Overview and Comparison of OpenCL and CUDA Technology for GPGPU

Ching-Lung Su, Po-Yu Chen, Chun-Chieh Lan, Long-Sheng Huang, and Kuo-Hsuan Wu
Department of Electronic Engineering
National Yunlin University of Science and Technology
Yunlin, Taiwan, R.O.C.

***Abstract*—GPU (Graphics Processing Unit) has a great impact on computing field. To enhance the performance of computing systems, researchers and developers use the parallel computing architecture of GPU. On the other hand, to reduce the development time of new products, two programming models are included in GPU, which are OpenCL (Open Computing Language) and CUDA (Compute Unified Device Architecture). The benefit of involving the two programming models in GPU is that researchers and developers don't have to understand OpenGL, DirectX or other program design, but can use GPU through simple programming language. OpenCL is an open standard API, which has the advantage of cross-platform. CUDA is a parallel computer architecture developed by NVIDIA, which includes Runtime API and Driver API. Compared with OpenCL, CUDA is with better performance. In this paper, we used plenty of similar kernels to compare the computing performance of C, OpenCL and CUDA, the two kinds of API's on NVIDIA Quadro 4000 GPU. The experimental result showed that, the executive time of CUDA Driver API was 94.9%~99.0% faster than that of C, while and the executive time of CUDA Driver API was 3.8%~5.4% faster than that of OpenCL. Accordingly, the cross-platform characteristic of OpenCL did not affect the performance of GPU.**

I. INTRODUCTION

As Graphical User Interface (GUI) becomes more widely-used, it comes to develop architectures like GPU [1]. Traditional GPU can only be applied in graphic processing. GPU has two features; one is that it uses hundreds of simple cores in parallel processing to enhance performance, the other is that it has wider memory bandwidth to provide GPU with better data transmitting quality. When the powerful computing performance was applied to other fields, the concept of GPGPU (General-Purpose Computing on Graphics Processing Units) [2] was developed. However, the researchers and developers need to first understand the program design of OpenGL [3] or DirectX [4] before they can apply GPU in other fields through computer graphics processing. For those who are not familiar with graphics, it is an arduous task to apply GPU in their development. However, with OpenCL [5] and CUDA [6], the two programming models, researchers and developers can use the powerful computing performance of GPU in a much easier way.

OpenCL is standardized by Khronos Groups and public on December 2008. The main purpose of OpenCL is to provide an easy-to-use open standardized API without any device limitation. In other words, an API with the characteristic of cross-platform. OpenCL provides two ways of parallel processing, Task-parallel and Data-parallel. Task-parallel mode uses work-items to control multi-core CPU. Data-parallel mode uses hundreds of parallel processors to process great many of work-items. As for the application of GPGPU, most applications are related to the Data-parallel of OpenCL. Some digital image processing methods such as Sobel filter [7], Gaussian filter [8] are using it. Both has tons of information and need to process at the same way. Besides, each of information has no relative with other information make GPU parallel process much more effective.

CUDA is a parallel computer architecture developed by NVIDIA and debuted on June 2007. The purpose of CUDA is to provide an easy-to-use architecture for parallel processing only. That is, for NVIDIA's GPU only. Researchers and developers can use the powerful GPU through simple programming language and the extension syntax of CUDA. CUDA provides two kinds of API, including Runtime API and Driver API. The main benefit of Runtime API is that researchers and developers do not have to know the basic syntax of CUDA or to dispose the resources of CUDA, which makes it easier for them to use Runtime API to develop their programs. Besides, the main difference between Driver API and Runtime API is that Driver API can control the hardware resource, including device management and context management, which allows the researchers and developers to directly use the hardware resource to accomplish more complicated function or achieve better performance.

On the other hand, although the GPU contains both OpenCL and CUDA programming model, but they have the same ability to use GPU. In OpenCL, kernel only compiles when the program is running. This can provide the properties of cross-platform and device optimization, but it requires extra time to perform. Besides, in CUDA, kernel has to be run on the GPU developed by NVIDIA Company. Accordingly, CUDA can use more functions and achieve higher performance.

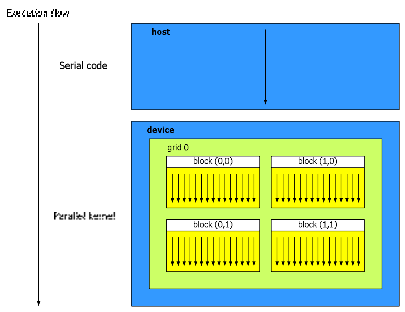

Fig. 1. Programming model

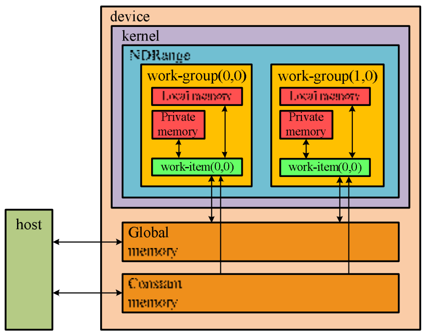

Fig. 2. OpenCL device architecture diagram

In this paper, we used plenty of similar kernels to compare the computing performance of C, OpenCL and CUDA, the two kinds of API's on NVIDIA Quadro 4000 GPU. In order to make this comparison fair, we didn't use any optimized program in the comparison of OpenCL and CUDA.

