Parallel Application Performance Comparison with Vulkan, CUDA and OpenMP

SeongHu Hong* and DoHyeong Kim*, Chang-Sung Jeong
Department of Computer Engineering, Korea University
Anam-dong 5-ga, Seongbuk-gu 136-713, South of Korea
E-mail: valvenis@korea.ac.kr, 2015010681@korea.ac.kr, csjeong@korea.ac.kr

Abstract: Vulkan is an API (Application Programming Interface) for graphics and compute hardware, launched by Khronos. For Programming model, it recently draws attention as a Graphic and Compute Integrated API, and much works have been conducted focused on Vulkan's Graphic API performance, but not so much works on Vulkan's Compute API. Therefore, in this paper, we shall evaluate and compare the performance of Vulkan's Compute API with respect to CUDA and OpenMP by implementing three parallel applications using them respectively. Also, we shall show that the performance of Vulkan is similar to that of CUDA, and has advantage of using graphic and compute operations at the same time.

Keywords—**OpenMP, CUDA, Vulkan, GPGPU, Parallel Application**

1. Introduction

GPU has a dissimilar design concept of CPU[1]. A couple of CPUs can sequentially perform diverse tasks. On the contrary, GPU has many-core, a GPU's single core has worse performance than a CPU core, and has carried out graphic rendering computation. Because many-core can manage many independent data, kind of pixel, but CPU's sequential operation takes a long time compared with GPU. GPU is optimized for single instruction multiple data (SIMD). So if each operation has operating on independent data, the application can be effectively parallelized on GPU than CPU. Lately, due to this feature of many-core and parallel operation, GPU is taking advantage of various field.

Usage of GPUs for different operation such as general signal processing, physical simulation, financial forecasts, biological calculation means GPGPU (General Purpose Graphics Processing Unit)[2][3]. By utilizing GPGPUs, we can reduce time of massive data processing or plenty of mathematical calculations. However, when we directly handle GPUs to use different purpose, it is difficult to control GPU memory and make program without detail GPUs System information and graphic engine. So, until the compute unified framework introduced, such as CUDA [4], OpenCL and OpenACC, there were many restrictions to use. In addition, Programmers were not easy to use for GPGPU. Therefore, programming models are developed, which support GPGPU such as CUDA, OpenACC, Vulkan API and so on, in order to use GPU as general purpose. Moreover, from GPGPU support high-level language, it makes it easier to use.

Khronos launched the Vulkan specification on 2016, and Khronos members released Vulkan drivers and SDKs also. It can be used not only for Graphic rendering but also used as GPGPU like OpenCL, CUDA that has features of Compute API. However, it still does not have a sufficient reference and performance analysis as GPGPU either. Most of the works about Vulkan focused on Graphic rendering.

In this paper, we shall evaluate and compare the performance of Vulkan's Compute API with respect to CUDA and OpenMP by implementing three parallel applications using them respectively. We use OpenMP and CPU programming model tocreate and perform thread while CUDA and Vulkan API to compare each performance of GPGPU model using three parallel applications. Also, we shall show that the performance of Vulkan is similar to that of CUDA, and has advantage using graphic and compute operations at the same time.

This paper is organized as follows. In Section 2, we give some information of Programming model which be used experiment. Section 3 explains parallel applications. The fourth section shows results in comparison of OpenMP, CUDA and Vulkan API with above three applications.

2. Programming Languages

In this work, three parallel applications are evaluated by OpenMP, CUDA and Vulkan API to compare with CPU thread model and GPGPU model. OpenMP is CPU thread model, CUDA and Vulkan API are GPGPU model.

2. 1 OpenMP

OpenMP(Open Multi-Processing) is an API that provide shared memory multiprocessing programming in C, C++, and Fortran(77, 09, 95)[5]. It is effective at loop-level parallism, which is set of compiler directives, libarry rotines, and evironment variables. It effectively enable parallel processing at loop-level prallelism in applications by simple directives. OpenMP uses the fork-join model. In other words, when OpenMP program start, a master thread operate sequentially untill the openmp directive. The master thread create parallel threads which is slave threads if the master thread meet the directive.
One of the several OpenMP's advanges can reduce modification of code by using directive. Onother advantage is able to communicate each thread easily, because threads have shared memory architecture. However, if the number of thread is more then CPUs, program performance is fallen.

