Mathematical Statisticianand Engineering Applications
ISSN:2094-0343
2326-9865

Comparing Cuda and Opencl: Unleashing Gpu Computational Power in Contemporary High-Performance Computing

Palyam Nata Sekhar [1], Dr. Arpana Bharani [2]

[1] Research Scholar, Department Of Computer Science, Dr. A. P. J. Abdul Kalam University, Indore, Madhya Pradesh

[2] Supervisor, Department Of Computer Science, Dr. A. P. J. Abdul Kalam University, Indore, Madhya Pradesh

Article Info
Page Number: 2145 - 2152
Publication Issue:
Vol 71 No. 3 (2022)

Article History
Article Received: *12 January 2022*
Revised: *25 February 2022*
Accepted: *20 April 2022*
Publication*: 09 June 2022*

Abstract

The contemporary landscape of high-performance computing has been irrevocably altered by the advent of parallel programming frameworks, CUDA and OpenCL. These frameworks, CUDA being NVIDIA's proprietary creation and OpenCL a platform-agnostic standard, have unlocked the immense computational potential latent within Graphics Processing Units (GPUs). In this article, we evaluate CUDA and OpenCL with the identical complicated kernels. We demonstrate that utilizing the NVIDIA compiler tools, very minor adjustments are required to successfully convert a CUDA kernel to an OpenCL kernel. Before this kernel can be created using ATI's tools, more tweaks are required. Our benchmarks compare CUDA with OpenCL in terms of data transfer rates to and from the GPU, the time it takes for the kernel to execute, and the total amount of time it takes for the application to run.

Keywords: Kernel, Transfer, Running time, Execution, Graphics

I.Introduction

In the fast-paced world of modern computing, where performance and parallelism are king, two parallel programming frameworks have emerged as cornerstones of the GPU (Graphics Processing Unit) revolution: CUDA and OpenCL. These two powerful tools have fundamentally transformed the way we harness the computational prowess of GPUs, unlocking unprecedented levels of speed and efficiency across a spectrum of applications ranging from scientific simulations and artificial intelligence to video game rendering and cryptocurrency mining. Developed by NVIDIA, CUDA, which stands for Compute Unified Device Architecture, and the open standard OpenCL, maintained by the Khronos Group, represent two distinct approaches to GPU programming, each with its unique strengths and applications.

At their core, CUDA and OpenCL share a common goal: to tap into the immense parallel processing capabilities of GPUs for general-purpose computing tasks. However, the paths they tread and the philosophies they adhere to diverge, giving rise to distinct ecosystems and communities of developers. CUDA, born from the minds at NVIDIA, boasts a proprietary heritage but provides unrivaled integration with NVIDIA GPUs, offering fine-grained control and performance optimization tailored to their architecture. On the other hand, OpenCL adopts a more open and platform-agnostic stance, aiming to create a universal interface for parallel

computing across a multitude of hardware vendors, including NVIDIA, AMD, Intel, and more. This fundamental difference in approach has profound implications for developers, researchers, and organizations seeking to harness the GPU's immense computational potential.

As we venture deeper into the intricacies of CUDA and OpenCL, this exploration will traverse their architecture, programming models, and ecosystem dynamics. Additionally, we will embark on a comparative journey, shedding light on the nuanced performance attributes of CUDA and OpenCL, offering insights into when to choose one over the other based on specific use cases and hardware configurations. Furthermore, we will uncover real-world applications where these frameworks have made indelible marks, advancing fields such as deep learning, scientific research, financial modeling, and many others.

In the ever-evolving landscape of computing, CUDA and OpenCL have carved out their niches, propelling GPU technology into mainstream computing and accelerating innovation in ways previously unimaginable. To understand the true potential and impact of these frameworks, we must delve into their histories, dissect their inner workings, and appreciate the broad spectrum of challenges they have overcome. As we embark on this journey through the realms of CUDA and OpenCL, we will gain a comprehensive understanding of their roles in shaping the present and future of high-performance computing.

