Turkish Journal of Computer and Mathematics Education Vol.12 No.13 (2021), 1920-1923

Research Article

Data transfer optimization in CPU/GPGPU Communication

Siddheshwar V. Patil[a], Dinesh B. Kulkarni[b]

[a] Research Scholar, Department Computer Science and Engineering, Walchand College of Engineering, Sangli (Assistant Professor, Department of CSE, ADCET, Ashta)
[b]Professor, Department of Information Technology, Walchand College of Engineering, Sangli

Article History: Received: 11 January 2021; Revised: 12 February 2021; Accepted: 27 March 2021; Published online: 4 June 2021

Abstract: The objective of this paper is to make data transfers efficient. The performance characteristics of various data transfer methods in GPGPU computing are presented in this paper. In GPGPU computing, for the data transfer between the host (CPU) and the device (GPGPU), the performance in terms of bandwidth requirement and latency is discussed for the pageable and pinned memory mechanism. The result shows that the data transfer latency gets reduced in pinned memory mechanisms as compared to pageable memory mechanisms. It is also seen that there is significant optimization in the results by concurrently running data transfer among the host - device and kernel execution.

Keywords: Data Transfer, General purpose graphics processing units, Performance, Bandwidth, Latency, Compute Unified Device Architecture, Streams

1. Introduction

Now a day, General purpose graphics processing units (GPGPUs) are getting attractive due to their many-core computation capability [1]. GPGPUs are having thousands of processing cores which give a performance in Teraflops. Due to current advancements in programming models and hardware architectures, they are becoming more popular for any problem-solving environment. So, the compute-intensive and data-parallel applications are showing noteworthy performance improvements because of the GPGPU. A major application of GPGPU is supercomputing and the world's top supercomputers are listed in [2]. Other application includes graph processing [3], autonomous driving [4], healthcare [5], wireless routing [6], etc. In such applications, there is a need to further develop GPGPU technology to improve the performance of compute-intensive and data-parallel applications. The related programming certainly needs deserving data transfers among the device (GPGPU) and the host (CPU). The latency obtained while achieving data transfer will be the performance breaker in GPGPU based application domains. In the state of art literature, the latency and performance issues are addressed. Consider the computing kernel offloaded onto the GPGPU. In this case, the latency and performance are conquered by the compiler and hardware technology while the data transfer optimization is complimented by the system software because of the PCI devices' constraints. For example, the bandwidth for data transfer among the device memory and the GPGPU is greater than the bandwidth required for host-device data transfer. So, the data transfer among the host - GPGPU memory will be a bottleneck for overall application performance.

The data transfer job is also affected by the computational workload at the CPU side at the same time the computing kernels isolated at the GPGPU side. Thus, the data transfer became an important issue to achieve low latency GPGPU computations.

The rest part of the paper is organized as follows, a literature review is discussed in section-2, and section-3 represents methodology and experimental work. Section-4 gives the conclusion.

2.Literature Review

In many GPGPU applications, the data transfer rate is addressed. In the plasma control applications mentioned in [7], zero-copy techniques to reduce data transfer latency were showcased. The author discussed that the transfer methods such as DMA don't meet the latency condition. Numerous zero-copy data transfer approaches were presented by the author; some are memory-mapped read and writes. However, such kind of work has considered the small size of data transfer. In the Gdev project, Data I/O methods and hardware-based DMA were concisely presented [8]. It has been shown that, for large data, the hardware-based DMA methods are used.

Proper scheduling of GPGPU data transfer methods will help to increase the performance of GPGPU computing [9], [10]. This work elaborates preemption fact making for DMA non-preemptive transfer in GPGPU; however underlying architecture relies on the proprietary software's. While the open-source implementation is provided to showcase the GPGPU data transfer methods. It has been observed that hardware-based DMA transfer methods are unsuitable if data size and workloads are considered because of the preemptive I/O read and write mechanisms, while microcontroller-based techniques are partially preemptive

3.Methodology and experimental wor

For the work organized in this paper, it is assumed that for GPGPU programming, a parallel computing platform, Compute Unified Device Architecture (CUDA) [11, 12] is used. A kernel is a unit of code that is executed onto the GPGPU. The kernel can have multiple threads which are helping to run the code in parallel.

