An Early Performance Comparison of CUDA and OpenACC

Xuechao Li[1] and *Po-Chou* Shih[2]

[1]Department of Computer Science, Concordia University Chicago, River Forest IL, USA.
[2]Institute of Industrial and Business Management, National Taipei University of Technology, Taipei, Taiwan.

Abstract. This paper presents a performance comparison between CUDA and OpenACC. The performance analysis focuses on programming models and underlying compilers. In addition, we proposed a Performance Ratio of Data Sensitivity (PRoDS) metric to objectively compare traditional subjective performances: how sensitive OpenACC and CUDA implementations are to change in data size. The results show that in terms of kernel running time, the OpenACC performance is lower than the CUDA performance because PGI compiler needs to translate OpenACC kernels into object code while CUDA codes can be directly run. Besides, OpenACC programs are more sensitive to data changes than the equivalent CUDA programs with optimizations, but CUDA is more sensitive to data changes than OpenACC if there are no optimizations. Overall we found that OpenACC is a reliable programming model and a good alternative to CUDA for accelerator devices.

1. Introduction

High-performance computing has been applied into more and more scientific research applications such as chemical reaction process, information retrieval system [1], explosion process and fluid dynamic process. Currently the most popular high-performance computing device is the NVIDIA accelerator. However, the CUDA programming framework [13] requires programmers to fully grasp solid knowledge of software and hardware, and it is an error-prone and time-consuming work if legacy code needs to be rewritten by CUDA. To overcome the above limitations of CUDA and reduce developers' coding workload, the OpenACC programming model [14] has been released. It provides simple directives to achieve similar functions in CUDA and the OpenACC code can be run under different platforms. Hence, it is important to compare OpenACC and CUDA especially for parallel programming beginners because OpenACC can extremely reduce programmers' workload if they have the similar performance.

In terms of directive-based GPU programming models, some research work has been done on OpenMP accelerator [17], hiCUDA [18] and R-stream [19]. But with OpenACC motivation of simplifying CUDA programming with similar functions, in this paper we investigated the performance comparison of CUDA and OpenACC with the following two factors: (1) CUDA is one of the most popular parallel programming models and OpenACC is an easily learned and simplified high-level parallel language especially for programming beginners; and (2) One motivation of OpenACC development aims to simplify low-level parallel language such as CUDA. Finally, in this paper we make the following key contributions:

(1) We used 19 kernels in 10 benchmarks covering real-world applications and synthetic applications to present a detailed performance comparison between OpenACC and CUDA. To our best knowledge, this is the first paper of the performance comparisons between OpenACC and CUDA with a tremendous amount of benchmarks and performance metrics proposed.

(2) We made a quantitative comparison and a qualitative conclusion of CUDA and OpenACC with regard to programming models, optimization technologies, underlying compilers and data sensitivity.

The remainder of this paper is organized as follows: Section 2 discusses related work on performance comparison of diverse parallel programming models; Section 3 shows our configuration, testbeds and the selected benchmarks; Section 4 presents a methodology used in this paper; Section 5 discribes an overall performance analysis result, identifies and explains the main differences between CUDA and OpenACC; Section 6 presents the background of CUDA and OpenACC; Section 7 concludes this paper and presents our future work.

2. Related work

NVIDIA GPU devices are currently one of the most widely used accelerators so the CUDA programming language is used by most programmers, which requires professional software and hardware knowledge. To alleviate or avoid this limitation especially for beginners, OpenACC was developed by CAPS Enterprise, The Portland Group (PGI), Cray Inc. and NVIDIA. A good

performance comparison between OpenACC and CUDA can effectively assist programmers to make a decision on which language is better for their specific research work. Suejb *et al* [20] investigated OpenCL, OpenACC, CUDA and OpenMP, the threat that CUDA is not able to be run on Intel GPUs made conclusions less convinced. Also, compared to other papers [2-4] where only 1–2 benchmarks were used, we use 19 kernels in 10 benchmarks to fully evaluate OpenACC and CUDA programming models and use the PRoDS to objectively compare the data sensitivity. Although the results from the paper [2] showed that in general OpenACC performance is slower than CUDA's, we found that all results were based on only two micro benchmarks and one application. The use of few benchmarks cannot fully evaluate their performances so based on results from 19 kernels in 10 benchmarks in this paper, we find that CUDA codes still outperforms OpenACC codes in terms of kernel running time. In this section, a number of performance comparisons among different parallel programming models are presented.

