU applications in memory management. Given that the CPU starts the user program and the user-space virtual memory create the user buffer on the host memory. The operating-system virtual memory copied that buffer, since the device driver must access it to transfer data onto device memory. The POSIX standard as the mmap system call is supported by memory-mapped buffer. The user program can use memory-mapped buffer quite flexibly. The user-space virtual memory, operating-system virtual memory and GPU-Kernel virtual memory are the three address spaces associated with memory management.

IJRASET: ISSN: 2321-9653 Volume IX Issue IX, September 2014

V. OPERATING SYSTEMS CHALLENGES

In this part, we examine the operating systems challenges for GPU resource management. The following discussion is based on our favourite research perspective but it does not cover the complete area of GPU resource management. The following discussion will lead to the ideas of where we are at and where to go .

5.1 GPU Scheduling

The scheduling of GPU is the most important task to hold the GPU in multi-tasking environments. The GPU kernel program are executed in first-in-first-out (FIFO) manner just because of GPU Scheduling is not done. A response time problem also caused due to the absence of GPU scheduling support. The GPU has been idle when the first launch of the high-priority task is serviced . The preceding executions of GPU contexts launched by the low-priority task after the second and third launches of the high-priority task are blocked. Due to the nature of FIFO dispatching this blocking problem appears. The solution of this problem is scheduling the GPU appropriately .

5.2 GPU Clustering

For clustering multiple GPUs is the another challenge of operating systems. To use GPUs for HPC application the GPU clustering[4] is the only way. These applications are compute-intensive as well as data-intensive. Most of the GPU clusters are hierarchial.

On-board GPU clusters: On board management of multiple GPUs may be either in the user-space runtime or the operating system.

Networked GPU clusters: The network is more challenging due to the management of multiple connected GPUs networking.

5.3 GPU Virtualization

The useful techniques widely adopted in many application domains to isolate clients in the system and make the system compositional and dependable is Virtualization[13]. The support has been provided by run-time engines, VMMs and I/O managers in the literature in GPU virtualization. The different guest operating system which are installed in VMs is the major concern for GPU virtualization. The different firmware images and assumptions are used in different guest operating systems by GPU device drivers.

5.4 GPU Device Memory Management

The user programs is allocated in the device memory spaces which are typically pinned. The allocatable memory size is limited to the device memory size. So, it is not an efficient memory management model. Virtual memory to isolate address spaces among GPU channels is supported by GPU. The functionality of this virtual memory is to expand the allocatable device memory which is utilized by the operating systems.

5.5 Coordinate with Runtime Engines[16]

The GPU command groups issued from user-space programs control the GPU operations. The user program is activated by a specific sets of GPU commands when the GPU kernel launches and data copies between the host and device memory. The types of GPU commands are issued from user-space programs does not recognize by the operating system. To obtain such information from user space program , an interface is provided to the operating system.

VI. CONCLUSION

In this paper, we have presented the numbers of challenges which are offered by operating system for GPU resource management . We have also discuss about the state of art in GPU resource management . GPU technology has high performance and energy efficient due to which it start promising in many application domains. This paper identified core challenges for operating systems research to efficiently use the GPU in multi-tasking environments and also provide some insights into their solutions. As our understanding progresses the identified list of challenges needs to expand . The additional timing issues need to address in real-time system. We believe that this paper encourage all of you to known about a grander vision of GPU technology.

IJRASET: ISSN: 2321-9653 Volume II, Issue IX, September 2014

REFERENCES

[1]. Intel. IntelMicroarchitecture Codename Sandy Bridge. http://www.intel.com/.
[2]. NVIDIA. Linux X64 (AMD64/EM64T) Display Driver. http://www.nvidia.com/.
[3]. NVIDIA. NVIDIA's Next Generation CUDA Compute Architecture:Fermi (Whitepaper). http://www.nvidia.com/content/PDF/fermi white papers/NVIDIA Fermi ComputeArchitecture Whitepaper.pdf.
[4]. Z. Fan, F. Qiu, A. Kaufman, S. Yoakum-Stove, "GPU Cluster for High Performance Computing,", *in Proc. ACM/IEEE conference onSupercomputing*, 2004.
[5]. D. Roeh, V. Kindratenko, R. Brunner, "Accelerating Cosmological Data Analysis with Graphics Processors," *in Proc. 2nd Workshop on General-Purpose Computation on Graphics Processing Units*, 2009.
[6]. J. Ferreira, J. Lobo, and J. Dias. Bayesian real-time perception algorithms on gpu. Journal of Real-Time Image Processing, 6:171–186, 2011. 10.1007/s11554-010-0156-7.