The program which can be executed on GPGPU has the following stages: (i) allocate device memory, (ii) copying of host data to device memory, (iii) kernel execution at GPGPU, (iv) device to host copying of output data, and finally (v) free the memory allocated device.

3.1.Data transfer time measurement using nvprof

The time spent in data transfer is measured by recording the CUDA event before the data transfer and after the data transfer [13]. The CUDA event cudaEventElapsedTime () is used for the same. From CUDA 5 onwards, there is a feature of CUDA profiler which can be used to measure the elapsed time. The CUDA toolkit includes nvprof, the command line CUDA profiler.

3.2.Data transfer minimization

To decide whether the code can be executed on CPU or GPGPU, we cannot measure only the time required to execute any code on GPGPU or CPU, data transfer cost across PCI-e bus is also considered. The CUDA's model practices both the CPU and GPGPU due to its heterogeneous programming. Depending upon the requirement, the code can be ported to CUDA's kernel. Sometimes, the data transfer dominates the overall execution time. It is necessary to have labels on the time required on data transfer individually from the time required in the kernel execution using the command line profiler tools. If more parts from the code will be ported, it will help to reduce intermediate transfer and reduction in total execution time [13].

3.3.Pinned host memory

By default the data allocation on the host side is pageable. The GPGPU is not accessing data from pageable host memory directly. While transferring data to device memory from pageable host memory, a temporary page-locked host array is allocated by the CUDA driver. It is called pinned memory. So, the data copy will take place from host to pinned array then afterward transfer's to the device from a pinned array which is shown in figure I. From figure I, it is seen that, while transferring data from device to host, pinned memory is used as a staging area. The data transfer cost among pageable host arrays and pinned host arrays can be reduced by memory allocation of host arrays in pinned memory directly. Using cudaHostAlloc () or cudaMallocHost () function pinned host memory CUDA can be allocated in CUDA, and freed using cudaFreeHost (). Sometimes, there are chances of failing pinned memory allocation. A proper error checking mechanism should be used for the same.

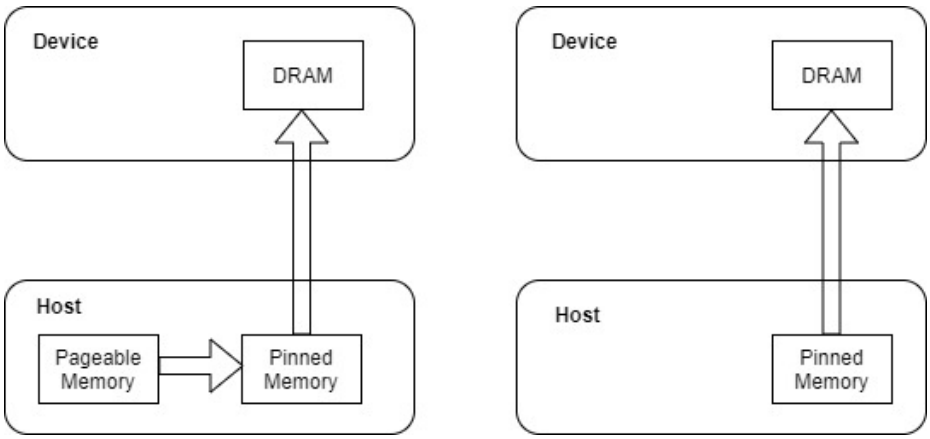

Figure I: a) Pageable data transfer b) Pinned data transfer

The data transfer rate is dependent on the workstation type as well as the GPGPU. In this work, a workstation which has an Intel(R) Xeon(R) 4112 CPU @ 2.60GHz CPU (16 Cores) and an NVIDIA V100 (2 Tesla V100 GPUs, 128 GB memory per Node, PCIe-16GB) GPGPU is used. Running of the bandwidth test program to transfer the data of size 32MB produces the sample result shown in figure-II. Similar results for other data size transfers from host to device and vice versa are shown in Table-I. It is seen that the pinned data transfer rate is

higher than the pageable data transfer rate.