Christgau *et al.* [4] presented a very interesting application--Tsunami Simulation EasyWave—for comparing CUDA and OpenACC using two different GPU generations (Nvidia Tesla and Fermi). Through listing runtime values under different hardware-specific optimizations—memory alignment, call by value and shared memory—three key points were revealed that (1) even the most tuned code on Tesla does not reach the performance of the unoptimized code on the Fermi GPU; (2) the platform independent approach does not reach the speed of the native CUDA code; (3) memory access patterns have a critical impact on the compute kernel's performance.

Tetsuya *et al.* [2] used two micro benchmarks and one real-world application to compare CUDA and OpenACC. The performance was compared among four different compilers: PGI [15], Cray, HMPP [16] and CUDA with different optimization technologies: thread mapping and shared memory in Matrix Multiplication micro benchmark and branch hoisting and register blocking were applied into 7-Point stencil micro benchmark. Finally, a performance relative to the CPU performance was compared in computational fluid dynamics application with kernel specification, loop fusion and fine-grained parallelization in the matrix free Gauss-Seidel method optimization. The results showed that OpenACC performance is 98% of CUDA's with careful manual optimization but in general it is slower than CUDA.

In [9] authors compared kernel execution times and data transfer times between host and device and end-to-end application execution times for both CUDA and OpenCL. The only application tested was Adiabatic Quantum Algorithms (AQUA) [10] which was implemented by complex, near-identical CUDA and OpenCL kernels. All run times and data transfer times were listed with different input sizes-Qubits. The conclusion was made that CUDA performed better when transferring data and CUDA's kernel execution was also consistently faster than OpenCL's.

An overview and comparison of OpenCL and CUDA was also presented in [11]. Five application benchmarks—Sobel filter, Gaussian filter, Median filter, Motion Estimation, Disparity Estimation—were tested under NVIDIA Quadro 4000. This paper compared and analyzed C, OpenCL and CUDA and the two kinds of APIs—CUDA runtime API and CUDA driver API. Detailed data comparison generated by different benchmarks generally showed CUDA had a better performance compared with OpenCL.

3. Experiment

3.1 Testbeds

All experimental results were obtained by nvprof (NVIDIA 2015) from the NVIDIA profiler toolkit and generated on a server which consists of the host CPU and the attached GPU. The host processor was a 10-core Intel Xeon E5-2650 at 2.3 GHz with 16 GB DDR4 RAM and 25 MB cache; the device was an NVIDIA Tesla K40c at 0.75 GHz with 11,520 MB global memory. We used the NVIDIA CUDA compiler v7.0.17 and the PGI Accelerator compilers (for OpenACC) v15.4, running on CentOS 7.1.1503.

3.2 Benchmarks selection

Benchmarks were selected from the Rodinia suite [5] and a code sample from NVIDIA. Also we developed a particular scientific computation application with an emphasis on computing speed. To get a "balanced" comparison data, we followed a similar benchmark categories standard in the paper [6] to select benchmarks from synthetic applications to real-world applications and following the guideline of the Dwarfs [7], details of categories, Dwarfs, domains and descriptions were described in Table 1 where Syn denotes a synthetic application and R-W denotes a real-world application. Finally, more details about each benchmark in the Rodinia suite are described in
the papers [6]. Following a similar explanation in the paper [6], we presented a different meaning of Real-world applications and Synthetic applications in this paper.

- Real-world applications. To solve realistic problems in other scientific research areas, such applications usually use well-known and classic algorithms to do iteration calculations.
- Synthetic applications. For the purpose of pure theoretical measurement, sometimes researchers need highly-intensive computation to explore the biggest machine performance. Here, MatMul in both CUDA and OpenACC, Jacobi (CUDA) are developed by ourselves (denoted by SELF).

4. Methodology

4.1 Data sensitivity

An interesting phenomenon was found when time was measured with different input sizes in some benchmarks

under different programming models. Time change does not follow the pace of input size change. For those well-known or classic algorithms, the time complexity can clearly tell the relationship of time and input size such as an O(n log n) in the Fast Fourier Transform(FFT) algorithm, but it cannot be perfectly applied to our benchmarks. For example, in CFD the input size is mainly showed as a file size and in some benchmarks, there are no classic algorithms. Also, in this paper we focus on the impact of different programming models on run time, rather than the algorithms' impact. We call this kind of impact Performance Ratio of Data Sensitivity (PRoDS) – how sensitive both OpenACC and CUDA programming models are to data changes in this paper specifically.