II.Review Of Literature

Li, Xuechao & Shih, Po-Chou (2018) In this research, we compare and contrast the features of CUDA and OpenACC. Compilers and frameworks are the primary targets of the performance analysis. To objectively analyze the subjective performances of OpenACC and CUDA implementations with regards to their sensitivity to changes in data size, we have established a Performance Ratio of Data Sensitivity (PRoDS) measure. Since OpenACC kernels need to be transformed by the PGI compiler into object code before they can be run, the results show that OpenACC performs worse than CUDA in this regard, while CUDA instructions can be executed instantly. Furthermore, CUDA is more susceptible to data changes without optimizations, but OpenACC is more sensitive to data changes with optimizations. We found that OpenACC offers a more reliable method of programming when compared to CUDA, the current standard for accelerator devices.

Bhura, Mayank et al., (2016) As heterogeneous systems grow more prevalent, GPUs are increasingly being used for high-performance computing tasks. Until recently, practically all such NVIDIA GPU-based GPGPU apps were written in CUDA. However, the framework currently only supports NVIDIA GPUs, thus any new processing devices would need to be reimplemented elsewhere. Thanks to its open and vendor-agnostic nature, OpenCL may be thought of as a "write once, run anywhere" framework that can be used with a wide variety of processors and accelerators. Nonetheless, there are benefits and drawbacks to both paradigms.

de Paula, Lauro (2014) In this work, we examine the similarities and differences between Compute Unified Device Architecture (CUDA) and Open Computing Language (OpenCL) as two parallel computing architectures. The computational performance of the two designs has been compared in certain published papers. To far, however, there has been no comprehensive

and up-to-date report that lays out exactly which architecture may be deemed the most effective. The focus of this study is on emphasizing the most cost-effective option among those that meet certain criteria, such as level of hardware and software, technical trends, and convenience of use. We do this by providing descriptions of the major works that make use of each of the architectures. Being a heterogeneous system, OpenCL may appear like the more natural option. Even though CUDA is exclusive to NVIDIA® graphics cards, we found that it has become a standard in the field.

Sugawara, Makoto et al., (2013) It is crucial to understand how OpenACC differs from more traditional GPU programming models in terms of available programming and tuning approaches, often known as performance tunabilities, before designing and developing any auto tuning mechanisms for OpenACC. Therefore, the performance-tuning capabilities of OpenACC and OpenCL are discussed in this work. Due to its inability to synchronize threads operating on GPUs, OpenACC is missing out on key crucial methods. As a result, we also create a new compiler directive for synchronizing threads. The results of the evaluations reveal that both OpenCL and OpenACC need architecture-aware optimizations, and that comparable techniques for enhancing the performance of the two are successful. With this new directive, OpenACC might follow OpenCL's lead and define a wider variety of tuning methods. If OpenACC can match the efficiency of traditional GPU programming models like CUDA and OpenCL, it will be a highly attractive programming paradigm, since it is clearly more productive than OpenCL, particularly for legacy application migration.

Su, Ching-Lung et al., (2012) The Graphics Processing Unit (GPU) has revolutionized the computer industry. Researchers and developers use GPU's parallel computing architecture to boost the performance of computer systems. However, GPU comes with two programming models—OpenCL (Open Computing Language) and CUDA (Compute Unified Device Architecture)—that may help cut down on the time it takes to create new goods. Researchers and developers may utilize GPU with little knowledge of OpenGL, DirectX, or any other program design thanks to the combination of these two programming paradigms. OpenCL is a standard, open API that works on a variety of different platforms. NVIDIA's CUDA architecture is a parallel computing framework that features a Runtime API and a Driver API. CUDA's performance is superior to that of OpenCL. In this research, the computational performance of C, OpenCL, and CUDA, the three different APIs available for the NVIDIA Quadro 4000 GPU, was compared using a large number of kernels that were comparable in nature. Based on the results of the experiments, the CUDA Driver API was shown to have an executive time that was between 94.9 and 99.0 percent quicker than C, and between 3.8 and 5.4 percent faster than OpenCL. Thus, OpenCL's platform independence has no negative effect on GPU performance.