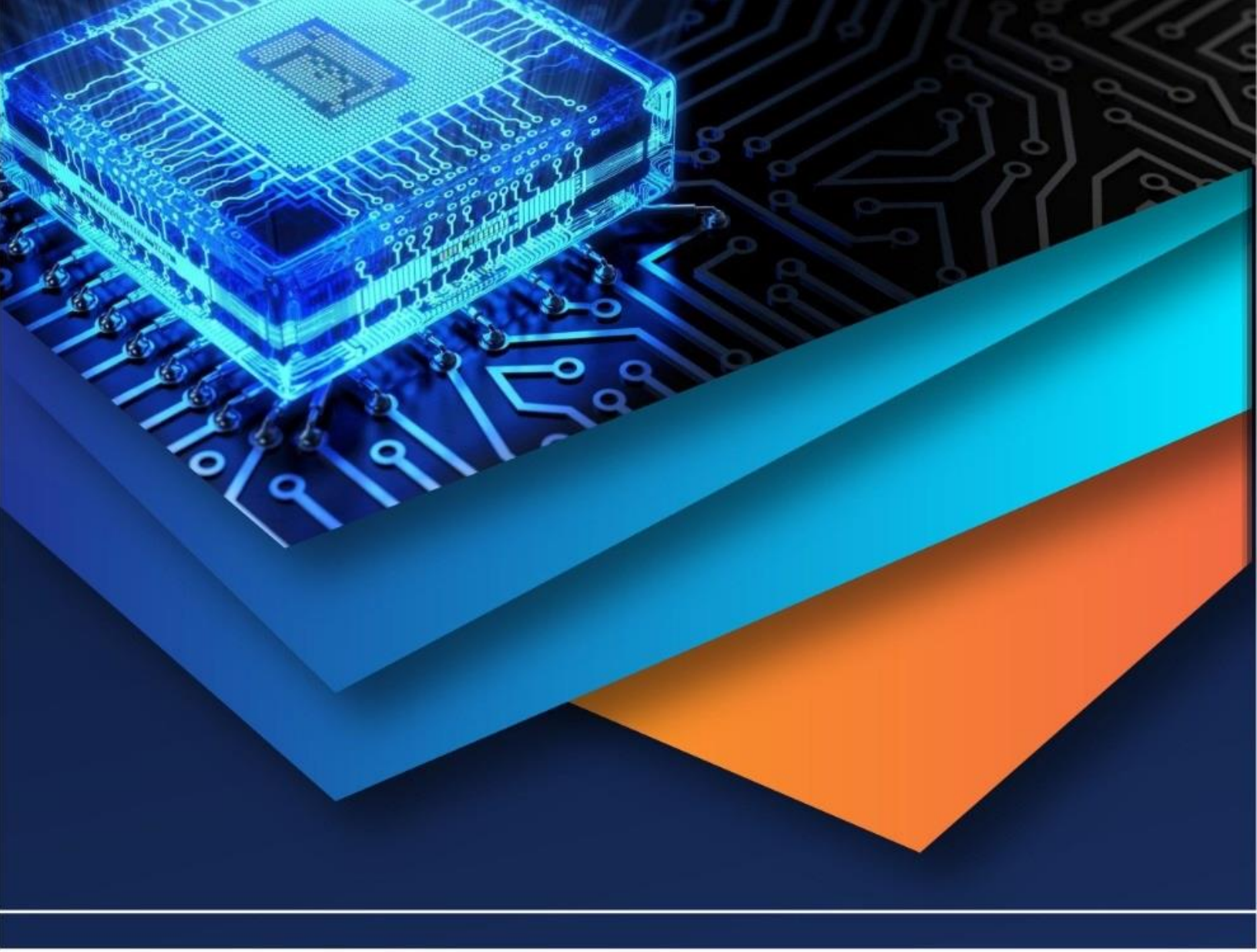
[7]. T. Kelly. Bmw self driving car: Carmaker shows off hands-free car on autobhan, 2012.

[8]. J. Markoff. Google cars drive themselves, 2010.

[9]. S. Kato, K. Lakshmanan, R. Rajkumar, and Y. Ishikawa. TimeGraph: GPU Scheduling for Real-Time Multi-Tasking Environments. In Proceedings of the USENIX Annual Technical Conference, 2011.
[10]. S. Kato, K. Lakshmanan, A.kumar,Y.Ishikawa, and R. Rajkumar. RGEM: A Responsive GPGPU Execution Model for Runtime Engines. In Proceedings of the IEEE Real-Time Systems Symposium, pages 57–66, 2011.
[11]. NVIDIA Press. NVIDIA GPUs to Be the Infotainment Centerpiece Across BMW's Next-Generation of Cars. http://pressroom.nvidia.com/easyir/customrel.do?easyirid=A0D622CE9F579F09&version=live&prid=704317&releasejsp=release_157&xhtml=true , seen on March 7th, 2012.
[12]. Top500 Supercomputing Sites. http://www.top500.org/.
[13]. M. Dowty and J. Sugeman. GPU Virtualization on VMware's Hosted I/O Architecture. *ACM SIGOPS Operating Systems Review*, 43(3):73–82, 2009.

[14]. K. Li and P. Hudak. Memory coherence in shared virtual memory systems. *ACM Transactions on Computer Systems*, 7(4):321–359, 1989.
[15]. P. Michel, J. Chestnutt, S. Kagami, K. Nishiwaki, J. Kuffner, and T. Kanade. GPU-accelerated Real-Time 3D Tracking for Humanoid Locomotion and Stair Climbing. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 463–469,2007.
[16]. M. Bautin, A. Dwarakinath, and T. Chiueh. Graphics Engine Resource Management. In *Proceedings of the Annual Multimedia Computing and Networking Conference*, 2008.
[17]. A. Gharaibeh, S. Al-Kiswany, S. Gopalakrishnan, and M. Ripeanu. A GPU Accelerated Storage System. In *Proceedings of the ACM International Symposium on High Performance Distributed Computing*, pages 167–178, 2010.

IJRASET: ISSN: 2321-9653 Volume II Issue IX, September 2014



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)