Overcoming Memory Bottleneck with Near-Data GPU Acceleration

Hanjae Lee, Eui-Young Chung
Department of Electrical and Electronic Engineering, Yonsei University
50 Yonsei-ro, Seodaemun-gu, Seoul 03722, Republic of Korea
E-mail: tmxk37@dtl.yonsei.ac.kr, eychung@yonsei.ac.kr

Abstract

High performance computing is one of the most interesting topic in modern research. General Purpose Graphic Processing Units (GPGPU) is widely used to accelerate compute-intensive workloads, however limited main memory bandwidth remains a critical issue. In many workloads, huge fraction of GPU running time is spent for copying data from main memory. Based on this point, we propose in-memory GPU architecture to improve overall performance through scheduling GPU operations.

Keywords: Processing In Memory, General Purpose Graphic Processing Unit, Memory Bandwidth.

1. Introduction

Recently, memory wall crisis is emerging problem on high performance computing system. Accordingly, Processing in Memory (PIM) is expected to be the viable solution for the problem. There have been many studies investigating PIM, some research focus PIM on main memory, and some others investigate on storage level [1] or graphics memory [2]. PIM methodologies provides highly utilized memory bandwidth so that the host processor would be able to allocate its resources for higher priority tasks.

In this paper, we propose in-memory GPU architecture to optimize GPU operation time. Our goal in this work is to investigate how much the GPU performance can be improved through processing in memory.

2. Motivation

Figure 1 shows proportions of GPU kernel time and memory copy latency in various workloads. Basically to run GPU kernel, its required data need to be stored in graphics memory. After the kernel operation ends, the data may remain on the GPU and only the resulting part would be moved from graphics memory to main memory.

In some workloads, however, the host processor constantly checks the status of whole data, and may need to use the data in real time process, for example big-data management. In this cases, transferring data between main memory and graphics memory may be critical on the overall system performance. Therefore, reducing the memory transfer duration by processing GPU kernels in main memory is reasonable in such cases.

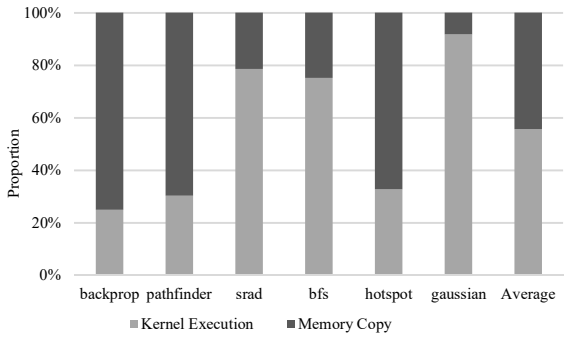

Figure 1. Proportions of GPU operations

3. in-Memory GPU Architecture

Our proposed architecture has two GPU models, one is conventional GPGPU model and the other is In-Memory GPU (IMGPU) which is directly connected to main memory. In our system, the main memory is used as graphics memory for IMGPU. The overall block diagram is shown in following figure 2.

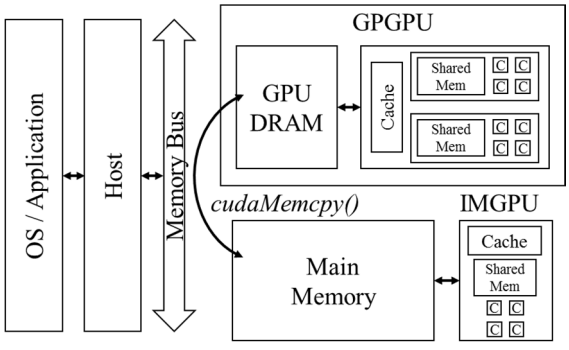

Figure 2. Block diagram for IMGPU architecture

The number of cores of baseline GPGPU is set to 32. For IMGPU, Pattnaik et al. [2] considered thermal feasibility for their PIM based GPU scheduling. According to their study, we configured the number of cores in the IMGPU to four, in order to assume even worse conditions.

Since profiling GPU kernel or memory operations have been well researched in many previous works, we assume that we know the running time of each kernel operation before executing. The main idea for scheduling GPU operation is represented in figure 3. If the sum of kernel time and the sum of durations for kernel dependent memory copy is greater than the kernel time on PIM, the kernel is scheduled to PIM.