Du, Peng et al., (2012) In this study, we assess OpenCL's viability as a language for creating GPGPU programs that are independent of their hardware's performance capabilities. Even though the Khronos group designed OpenCL with code portability in mind, performance may suffer when switching between platforms. Initializations in OpenCL have a negative effect on performance, while such a requirement does not present in competing languages like CUDA. By considering these factors, we are able to provide a single library that performs adequately

across several platforms. As examples of level 3 BLAS routines, we chose triangle solver (TRSM) and matrix multiplication (GEMM) to implement in OpenCL. By profiling TRSM, we can determine how OpenCL's runtime system distributes its time. For the most recent graphics processing units (GPUs), the NVIDIA Tesla C2050 and the ATI Radeon 5870, we supply customized GEMM kernels. We investigate how leveraging the texture cache helps speed, how transferring data into pictures impacts performance, and how optimizations differ between the OpenCL and CUDA compilers. From our experiments, we know that both GPUs can achieve close to 50% of their max performance in OpenCL for GEMM. We also demonstrate that these kernels' performance is not very portable. Last but not least, we suggest auto-tuning as a means by which the parameter space of these kernels may be further explored using search harness.

Fang, Jianbin et al., (2011) This study provides a detailed analysis of the relative performance of CUDA and OpenCL. We have chosen 16 benchmarks that represent a wide variety of synthetic and real-world use cases. We undertake a deep dive into the performance discrepancies, investigating the underlying compilers, programming models, optimization techniques, and architectural specifics. Based on our findings, CUDA is only up to 30% faster than OpenCL. We further demonstrate that this disparity is the result of using misleading comparisons, and that OpenCL is capable of achieving results on par with CUDA. As a result, we provide criteria for a balanced evaluation of the two platforms, laying the groundwork for future studies. We also run the benchmarks on a number of other prominent platforms, with only minor tweaks made to demonstrate OpenCL's portability. Overall, we discover that OpenCL's performance is not drastically impacted by its portability, and that it is a competitive option to CUDA.

Komatsu, Kazuhiko et al., (2010) In addition to CUDA, a new open programming standard called OpenCL has just been accessible for GPGPU programming. Because of its higher abstraction programming environment, OpenCL is able to handle a wide range of computing devices. Given the semantic difference between OpenCL and computing devices, it is crucial to clarify the OpenCL C compiler's capabilities so that they may be used to their fullest. In this research, we conduct a systematic comparison of the performance of CUDA and OpenCL applications. We begin by creating almost identical applications in CUDA and OpenCL and comparing their speeds. Then, the primary causes of the observed performance gaps are examined. If the kernel codes are properly optimized by hand or by the compiler optimizations, the evaluation results imply that OpenCL applications are equivalent to those of CUDA ones in terms of performance. This research also compares the performance of GPUs manufactured by NVIDIA and AMD to provide light on the differences between their OpenCL implementations. Based on the results of the performance comparison, it is clear that in order to get the most out of each GPU, the compiler choices of the OpenCL C compiler and the execution configuration parameters must be improved. Since a single OpenCL code must operate effectively on a wide range of GPUs, automated param-eter tweaking is a need.

III.Implementation

Adiabatic QUantum Algorthms (AQUA) is a C++ program that does a Monte Carlo simulation of a quantum spin system and is utilized in this work. As a rough approximation, we use a classical Ising spin system to model the quantum spin configuration. Copies of the quantum system that are magnetically connected to one another make up the classical approximation. Each replica is joined to another replica twice, creating a ring. The Suzuki-Trotter decomposition describes this approximation procedure. The resulting structure is referred to as "layered" here.

At various stages of an adiabatic quantum development, we model each layered system. Since the program simulates the adiabatic development of a full layered system at each point, The total number of variables it handles is proportional to the product of the number of points by the number of systems involved.

