Advanced OS Report#12

37186305
1st year in Master Course, Dep. Aeronautics & Astronautics, School of Engineering
Hidehisa Arai

2019 年 1 月 17 日

1 Difference between PTask and TimeGraph, Gdev

1.1 Difference according to the problem consciousness

In this subsection, difference according to underlying problem consciousness between these researches will be explained.
First, there are three components in the problem consciousness behind the PTask.

- OS abstraction related to GPU usage is not sufficient enough to conduct interactive processes using GPU. To use GPU for interactive usage, high response frequency and low latency is required, but processes using GPU sometimes suffer from the double buffering problem which leads to high latency.
- GPU systems are lack of fairness and isolation, which means that, GPU bound tasks and CPU bound tasks are mutually influenced by each other.This means that when using GPU, the activity of CPU bound tasks will be suffered from the GPU activity while memory and CPU usage of host is not so active. CPU activities will also interfere GPU throughput vice versa. This happens because the scheduling policy is hidden by the device driver and the policy is sometimes so primitive that it can cause a performance problem.
- Current ioctl based API is not sophisticated enough, and being a burden for the programmers to handle it. To handle GPU effectively, static knowledge about OS level issues are required, and this can be a hurdle for the programmers.

To summarize, current API for GPU usage is not sophisticated enough and new API to treat GPU as a first class computing resource is required.
The problem consciousness behind the Gdev and TimeGraph are as follows,

- The system software support for GPU usage is not well-designed to integrate GPU to general purpose multi tasking processes which requires the GPU support for parallel computing.
- In the current system, GPU usage are limited to the user space, and OS cannot use the API for GPU.
- Memory usage of GPU cannot exceed the physical memory capacity of the GPU.
- Scheduling for GPU is not sophisticated enough, partially because GPU is command driven and cannot be stopped while it's working, but this problem prevent GPU from being used for real time usage.

These problem consciousness will also be summarized as follows, that new API sets to treat GPU as a first class computing resource is required. So the basic problem consciousness are similar, but in detail there are some differences.
First, the authors of PTask concerned about the double buffering problem as serious bottleneck while the authors of Gdev did not mentioned about that problem.
Second, the authors of Gdev mentioned about the capacity of GPU memory while the authors of PTask didn't.

1.2 Difference according to the solution

In this subsection, difference according to the suggested solution between these researches will be explained.
PTask suggested graph based API sets. In detail, PTask supports a data flow programming model in which individual tasks are assembled into a acyclic computational graph. The programmers need to define the ptask which corresponds to the node(vertices) of the graph and the channel which corresponds to the edge of the graph. Each ptasks is a step in the calculation flow, and each channels is a data flow in the calculation. Programmers need to connect the channel to the ports which are the gates attached to the ptasks. Finally programmers need to define the data entities called data blocks and meta data for the calculation called templates.
The characteristics of this API sets are as follows,

- Bring GPUs under the purview of a single resource manager, allowing the entity to provide meaningful guarantees for fairness and isolation.
- Provide a programming model that simplifies the code development for the accelerators by abstracting away codes that manages devices, performs I/O, and deals with disjoint memory spaces and freeing the programmers to focus on the algorithm or data flows.
- Provide a programming environment that allow both modular and fast.

TimeGraph and Gdev provide a scheduler and API sets to treat GPUs as first class computational resources. In detail, they implemented a wrapper API and enabled the OS itself to use GPUs. They also provided shared memory functionality for GPU to enable GPU contexts to allocate memory exceeding the physical size of device memory. They also introduced the inter-process communication for memory management and scheduling for GPU virtualization.
The largest difference between PTask and (Gdev, TimeGraph) is the data flow programming model of the PTask to create a static computational graph to optimize the calculation. Both of the researches introduce the scheduling mechanism and sort of shared memory inside the GPU.

2 Main solution of PTask for data-flow programming models

The basics of the PTask's graph oriented data-flow programming models are stated above, so in this section, more detailed things will be explained.

- ptask is an analogous to the traditional OS abstraction, but a ptask runs on a GPU or other accelerators, so

it is different from processes.A ptask has a list of input and output resources, which is an analogous to the POSIX stdin, stdout, stderr, that can be bound to the ports.

- A port is an object in the kernel namespace that can be bound to ptask input and output resources.Ports has three subtypes, InputPort, OutputPort, and StickyPort. The former two correspond to stdin and stdout of POSIX, and the latter represents an input which retain its value across multiple invocations of ptasks.
- A channel is an analogous to the POSIX pipe, it connects a port to another port, or to other data sources and sinks in the system such as I/O buses, files, and so on.
- A graph is collection of ptasks whose ports are connected by channels. Multiple graph may be created and executed independently, with the PTask runtime being responsible for scheduling them fairly.
- A datablock represents a unit of data flow along an edge in a graph. A template provides meta-data describing datablocks.

3 What is improved by PTask and how much?

They evaluated their API sets with some benchmarks.First, they evaluated with gestural inference, compared with other three methods.

The configuration of other three methods are, hand-coded, which directly manipulate the user mode camera component of the camera driver to offload work to GPU, in order to eliminate unnecessary copy of the data.

The configuration for the second method is to connect four processes with pipe. This method is modular, but with this method unnecessary movement at the boundary of kernel space and GPU incurs, and also inter-process communication is needed which leads to extra overhead. The configuration for the third one is almost the same as the second one, but all the procedures are conducted in a single process, which means that unnecessary IPC is eliminated.

Among these three configurations, the first one, hand-code scores the best performance while the second one, pipe is the easiest to implement. Compared with these configurations, PTask scores the best. It got 11.6% higher throughput compared with the hand-code and 7% low latency, and 22.5% low CPU utility. This happens because PTask eliminate unnecessary data movement at the boundary of the kernel and GPU.

Second, they checked that their GPU scheduling method to allow scheduler to change the priority of the GPU eliminated the problem stated above.