* These two authors contributed equally to this work.

2. 2 CUDA

CUDA (Compute Unified Device Architecture) is a model of parallel programming by using graphics processing unit (GPU), which is developed by NVIDA to increase GPUs' flexibility [6]. In the past, people took advantage of GPUs for graphic processing, not general purpose. The goal of CUDA is supported integrated development environment (IDE) of GPGPU by GPUs.

CUDA is simple to use. Host allocates GPUs' memory and then send data in the host to the GPUs. GPUs operate calculation with received data and then results send back to the host. GPUs have a shared memory architecture, which available fast communicatoin with each threads. We use shared memory to optimize program.

Figure 1 shows CUDA Thread Batching model. Data parallel parts of an application are executed on the CUDA Device as kernels which run in parallel on many threads. A Kernel is executed as a grid of blocks. Blocks are consist of threads.

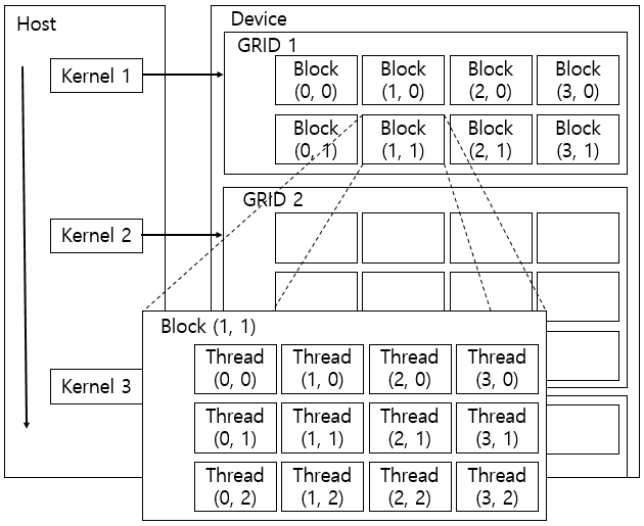

Figure 1. CUDA Thread Batching Model

2. 3 Vulkan API

Vulkan API is a graphics and compute API published by Khronos on February 16, 2016[7]. It is an advanced API from OpenGL that operates 3D graphic rendering and has computing operation like GPGPU. Goals of using Vulkan API are greatly reducing CPU overhead and driver load, and provide cross platform to utilize mobile machine or personal computer.

We are interested in Vulkan API's computing operation. Experiment is done except for graphic rendering in order to found capacity of computing operation as GPGPU. Basically, Vulkan Compute API for GPGPU is shader based. Each terminology is different with CUDA, but the parallel programming model in Vulkan shows similar with programming model in CUDA. Table 1 shows the general terminology of Vulkan and CUDA. In addition, Figure 2 shows Work Group model of Vulkan. It has similar structure with CUDA. Programmer implements the part of matching Kernel on CUDA on shader code and can execute the shader code through call cmdDispatch function.

Table 1. Vulkan & CUDA General Terminology

Vulkan	CUDA
Work Item(Invocation)	Thread
Work Group	Thread Block
Global Work Group	GRID

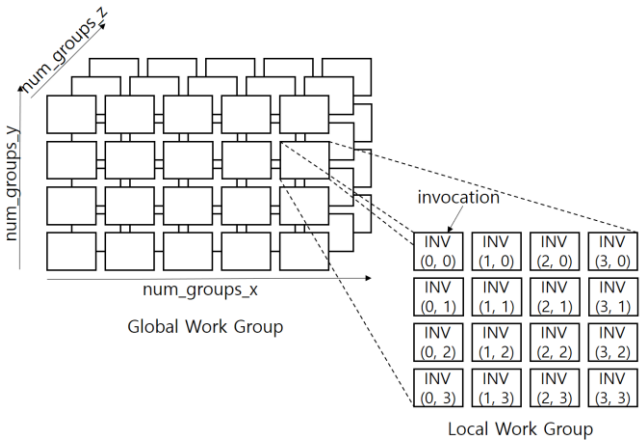

Figure 2. Vulkan Work Groups

3. Applications

We evaluate the performance of three applications on CPU and GPGPUs models.