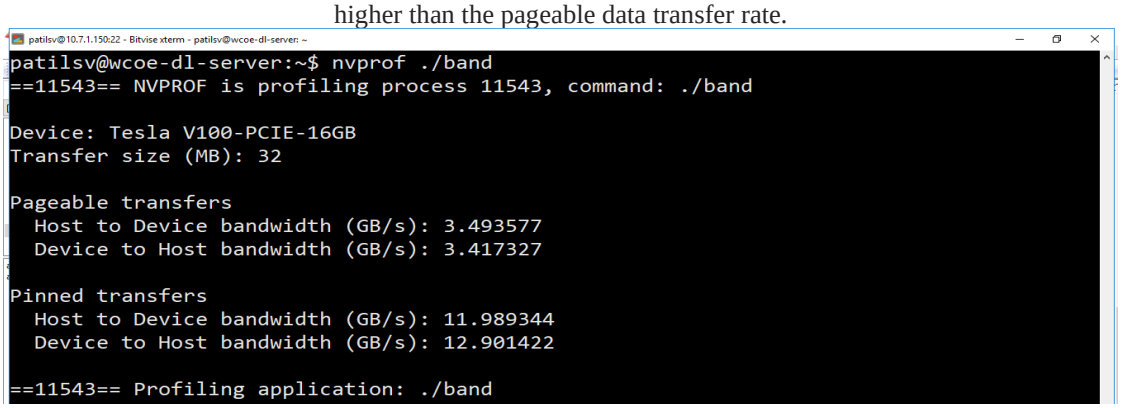

Figure II: Pageable and pinned memory results

Data Transfer size	**Pageable transfers (GB/s)**		**Pinned transfers (GB/s)**	
	HtoD Data rate	**DtoH Data rate**	**HtoD Data rate**	**DtoH Data rate**
64MB	3.58	3.37	12.15	12.91
128MB	3.51	3.14	12.20	9.94
256MB	2.20	1.86	12.28	10.03
512MB	1.57	2.23	12.33	10.21
1024MB	3.58	1.64	12.34	13.11

Table I - Pageable and pinned memory comparative results

There should not be any over-allocation of pinned memory. Making it so will reduce the total performance of the system since it shrinks the volume of physical memory existing for the OS and also for the other programs.

3.4.Overlap data transfers in CUDA

In CUDA, overlapping of data transfer along with computation at device side, and sometimes other data transfer amid host and device is possible with help of CUDA streams. CUDA *streams* are the sequence of operations that executes at the device in the order they are allotted by the host. The operations in diverse streams can be incorporated and they can concurrently execute. The data transfer process will be overlapped with kernel execution and they should occur in *diverse, non-default* streams. In case of data transfer, host memory should be pinned memory.

In this work, the host code is modified to use multiple streams and overlapping. The array with size N is broken into chunks of stream size elements. As the kernel operation performs individually on the elements, every chunk will be processed autonomously. The number of streams is calculated as, nstreams=N/stream size. So, the sample performance result for 16MB data size for the sequential and asynchronous data transfer is shown in figure III. Similar results for other data sizes are shown in Table-II. It is observed that there is significant optimization in the time results when the overlap of the data transfers and kernel execution.