A CUDA implementation of the technique is shown, along with a discussion of how AQUA maps data structures to GPU and CPU threads. In this study, we focused on enhancing the way the kernel accessed memory. Then, we used NVIDIA's development tools to convert the CUDA kernel to OpenCL with a minimum of disruption to the core functionality of the GPU computing platform. OpenCL required rewriting of ancillary code, such as that used to identify and configure the GPU, or that which copied data to and from the GPU.

IV.Performance Tests

Our program was run through its paces on an NVIDIA GeForce GTX-260, which included both CUDA and OpenCL tests. Since we cared about keeping the computer's responsiveness while the program ran, we purposely lowered the GPU's load to version 2.3 of both CUDA and OpenCL. In order to get the most out of the experiments in this study, we heavily loaded the GPU and minimized the time spent by the CPU executing code and copying data. This resulted in an extremely slow response time from the computer's user interface during the testing. During the actual data-gathering runs, no user intervention was tried to preserve the GPU's computational resources exclusively for the AQUA program.

During execution, the program passes through these stages: (1) Prepare the graphics processing unit (this involves detecting the GPU, developing the kernel for OpenCL, etc.) Second, the input is read; third, data is copied to the graphics processing unit (GPU); fourth, the kernel is executed on the GPU; fifth, data is copied back to the host; and sixth, the CPU processes and outputs the returned data.

Table 1 summarizes the GPU Operations Time, which includes the time it takes to transfer data to and from the GPU and execute the kernel (steps 3, 4, and 5). Each layered system's variables were swept by both kernels 20,000 times. Table 1 displays the total amount of time it takes to complete all six phases of the application's flow as the End-To-End time. We ran each task on CUDA and OpenCL 10 times each to get stable mean times.

Table 1 GPU and application running time (in seconds)

Qubits	GPU Operations Time				End-To-End Running Time			
	CUDA		OpenCL		CUDA		OpenCL	
	value	stdev	value	stdev	value	stdev	value	stdev
4	0.92	0.033	2.28	0.008	2.91	0.005	4.32	0.162
15	2.82	0.008	4.71	0.015	5.43	0.007	7.47	0.022
31	6.73	0.009	9.06	0.015	10.19	0.011	12.82	0.008
47	12.71	0.011	19.92	0.008	17.76	0.015	26.72	0.018
70	25.01	0.032	42.34	0.086	32.79	0.029	54.88	0.105
95	60.35	0.067	72.32	0.059	76.20	0.031	92.90	0.067
121	102.11	0.519	113.91	0.763	123.52	1.088	142.88	1.077

Table 2 breaks down the GPU Operations Time into the Kernel Running Time (step 4; executed once for each problem) and the Data Transfer Time to and from the graphics device (steps 3 and 5); this gives an idea of how well CUDA and OpenCL perform in these areas.

Table 2 Kernel execution and GPU data transfer (in seconds)

Qubits	Kernel Running Time				Data Transfer Time			
	CUDA		OpenCL		CUDA		OpenCL	
	value	stdev	value	stdev	value	stdev	value	stdev
8	1.92	0.024	2.20	0.002	0.008	0.005	0.012	0.009
16	3.87	0.008	4.75	0.015	0.012	0.002	0.025	0.011
32	7.69	0.009	9.03	0.010	0.022	0.013	0.042	0.013
48	13.65	0.011	19.82	0.011	0.058	0.013	0.084	0.005
72	25.90	0.041	42.14	0.088	0.109	0.004	0.149	0.012
96	61.13	0.067	71.94	0.052	0.211	0.011	0.290	0.014
128	100.81	0.531	113.51	0.759	0.308	0.013	0.415	0.005

The quantity of information sent and received between the GPU and the host is shown in Table 3. Both Step 3 (transferring data to the GPU) and Step 5 (transferring data from the GPU to the host) transfer half of the total given in Table 3.

