Data Transfer Matters in Low-Latency GPU Computing

Yusuke Fujii, Takuya Azumi, and Nobuhiko Nishio
College of Information Science and Engineering
Ritsumeikan University

Shinpei Kato and Masato Edahiro
Department of Information Science
Nagoya University

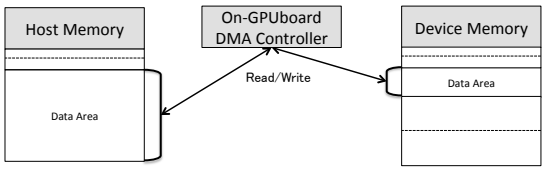

Figure 1. Standard DMA.

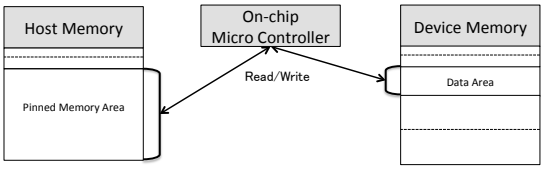

Figure 2. Microcontroller-based data transfer.

***Abstract*—**

***Keywords*-Low Latency; Data Transfer; GPUs**

I. Data Transfer Methods

In this section, we investigate data transfer methods for the GPU. A standard data transfer method uses hardware DMA engines integrated on the GPU, while direct data read and write accesses to the GPU device memory are allowed through PCIe BARs. We can also use microcontrollers integrated on the GPU to send and receive data across the host and the device memory. Unfortunately, only a limited piece of these schemes has been studied in the literature. Our investigation and open implementations of these schemes provide a better understanding of data transfer mechanisms for the GPU. Note that we restrict our attention to the NVIDIA GPU architecture, but the concepts of data transfer methods introduced in this paper are mostly applicable to PCIe-connected compute devices.

A. Standard DMA

The most typical method for GPU data transfers is to use standard DMA engines integrated on the GPU. There are two types of such DMA engines for synchronous and asynchronous data transfer operations respectively. We focus on the synchronous DMA engines, which always operate in a sequential fashion with compute engines.

Figure 1 shows a concept of this standard DMA method. To perform this DMA, we write *GPU commands* to an on-board DMA engine. Upon a request of GPU commands, the DMA engine transfers a specified data set between the host and the device memory. Once a DMA transfer starts, it is non-preemptive. This method is often the most effective to transfer a large size of data. Details of the GPU DMA mechanism can be found in previous work [2], [3].

B. Microcontroller-based Data Transfer

The GPU provides on-board microcontrollers to control GPU functional units (compute, DMA, power, temperature, encode, decode, etc.). Albeit tiny hardware, these microcontrollers are available for GPU resource management beyond just controlling the functional units. Each microcontroller supports special instructions to transfer data in the data sections to and from the host or the device memory. The data transfer is offloaded to the microcontroller, *i.e.*, DMA, but is controlled by the microcontroller itself. Leveraging this mechanism, we can provide data communications between the host and the device memory.

Figure 2 shows a concept of this microcontroller-based data transfer method. Since there is no data path to directly copy data between the host and the device memory using a microcontroller, each data transfer needs to take two hops: (i) the host memory and a microcontroller and (ii) the device memory and a microcontroller. This is non-trivial overhead but the handling of this DMA is very light-weight as compared to the standard DMA method.

C. Memory-mapped read and write

The aforementioned two methods are based on DMA functions. DMA is usually high-throughput but it inevitably incurs overhead in the setup. A small size of data transfers may encounter severe latency problems due to this overhead. One of good examples can be found in the plasma control system [1]. If low-latency is required, direct read and write accesses are more appropriate than hardware-based DMA. Since the GPU as a PCIe-connected device provides memory-mapped regions upon the PCI address space, the CPU can directly access the device memory without using bounce buffers on the host memory.

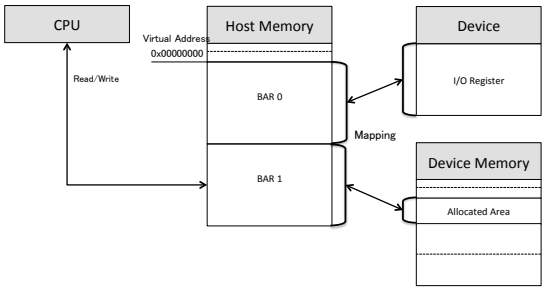

Figure 3. Memory-mapped read and write.

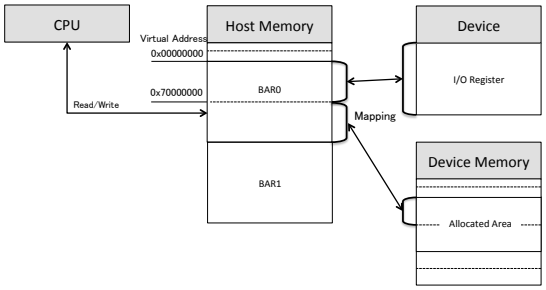

Figure 4. Memory-window read and write.

Figure 3 shows a concept of this memory-mapped read and write method. NVIDIA GPUs as well as most other PCIe devices expose base address registers (BARs) to the system, through which the CPU can access specific areas of the device memory. There are several BARs depending on the device. NVIDIA GPUs typically provide six BARs:

BAR0 Memory-mapped I/O (MMIO) registers.
BAR1 Device memory aperture (windows).
BAR2 I/O port or complementary space of BAR1.
BAR3 Same as BAR2.
BAR5 I/O port.
BAR6 PCI ROM.

Often the BAR0 is used to access registers of the GPU while the BAR1 makes the device memory visible to the CPU. This direct read and write method is pretty simple in that all we have to do is to configure the page table of the GPU.

D. Memory-window read and write

REFERENCES

[1] S. Kato, J. Aumiller, and S. Brandt. Zero-Copy I/O Processing for Low-Latency GPU Computing. In *Proc. of the IEEE/ACM International Conference on Cyber-Physical Systems*, 2013 (to appear).

[2] S. Kato, K. Lakshmanan, R. Rajkumar, and Y. Ishikawa. TimeGraph: GPU Scheduling for Real-Time Multi-Tasking Environments. In *Proc. of the USENIX Annual Technical Conference*, 2011.

[3] S. Kato, M. McThrow, C. Maltzahn, and S. Brandt. Gdev: First-Class GPU Resource Management in the Operating System. In *Proc. of the USENIX Annual Technical Conference*, 2012.