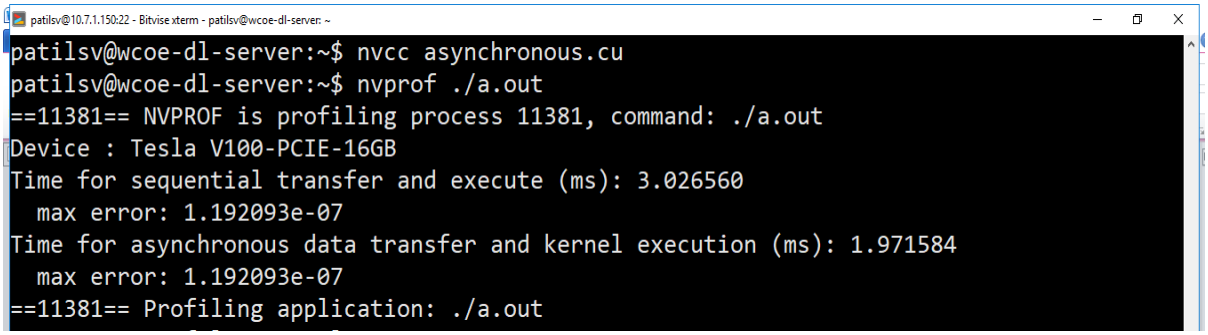

Figure III: Overlapping of kernel execution and data transfers

Data Transfer size	**Sequential transfer and Kernel execution time (ms)**	**Asynchronous transfer and kernel execution time (ms)**
64MB	12.96	10.20
128MB	26.05	20.38
256MB	52.19	40.72
512MB	91.72	60.43
1024MB	208.40	163.23

Table II - Overlapping of kernel execution and data transfers

4.Conclusion

The data transfer mechanisms are discussed in this paper. It is summarized that the data transfer among host-device should be reduced whenever possible. The host-to-device data transfer gives higher bandwidth with the help of pinned memory mechanism. The average latency gets reduced to 35% with the pinned memory mechanism as compared to the pageable memory mechanism. Many smaller transfers can be batched into one large transfer can help to get better results since it eliminates the pre-transfer overhead. It also shows significant optimization in the results by concurrently running data transfer among the host - device and kernel execution.

References

1. Zahaf, Houssam-Eddine, and Giuseppe Lipari., "Design and analysis of programming platform for accelerated GPU-like architectures" In Proceedings of the 28th International Conference on Real-Time Networks and Systems, pp. 1-10. 2020.
2. Top500 Supercomputing Sites, http://www.top500.org/.
3. Shi, Xuanhua, Zhigao Zheng, Yongluan Zhou, Hai Jin, Ligang He, Bo Liu, and Qiang-Sheng Hua., "Graph processing on GPUs: A survey" ACM Computing Surveys (CSUR) 50, no. 6 (2018): 1-35.
4. M. McNaughton, C. Urmson, J. Dolan, and J.-W. Lee, "Motion Planning for Autonomous Driving with a Conformal Spatiotemporal Lattice," in Proc. of the IEE International Conference on Robotics and Automation, 2011, pp. 4889–4895
5. Rao, Naseem, and Safdar Tanweer., "Performance Analysis of Healthcare data and its Implementation on NVIDIA GPU using CUDA-C" Journal of Drug Delivery and Therapeutics 9, no. 1-s (2019): 361-363.
6. S. Hand, K. Jang, K. Park, and S. Moon, "PacketShader: a GPU-accelerated software router," in Proc. of ACM SIGCOMM, 2010.
7. 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.
8. 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.
9. C. Basaran and K.-D. Kang, "Supporting Preemptive Task Executions and Memory Copies in GPGPUs," in Proc. of the Euromicro Conference on Real-Time Systems, 2012, pp. 287–296.
10. S. Kato, K. Lakshmanan, A. Kumar, M. Kelkar, Y. Ishikawa, and R. Rajkumar, "RGEM: A Responsive GPGPU Execution Model for Runtime Engines," in Proc. of the IEEE Real-Time Systems Symposium
11. NVIDIA, "NVIDIA's next-generation CUDA computer architecture: Kepler GK110," http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf, 2012.
12. NVIDIA, "CUDA Documents," http://docs.nvidia.com/cuda/, 2013.
13. Fujii, Y., Azumi, T., Nishio, N., Kato, S., & Edahiro, M. (2013, December), "Data transfer matters for GPU computing" In 2013 International Conference on Parallel and Distributed Systems (pp. 275-282). IEEE.