The rest of the paper is organized as follows. Section II briefly introduces OpenCL and CUDA technique and their differences. Section III further compares OpenCL and CUDA. Finally, Section IV concludes the paper.

II. OVERVIEW OF OPENCL AND CUDA TECHNOLOGY

In order to understand the difference between OpenCL and CUDA, The chapter introduces OpenCL and CUDA, including programming model, architecture, memory model, framework, and the programming. The five parts are presented in subsection A, subsection B, subsection C, subsection D, subsection E, respectively.

A. Introduction to the two programming models— OpenCL and CUDA

OpenCL and CUDA are the same programming models. As Fig. 1 shows, when the program encounters a function with lower parallelism, it would use the host CPU to perform the function (Serial code). Compare with using GPU to execute the function, using host CPU can obtain better performance. On the other hand, if the program encounters a function with higher parallelism, it would use the device to perform the function (Parallel kernel). In other words, GPU can use hundreds of parallel processing cores and a great number of work-items/threads to perform functions with higher parallelism, which can unleash the powerful computing performance of GPU.

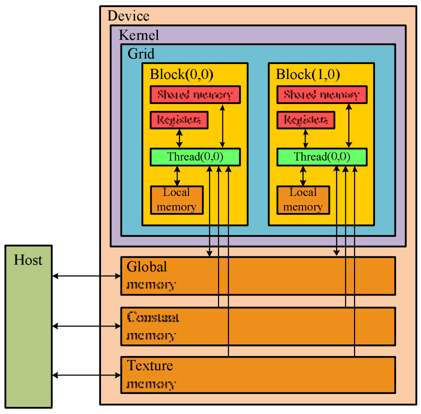

Fig. 3. CUDA device architecture diagram

B. Introduction to OpenCL and CUDA Architecture

Fig. 2 is the architecture diagram of OpenCL device, If the program encounters a function with higher parallelism, host CPU would call the kernel in the OpenCL device to perform a parallel operation. In the kernel, the smallest unit of actual parallel computing performance is called work-item. The group composed by multiple work-items is called work-group. On the other hand, a computing space arranged by multiple work-groups is called NDRange. The work-item, the work-group, and the NDRange can set the dimensions of space as one-dimensional, two-dimensional or three-dimensional space. Fig. 3 is the architecture diagram of CUDA device, whose structure is basically the same as that of OpenCL. The above-mentioned work-item, work-group and NDRange are corresponded with the thread, block and the grid of CUDA.

C. Introduction to OpenCL and CUDA memory model

Fig. 2 is OpenCL device architecture diagram, OpenCL memory model, includes Private memory, Local memory, Constant memory, and the Global memory, arranged according to their transmission speed.

1) Private memory

Private memory is stored in the GPU chip, which only allows work-items within a work-group to access the data. Since the data is stored in the high register, it can be instantly accessed.

2) Local memory

Local memory is stored in the GPU chip, which only allows work-items within a work-group to access the data. Since the data is stored in the high register, its accessing speed is the same as that of Private memory.

3) Constant memory

Constant memory is stored in the memory of the GPU. In the GPU, any work-item within the work-group can read data, but cannot write data; while CPU can read and write data in Constant memory. Since Constant memory has the features of Cache memory, its accessing speed is the fastest.

4) Global memory

Global memory provides wider memory bandwidth, but is also with higher latency, so its accessing speed is the slowest.

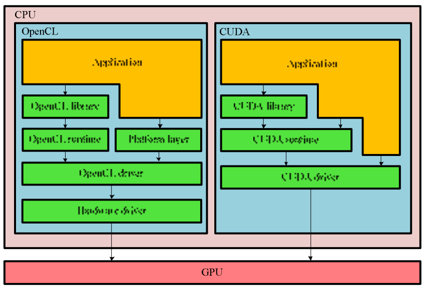

Fig. 4. OpenCL and CUDA framework

In Fig. 3 is CUDA device architecture diagram. The CUDA memory model includes Register, Shared memory, Local memory, Constant memory, Texture memory, and the Global memory, arranged according to their transmission speed. The aforementioned Register, Shared memory, Constant memory, and the Global memory are corresponded to the OpenCL Private memory, Local memory, Constant memory, and the Global memory, respectively.

