TITLE: Multitasking of Real-time GPU Computing Tasks on Mobile Platforms

Objectives

In the research proposed here I aim to address the growing multitasking needs of mobile platforms using Graphical Processing Unit (GPU) computing. Mobile devices increasingly interact with the physical world through various real-time applications. Most of these applications require performing tasks that process large blocks of data in parallel which are graphics related and/or non-graphics related.

The GPU is a specialized hardware which has hundreds of simple processors that work in parallel. Although, GPUs are historically designed to efficiently display graphics, with the introduction of GPU Computing in the last decade, GPUs also provide fast runtimes for non-graphics related data-parallel tasks. Speedups provided by GPUs are especially important for time-sensitive real-time applications. Very recently, exciting developments in the industry made GPU Computing also available on the mobile platforms. Hence, GPUs became an effective platform for running real-time mobile applications that involve graphics and/or GPU computing tasks.

As mobile devices increasingly interact with the physical world, the number of real-time applications they run also increases. Hence, the management of concurrent multiple real-time tasks on these platforms becomes more challenging. Although the recent developments made GPUs powerful platforms for running data-parallel real-time tasks on mobile devices, the current programming model of GPUs poses many challenges for multitasking of such tasks in an effective way. Hence, our ability to utilize the full power of GPUs on mobile platforms would require developing strategies that would allow these platforms effectively multitask among data-parallel real-time tasks, which I focus on in this project.

Specifically, to develop the strategies mentioned above, I design several alternative schedulers and compare them on different workloads to determine which scheduling approach is more effective for a given workload and why. To support and investigate the latest technology trend which recently made GPU Computing available on mobile settings, I target workloads that consist of real-time GPU Computing tasks. Based on my findings, I also recommend new features that, when added to the upcoming architectures, would allow better schedulers to be designed.

Description

To pursue the research objectives above, I conduct a literature review on a wide spectrum of scheduling strategies for multitasking among real-time data-parallel tasks. Based on this investigation, and by taking into account the specific characteristics of mobile real-time GPU computing workloads, I design several schedulers that are composed of combination of alternative strategies. These schedulers assign priorities to received tasks and sort them according to these priorities. Then they decide which tasks should run at a given time and how they should be processed.
The alternative strategies that the schedulers use include running the scheduling logic either on the CPU (host) or the GPU (device), not overlapping operations of different tasks or interleaving copy/execute operations and running kernels concurrently, changing the issue order dynamically or keeping it static, and using the synchronization command or atomics to determine the end of kernels in GPU schedulers.

The current programming model of GPUs poses several challenges for effectively managing workloads containing multiple concurrent data-parallel real-time tasks which typically run on mobile platforms. Some of these challenges and the techniques I use to overcome them are as follows:

(a) Historically, GPUs run one demanding task at a time: Current GPU computing research typically concentrates on high-performance computing applications performing only one demanding task at a time. Although the current architectures provide some features that allow running concurrent GPU kernels, since they are not primarily designed for running multiple real-time tasks concurrently, these features present several limitations for the purposes of this research. For instance, they do not allow the managing the scheduling of resources which is an essential part of a system that multitasks among real-time tasks. I overcome these limitations by developing several methods that are derived from the features provided by the GPU architectures that are currently available on mobile platforms, i.e. Fermi and Kepler architectures of NVIDIA. These features are related to overlapping multiple tasks (e.g. asynchronous memory copy, streams, hyper-q, concurrent kernel execution, atomics), timing individual task operations (e.g. events, “clock” device function), and running the scheduler logic on the GPU (e.g. persistent kernel, zero-copy memory, dynamic parallelism).

(b) GPUs are optimized for throughput not latency: Real-time tasks require either low latency or high throughput, or perhaps both. Understanding the needs of these two different types of tasks and scheduling them in parallel is a challenge. To remedy for this shortcoming, I use different priority assignments and performance calculations for latency and throughput-oriented tasks.

(c) GPUs lack important characteristics of real-time systems: GPUs do not provide priorities, preemption, and synchronized time across CPU-GPU. To overcome these challenges by assigning software priorities, scheduling two tasks at a time, using different time measures on CPU-GPU and a software synchronization. In addition there is a slow interface between CPU and GPU. To remedy for this shortcoming, I overlap data transfers of one task with kernel executions of another task.

Due to these challenges, very few prior studies focused on multitasking of GPU workloads with real-time tasks [Steinberger et al., 2012, Kato et al., 2011, Elliott and Anderson, 2010, Rossbach et al., 2011]. However, these studies either do not provide plausible scenarios or perform multitasking among collaborative tasks. In addition, these studies overlooked the need of scheduling different types of real-time concurrent tasks that may require low latency and/or high throughput. By addressing these challenges, this study provides support for multitasking among GPU workloads on mobile platforms which typically involve non-collaborative real-time tasks that have low latency and/or high throughput.
Results

I compare the performance of schedulers to determine which scheduling approach is more effective for a workload that may typically run on a mobile platform and why. A scheduler performance is based on a value that accumulates the slack time of each data.

Some of the important conclusions of this study include:

(a) We should use the approach that runs kernels concurrently if we have small kernels. If large kernels are used, the performance does not change on Kepler and degrades on Fermi.

Two kernels run concurrently if there are still hardware resources after all blocks of the first kernel are scheduled. This is the reason why small kernels can “truly” run concurrently but large kernels cannot do that. Hence, if large kernels are used, the performance does not change on Kepler. When large kernels are used performance is degraded on Fermi due to the “delay signal” phenomena that breaks the overlap between kernel execution and device to host copy. Since it has Hyper-Q, this phenomena does not occur on Kepler.

(b) If we have small kernels and kernel runtimes of higher-priority tasks are usually longer than those of lower-priority tasks, we should use the approach that changes the issue order dynamically to improve results of CPU schedulers running on the Fermi architecture:

Changing issue order approach, which is used in CPU schedulers, utilizes atomic operations which are expensive. Hence, use of these operations increases the runtime of large kernels and they are only advantageous for small kernels. Even if we have small kernels, changing issue order does not improve performance on Kepler, since it has Hyper-Q and the issue order does not affect the performance. In CPU schedulers, by default, the higher-priority task’s device to host copy is issued before that of lower-priority task. Hence, if runtime of higher-priority task is shorter, changing issue order would not improve performance. However, if runtime of higher-priority task is longer, to get the best overlap on Fermi, we need to switch the default issue order.

(c) Due to the limitations of the existing GPU architectures, currently we should use the approach that performs CPU scheduling instead of the one that performs GPU scheduling:

These limitations include the facts that running serial scheduler logic on a parallel GPU is not efficient; launching kernels from the GPU schedulers takes longer than launching them from the CPU schedulers; to enforce priority between tasks, the GPU schedulers cannot use the benefit of launching kernels in parallel from device; GPU schedulers require transferring scheduling messages between the host and device; since small amount of device resources are occupied by the scheduler, processing tasks take slightly longer in the GPU schedulers.

In addition to determining which scheduling approach is more effective for a given workload, I also highlight the shortcomings of current GPU architectures with regard to running workloads with multiple real-time tasks that are typically run on mobile platforms, and I recommend new features that, when added to the upcoming architectures, would allow better schedulers to be designed. These recommendations include adding features that would allow assigning hardware
priorities to blocks, performing preemptions, reserving a specific streaming multiprocessor for a particular block, providing programmable scheduling, having a common time concept between the host and device, disclosing the type of queue into which events would be placed, flushing the zero-copy memory, determining the end of concurrent kernels, waiting on only the work generated by the synchronizing block, having atomics across the host and the device, and achieving a faster launch of kernels from the device.

References