3.1 Monte Carlo Method

Monte Carlo Method is a class of computational algorithm. It uses repeated random sampling to define constraints on the value and then makes a sort of "best guess". They are often used when simulating physical and mathematical problems.

In this work, we used Monte Carlo Method to calculate the value for π . If there is a circle and a square where the length of a side of the square was the same as the diameter of the circle, the ratio of the area of the circle to the area of the square would be $\pi/4$. Then the value of π can be approximated using a Monte Carlo method[8].

3.2 Particle Operation

Particle systems are used for a lot of graphic effects such as smoke, explosions, smoke and spray[9]. Tens of thousands of particles are needed to quickly calculate positions to be shown effects at every frame. Therefore, it is suitable for use of GPGPU that is available many threads and operate many computations at the same time.

3.3 N-Queens Problem

N-Queens problem is that a set of n queens are laid on the chessboard, when each of n queens avoid lines that one queen can attack the other. The queen can attack horse on horizontal line, vertical line and diagonal line. It is possible to have numerous solutions. We found all number of cases and solutions.

4. Methodology

In this section, we describe methodology to check performances. We implement each application based on the identical code to compare performance of sequential and parallel applications using OpenMP, CUDA and Vulkan. Each application on OpenMP, CUDA and Vulkan platform is optimized for each programming language and tested.

Each application is implemented sequentially or recursively using C. Evaluation is performed after optimization that most fits to each programming language. Therefore, each parallel application has different features according to the programming model on it is implemented.

Test is repeated 10 times, and average execution time calculated from that. Execution time is just processing core computing, but not for communication and idle time.

All Experiments are carried out on the following hardware configuration.

Mother board Gigabyte Technology Co. Ltd. Z77-D3H
Processor Intel® Core™ i5-2500 CPU @ 3.30GHz
System Memory DDR3 12G
GPU NVIDIA GeForce GTX 750 Ti
OS Windows 10 pro

5. Experimental Result

In a PC belong GPUs, we evaluate three programming model OpenMP, CUDA and Vulkan API with three parallel applications. Figure 3 shows the results of time measurement for three parallel applications. In uppermost graph, orange color bar represents the time for searching all number of cases non-recursively in the N-Queens problem, while blue bar for finding solutions recursively. Vulkan API uses shader program to operate on GPUs, and does not support recursion. Therefore, we cannot experiment Vulkan API for recursive N-Queens application. Unlike Non-recursion application, GPGPU is slower than CPU for recursive case, since N-Queens problem does not require many cores. In addition, due to the task dependency between recursive call, all the processes need to synchronize for the next recursive call.

For Monte Carlo application with size of 256 x 256, CUDA has slightly better performance than Vulkan API's.

Particle operation has 8162 particles, and can be parallelized easily, since each particle can be processed independently. CUDA and Vulakn has much better performance than OpenMP and C. GPGPU programming models has the best performance for particle operation.