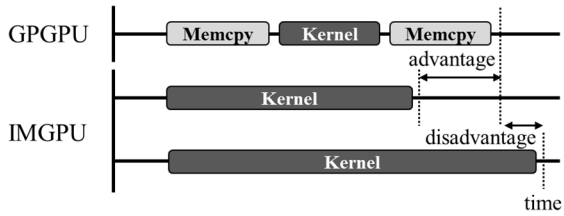

Figure 3. Concept diagram for the proposed GPU scheduling scheme

4. Experiment

We used gem5-gpu [3], which is cycle-accurate full system simulator. The simulator contains CPU, GPU and memory models. The following table provides our simulating GPU resource configurations for overall architecture.

Table 1: Simulator configurations

GPGPU	32 shader cores, 1.4GHz clock rate, 32 SIMT width
IMGPU	4 shader cores, 1.4GHz clock rate, 32 SIMT width
Core Resources	1536 threads, 48KB shared memory, 32K registers, Maxwell architecture
Caches	64KB 4way L1 I/D cache, 1MB L2 cache, 128B block size

Gem5-gpu provides Ruby cache hierarchy for memory model, and only one DRAM model is shared between the CPU and the GPU. In this work, we assume that main memory and GPU memory use the same type of DRAM, in this case, DDR3. Therefore, difference on the kernel time in two GPU system depends on configurations for computing resource. In future studies, we plan to evaluate our architecture with various types of DRAM configurations such as GDDR5 and HBM (High Bandwidth Memory).

We chose Rodinia benchmark suite [4] to evaluate our architecture. The Rodinia is designed for high performance computing system with NVIDIA CUDA implementation. Workloads were selected as having various characteristics.

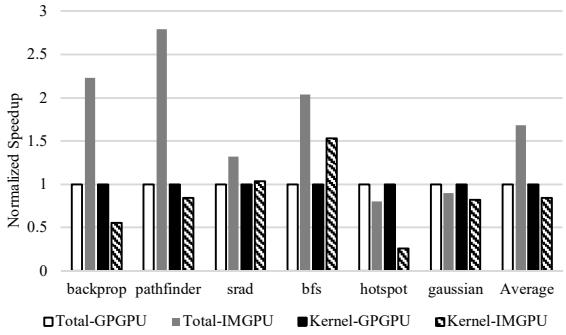

Figure 4. Normalized speedup for each workloads

Figure 4 shows the experiment results performed in each two GPU models. It contains speedups on total execution time and the only kernel time of GPU operations. Each result of conventional GPGPU system is fixed at one, and the following shows amount of increment of system performance with our architecture in the corresponding workload.

The first two workloads, backprop and pathfinder are ideal cases which spend most of the running time as memory copy, therefore using PIM is advantageous even if it takes longer to run kernel operation.

Gaussian is a typical failure case for our proposed architecture, which is compute-intensive benchmark that most of the running time is kernel time. In hotspot benchmark, on the other hand, it also failed even though its proportion of memory copy is almost same as backprop or pathfinder. In this case, performance degradation of kernel operation was more critical than the advantage on memory issue.

In case of srad and bfs, kernel time was measured to be the same or rather low, which shows GPU scalability does not guarantee linear performance improvements due to low utilization of cores as in the study of [5]. Pattnaik et al. [2] has also conducted PIM study within GPU memory with these characteristics.

5. Conclusion

In this paper, we proposed in-memory GPU architecture to improve overall GPU performance. To overcome memory wall crisis, our PIM method schedules GPU kernels to run in two separate GPUs, conventional GPGPU and our in-memory GPU. According to our experiments with Rodinia CUDA benchmarks, GPU operations can be improved with average 1.7 times speedup on the ideal condition. We will further investigate our method in practical conditions with dynamic profiling of GPU operations and various types of DRAM models.

Acknowledgement

This work was supported by the ICT R&D program of MSIP/IITP. [2016(R7177-16-0233), Development of Application Program Optimization Tools for High Performance Computing Systems]

References

[1] B Gu, et al., “Biscuit: A Framework for Near-Data Processing of Big Data Workloads”, ACM/IEEE International Symposium on Computer Architecture (ISCA), pp 153-165, Jun 2016.

[2] A Pattnaik, et al., “Scheduling Techniques for GPU Architectures with Processing-In-Memory Capabilities”, International Conference on Parallel Architecture and Compilation Techniques (PACT), pp 32-44, Sep 2016.

[3] J Power, et al., “gem5-gpu: A Heterogeneous CPU-GPU Simulator,” IEEE Computer Architecture Letters, pp 34-36, Jan 2014.

[4] S. Che, et al., Rodinia: A Benchmark Suite for Heterogeneous Computing. In Proceedings of the IEEE International Symposium on Workload Characterization (IISWC), pp. 44-54, Oct. 2009.

[5] K Choi, et al., “Study of GPU Scalability”, IEEE International Symposium on Consumer Electronics (ISCE), Jun 2014