1) Local memory

Local memory is stored in the memory of the GPU, which only allows the threads within a block to access the data. Compared with Global memory, the memory bandwidth of Local memory is lower, so it is with lower latency. The accessing speed of Local memory is between Global memory and Constant memory.

2) Texture memory

Texture memory is stored in the memory of the GPU. In GPU, any threads in a block can read the data, but cannot write data; while CPU can read and write data in Textual memory. Since Texture memory and Cache memory share same features, the accessing speed of Texture memory is the same as that of Constant memory.

D. Introduction to OpenCL and CUDA framework

Fig. 4 is the framework of OpenCL and CUDA. GPU parallel computing performed by OpenCL and CUDA are quite similar, since both are constituted of Runtime API and Library API. In OpenCL, device information is obtained through the Platform layer before performing the program. Then OpenCL driver calls the kernel in the GPU to perform parallel computing. CUDA is classified into Runtime API and Driver API. Through Runtime API, researchers and developers do not need to control hardware resource (e.g., Device management, Context management), but can call kernel directly to perform parallel computing. On the other hand, through Driver API, researchers and developers must control the hardware resources, and call the kernel to perform parallel computing.

E. Introduction to OpenCL and CUDA programming

In OpenCL programming, we provide a simple design process, which involves the following steps: work-item dimension and quantity setting, work-group dimension and quantity setting, input the resolution of test sequences, initialization and set the memory space required by program (GPU), input CPU data into the memory of the program (GPU), execute the programs of the program (using GPU parallel computation), rewrite the memory data of the program (GPU) back to the CPU, wait until all the work-items were executed before continuing with other execution.

In CUDA Runtime API programming,we provide a simple design process, which involves the following steps: set the dimension and quantity of thread, set the dimension and quantity of block, and input the resolution of test sequences, initialization and set the memory space required by kernel (GPU), wait until all the threads were executed before continuing with other execution, Input CPU data into the memory of the kernel (GPU), execute the program of the kernel (using GPU parallel computation), rewrite the memory data of the kernel (GPU) back to CPU, wait until all the threads were executed before continuing with other execution.

In CUDA Driver API programming,we provide a simple design process, which involves the following steps: set the dimension and quantity of thread, set the dimension and quantity of block, and input the resolution of test sequences, input CPU data into the memory of kernel (GPU), set input sequence of the parameter required by the kernel (GPU), execute the program of the kernel (using GPU parallel computation), wait until all the threads were executed before continuing with other execution, rewrite the memory data of the kernel (GPU) back to CPU.

In OpenCL and CUDA Runtime API, aside from memory configurations, function type qualifier and the fact that OpenCL does not have to allocate work-group and work-item to their corresponding data locations while CUDA Runtime API does, their basic functions within the program are the same. On the other hand, the difference between OpenCL and CUDA Driver API is the same as that between OpenCL and CUDA Runtime API. Besides, OpenCL and CUDA Driver API need to initialize and set the system, while CUDA Runtime API does not.

Through the above analysis we can see that there is no significant difference between the program design of OpenCL and CUDA, the two kinds of API's. Therefore, users only have to learn any one of the programming model to control the GPU. If users choose to use other programming model, they can still get used to it with ease.

III. OPENCL AND CUDA COMPARISON

This paragraph compares and analyzes C, OpenCL and CUDA, the two kinds of API's, which are written kernel according to the five application benchmarks. The five application benchmarks includes Sobel filter, Gaussian filter, Median filter from contain digital image processing [9], as well as Motion Estimation, Disparity Estimation [10] of multiview video coding (MVC) extended from H.264/AVC standard. As for Memory models, we exclude memory models with fast transmission speed and apply Global memory with the same transmitting speed for data transmission, to achieve an objective comparison. Finally, we introduce the experimental and simulation environment. The applied hardware platform includes VIA Nano processor L2200@1.6GHz for CPU, NVIDIA Quadro 4000@0.95GHz for GPU, 1.5GB memory and Linux system. As for sequences test, to examine the three application benchmarks

Table I Performance comparison of the two application benchmarks

Sequences	Two Application benchmarks							
	Executive time of Sobel filter (s)				**Executive time of Gaussian filter (s)**			
	C	**OpenCL**	**CUDA Runtime API**	**CUDA Driver API**	**C**	**OpenCL**	**CUDA Runtime API**	**CUDA Driver API**
Akko	4.65	0.249	0.25	0.239	6.46	0.32	0.313	0.29
Coral	4.73	0.244	0.25	0.237	6.43	0.31	0.317	0.28
Dolphins	4.64	0.246	0.25	0.238	6.49	0.32	0.32	0.279
FOREMAN	4.7	0.245	0.25	0.236	6.45	0.3	0.321	0.299
Mountain	4.75	0.247	0.24	0.239	6.37	0.31	0.32	0.289
Speed	4.69	0.246	0.24	0.236	6.47	0.33	0.32	0.283
Stefan	4.71	0.248	0.25	0.234	6.43	0.31	0.326	0.293
Average	4.69	0.246	0.247	0.237	6.4	0.314	0.32	0.288