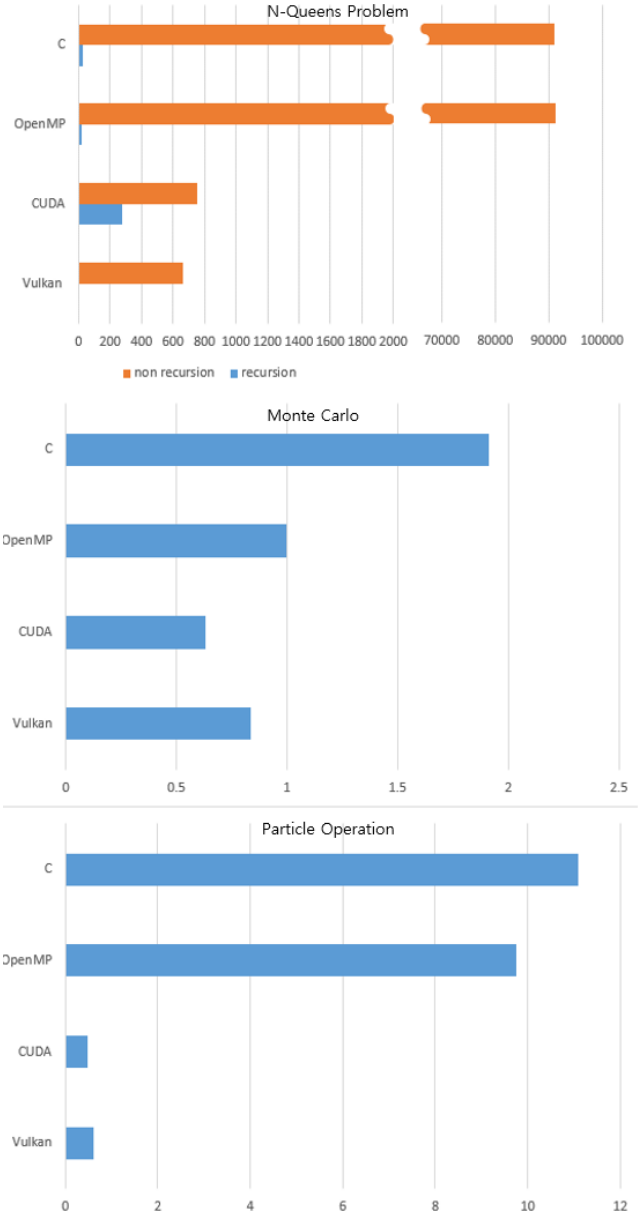

Figure 3. Time measurement of three parallel applications using parallel programming model.

6. Conclusion

In this paper, we have measured the performance of OpenMP, CUDA and Vulkan API on CPU and GPUl. There is no big difference in performance between CUDA and Vulkan API's. However, CUDA is slightly better than Vulkan API. For computational intensive processing such as high-performance servers and clusters, CUDA has an advantage over Vulkan. However, Vulkan API has many advantages: Vulkan API is supported on every platform and modern GPU. In addition, developer can use graphic API and compute API at the same time when using Vulkan. Also, Vulkan can execute Graphic rendering and compute operations at the same time. As a future work, we are going to carry out research on optimizing applications for computation intensive image processing by using Vulkan API.

References

[1] Ledur, Cleverson Lopes, Carlos MD Zeve, and Julio CS dos Anjos. "Comparative analysis of OpenACC, OpenMP and CUDA using sequential and parallel algorithms." *11th Workshop on parallel and distributed processing (WSPPD)*. 2013.

[2] Owens, John D., et al. "GPU computing." *Proceedings of the IEEE* 96.5 (2008): 879-899.

[3] Owens, John D., et al. "A Survey of general-purpose computation on graphics hardware." *Computer graphics forum*. Vol. 26. No. 1. Blackwell Publishing Ltd, 2007.

[4] Nickolls, John, et al. "Scalable parallel programming with CUDA." *Queue* 6.2 (2008): 40-53.

[5] Chandra, Rohit. *Parallel programming in OpenMP*. Morgan kaufmann, 2001.

[6] Lee, Seyong, Seung-Jai Min, and Rudolf Eigenmann. "OpenMP to GPGPU: a compiler framework for automatic translation and optimization." *ACM Sigplan Notices* 44.4 (2009): 101-110.

[7] https://www.khronos.org/vulkan

[8] Kalos, Malvin H., and Paula A. Whitlock. *Monte carlo methods*. John Wiley & Sons, 2008.

[9] Nguyen, Hubert. *Gpu gems 3*. Addison-Wesley Professional, 2007.