Motivation and Goal

Mobile devices increasingly interact with the physical world through various real-time applications which need processing large blocks of data in parallel, e.g.:

- augmented reality app that overlays information on to the physical world where a tablet is pointed at
- face recognition capability provided by a wearable computing device
- fingerprint unlocking functionality of a smartphone

Graphical Processing Units (GPUs) are historically designed for graphics which involve data-parallel computation.

With the advent of GPU computing, GPUs are also used for running non-graphics related data-parallel tasks.

Speedups provided by GPUs are especially important for time-sensitive real-time applications.

Recently, GPU computing also became 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.

However, current programming model of GPUs poses many challenges for multitasking of real-time tasks that typically run concurrently on mobile platforms.

The goal of this study is to address these challenges by:

- designing several alternative schedulers for multitasking of real-time GPU computing tasks that typically run on mobile devices
- comparing these schedulers on different workloads to determine which scheduling approach is more effective for a given workload
- recommend new features that, when added to the upcoming architectures, would allow better schedulers to be designed

Methods and Implementation

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, such as:

- running the scheduling logic either on the CPU or the GPU
- not overlapping operations of different tasks or interleaving copy/execute operations and running kernels concurrently
- changing the issue order dynamically or keeping it static
- using the synchronization command or atomics to determine the end of kernels in GPU schedulers

Some of the research challenges and the techniques I use to overcome them are as follows:

- Historically, GPUs run one demanding task at a time: I develop 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. Although these features are not primarily designed for multitasking of real-time tasks (e.g. concurrent kernel execution), they constitute a basis for such purposes.
- GPUs are optimized for throughput not latency: I use different priority assignments and performance calculations for latency and throughput-oriented tasks.
- GPUs lack important characteristics of real-time systems (i.e. they do not provide priorities, preemption, synchronized time across CPU-GPU, fast interface between CPU-GPU): I assign software priorities, schedule two tasks at a time, use different time measures on CPU/GPU, and overlap data transfers of one task with kernel executions of another task.

Results and Discussion

I compare the performance of schedulers to determine which scheduling approach is more effective for a GPU workload that may typically run on a mobile platforms.

Important conclusions of this study include:

- 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.
- 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.
- Due to the limitations of the existing GPUs, currently we should use the approach that performs CPU scheduling instead of the one that performs GPU scheduling.

I also highlight the shortcomings of current GPU architectures with regard to multitasking of real-time tasks and recommend new features that would support better schedulers to be designed.

These recommendations include adding functionalities that would allow:

- assigning hardware priorities to blocks
- performing preemptions
- reserving compute resources
- providing programmable scheduling
- having a common time concept between the CPU-GPU
- determining the end of concurrent kernels
- having atomics across the CPU-GPU
- achieving a faster launch of kernels from the GPU