Table 3 Data transferred between the GPU and the host (in KB)

Qubits	Data Transferred
8	649.08
16	1,633.28
32	3,553.49
48	8,210.19
72	15,338.82
96	33,124.47
128	49,541.06

V.Conclusion

In the ever-evolving world of computing, where the quest for performance and efficiency remains unending, CUDA and OpenCL have emerged as formidable pillars, each contributing its unique essence to the grand narrative of high-performance computing. CUDA, birthed within the confines of NVIDIA, has championed the cause of proprietary precision, offering meticulous control over GPU architecture and delivering exceptional performance for those within its ecosystem. OpenCL, on the other hand, stands as an emblem of open collaboration, fostering a platform-agnostic approach that seeks to unite diverse hardware vendors in a harmonious symphony of parallel processing. In the future, as technology continues its relentless march forward, CUDA and OpenCL will remain at the forefront of high-performance computing, adapting to new hardware architectures and emerging paradigms. Their legacy is not merely in lines of code or meticulously designed APIs, but in the breakthroughs they enable, the boundaries they push, and the ever-expanding horizons they beckon us to explore.

References: -

1. Li, Xuechao & Shih, Po-Chou. (2018). An Early Performance Comparison of CUDA and OpenACC. MATEC Web of Conferences. 208. 05002. 10.1051/matecconf/201820805002.

2. Bhura, Mayank & Deshpande, Pranav & Chandrasekaran, K.. (2016). CUDA or OpenCL:. 10.4018/978-1-4666-8737-0.ch015.

3. Mohhamad, Asad & Garg, Vikram. (2015). A Comparative Study of GPU Computing by using CUDA and OpenCL. International Journal of Computer Applications. 128. 1-3. 10.5120/ijca2015906022.

4. Zhang, Mingyu & Geng, Shujuan & Zhang, Junpeng. (2015). Comparison of CUDA and OpenCL performance on FDTD simulation. 10.1201/b19779-72.

5. de Paula, Lauro. (2014). CUDA vs. OpenCL: uma comparação teórica e tecnológica.

ForScience: Revista Científica do IFMG. 2. 31-46. 10.29069/forscience.2014v2n1.e53.

6. Sugawara, Makoto & Hirasawa, Shoichi & Komatsu, Kazuhiko & Takizawa, Hiroyuki & Kobayashi, Hiroaki. (2013). A Comparison of Performance Tunabilities between OpenCL and OpenACC. Proceedings - IEEE 7th International Symposium on Embedded Multicore/Manycore System-on-Chip, MCSoC 2013. 147-152. 10.1109/MCSoC.2013.31.

7. Su, Ching-Lung & Chen, Po-Yu & Lan, Chun-Chieh & Huang, Long-Sheng & Wu, Kuo-Hsuan. (2012). Overview and comparison of OpenCL and CUDA technology for GPGPU. IEEE Asia-Pacific Conference on Circuits and Systems, Proceedings, APCCAS. 448-451. 10.1109/APCCAS.2012.6419068.

8. Du, Peng & Weber, Rick & Luszczek, Piotr & Tomov, Stanimire & Peterson, Gregory & Dongarra, Jack. (2012). From CUDA to OpenCL: Towards a Performance-portable Solution for Multi-platform GPU Programming. Parallel Computing - PC. 38. 10.1016/j.parco.2011.10.002.

9. Bombieri, Nicola & Vinco, Sara & Bertacco, Valeria & Chatterjee, Debapriya. (2012). SystemC simulation on GP-GPUs: CUDA vs. OpenCL. 343-352. 10.1145/2380445.2380500.

10. Fang, Jianbin & Varbanescu, Ana & Sips, Henk. (2011). A comprehensive performance comparison of CUDA and OpenCL. Proceedings of the International Conference on Parallel Processing. 216-225. 10.1109/ICPP.2011.45.

11. Fang, Jianbin & Varbanescu, Ana & Sips, Henk. (2011). A comprehensive performance comparison of cuda and opencl. 2011 Int'l Conf. on Parallel Processing (ICPP). 216-225.

12. Komatsu, Kazuhiko & Sato, Katsuto & Arai, Yusuke & Koyama, Kentaro & Takizawa, Hiroyuki & Kobayashi, Hiroaki. (2010). Evaluating performance and portability of OpenCL programs.