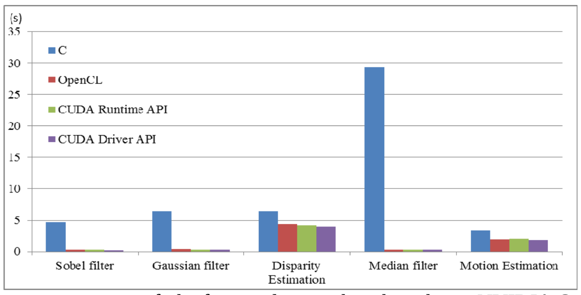

Fig. 5. Performance comparison of the five application benchmarks on NVIDIA Quadro 4000 GPU

of digital image processing, we apply seven video sequences, including video sequences with an original resolution of 720*480, such as “Akko”, “Coral”, “Dolphins”, “FOREMAN”, “Mountain”, “Speed”, “Stefan”. On the other hand, to examine the two application benchmarks of multiview video coding (MVC) extended from H.264/AVC, we applied six video sequences, including video sequences with an original resolution of 640*480, such as “Akko&Kayo”, “ballroom”, “exit”, “flamenco”, “race1”, “rena”. Each video sequence also involved many different video characteristics, such as still video, motion video, fast motion video, rotating video, complex background texture, recording with active camera, etc.

From Table I we can see that, according to the two application benchmarks— Sobel filter and Gaussian filter, the executive time of CUDA Driver API was 0.4%~1.8% faster than that of OpenCL, and the average executive time difference was quite small. On the other hand, the executive time of CUDA Driver API was 94.9%~95.5% faster than C, and the average executive time difference was significant. Accordingly, the extra executive time required by inter-platform characteristics did not affect the overall efficacy of OpencCL. Fig. 5 compares the performance of the five application benchmarks on NVIDIA Quadro 4000 GPU. From the figure we can see that the aforementioned situation can be experimentally verified by five other application benchmarks to ensure that the difference in application base did not change the aforementioned result.

IV. CONCLUSION

In this paper we applied five application benchmarks to compare the efficacy of C, OpenCL and CUDA, the two kinds of API’s. The comparison and analysis result showed that the executive time of CUDA Driver API was 94.9%~99.0% faster than that of C, while the executive time of CUDA Driver API was 3.8%~5.4% faster than that of OpenCL. Accordingly, the extra executive time required by the inter-platform characteristic of OpenCL, did not seem affect the overall performance of OpenCL. Besides, there was little difference between the program design of OpenCL and CUDA. This allows researchers and developers to choose one kind of programming model to control GPU according to their need, and thus enhance the overall system efficacy and reduce development time.

ACKNOWLEDGMENT

The authors are grateful to the National Science Council, Taiwan for the sponsorship under grant: NSC 100-2221-E-224-034-MY3.

REFERENCES

[1] K. Fatahalian and M. Houston, “A closer look at GPUs,” *Communications of the ACM*, Vol. 51, No. 10, October 2008.

[2] M. Dimitrov, M. Mantor, and H. Zhou, “Understanding Software Approaches for GPGPU Reliability,” *The 2nd workshop on General-Purpose Computation on Graphics Processing Units (GPGPU-2)*, 2009.

[3] D. Blythe, “OpenGL ES Common/Common-Lite Profile Specification,” version 1.0.02(annotated), Khronos Group, 2004.

[4] D. Lin and J. Hui. “Using C++ DirectX to achieve 3DS file read and control,” *Computer Age*. 2010, pp.42.

[5] Khronos OpenCL Working Group. “The OpenCL Specification Version 1.1. Khronos Group,” 2009. http://www.khronos.org/opencl.

[6] NVIDIA CUDA C Programming Guide 4.2. NVIDIA,May 2012.

[7] M. Wen, C. Zhong, “Application of Sobel Algorithm in Edge Detection of Image,” *China High-tech Enterprise*, pp. 57-62, 2008.

[8] V. Aurich, J. Weule, “Non-linear Gaussian Filters performing edge preserving diffusion,” *DAGM Symposium*, 1995.

[9] L. Yin, R. Yang, M.Gabbouj, Y. Neuvo, “Weighted Median Filters: A Tutorial,” *IEEE Trans. on Circuits Syst.*, vol. 43,no. 3, March, pp. 157-192, 1996.

[10] A. Vetro, P. Pandit, H. Kimata, A. Smolic and Y-K. Wang, “Joint Draft 8.0 on Multiview Video Coding,” *ISO/IEC JTC1/SC29/WG11 and ITU-T Q6/SG16, Doc. JVT-AB204*, Jul. 2008.