In order to objectively and quantitatively measure PRoDS, we propose the following formula to calculate the PRoDS. It is described:

$$PRoDS = \frac{|Difference_{time\_cuda}|}{|Difference_{time\_OpenACC}|}$$

If PRoDS < 1, the performance of OpenACC is more sensitive to data than CUDA; if PRoDS > 1, the performance of CUDA is more sensitive to data than OpenACC. If |1-PRoDS| < 0.1, we assume that both OpenACC and CUDA have the same data sensitivity as the definition in the paper (Fang, Varbanescu, and Sips 2011).

For the PRoDS formula, $Difference_{time\_cuda}$ means a difference of CUDA running time with different input sizes. For example, in Matrix Multiplication, if 10 seconds were spent when input size is 1024x1024 and 6 seconds were spent when input size is 512x512, $Difference_{time\_cuda}$ is 4 seconds. A similar explanation is in $Difference_{time\_OpenACC}$.

5. Data evaluation

5.1 Comparision on data sensitivity

To investigate which programming model is more stable when input size changes over a wide range, we proposed a metric of the Performance Ratio of Data Sensitivity (PRoDS) to convert a subjective measurement to an objective one. And an input size from benchmarks was listed in Table 2. Compared to the formula in [6], one obvious difference in the PRoDS formula is to add a condition to judge whether or not both programming models have the same data sensitivity, which means that PRoDS just denotes a ratio of performance changing extent. To obtain a "native" comparison of the data sensitivity, the performance in the thread mapping version of OpenACC was selected to compare to the CUDA version. Because many different input sizes were used in measuring the performance of each benchmark, we selected the "worst" gradient values as comparison samples, which have biggest values. Finally, to make a fair comparison of data sensitivity,
we divided all benchmarks into two categories: one without optimization and one with optimizations, and the PRoDS caused by different optimization technologies is ignored.

In non-optimization category, the CUDA implementation was more sensitive to data than the OpenACC implementation. This is particularly true for the first kernel in CFD (*cfd_k1*) and the second kernel in BFS (*bfs_k2*). As Figure 2 shows, in cfd_k1, the PRoDS of CUDA is 1.4 times more sensitive than one of OpenACC while in bfs_k2 CUDA is 13% more sensitive to data than OpenACC. Because the thread mapping version was used in this experiment, we explicitly added specific values for gang and vector parameter in kernel construct of OpenACC. In this way, the PGI compiler's flexibility of compiling parallelizable loops in OpenACC implementation was significantly limited. In contrast, CUDA not only allowed programmers to set up Gid/Block dimensions, but also gave programmers permission on operations of index such as *blockDim.x*, *threadIdx.x*.

In the optimization category, the OpenACC implementation was more sensitive to data than the CUDA implementation. This is particularly true for Hotspot and Lud with memory-coalescing optimization. As Figure 1 presented, in Hotspot the OpenACC implementation was 60% more sensitive to data than the CUDA one. In Lud the OpenACC implementation was 51% more sensitive to data

Table 1: Benchmarks selection

Benchmark	Suite	Category	Dwarf	Domain	Description
Jacobi	NVIDIA	Syn	Dense Linear Algebra	Image Processing	A Jacobi iteration on Laplace2D
MatMul	SELF	Syn	Dense Linear Algebra	Linear Algebra	Matrix multiplication
BFS	Rodinia	R-W	Graph Traversal	Graph Algorithms	Graph breadth first search
SRAD	Rodinia	R-W	Structured Grid	Image Processing	Removing speckles in an image
Gaussian	Rodinia	R-W	Dense Linear Algebra	Linear Algebra	Solving for variables in a linear system
Pathfinder	Rodinia	R-W	Dynamic Programming	Grid Traversal	Find a path from bottom row to top row
LUD	Rodinia	R-W	Dense Linear Algebra	Linear Algebra	Calculate solutions of linear equations
NN	Rodinia	R-W	Dense Linear Algebra	Data Mining	Find the k-nearest neighbors

Hotspot	Rodinia	R-W	Structured Grid	Physics Simulation	Thermal simulation
CFD	Rodinia	R-W	Unstructured Grid	Fluid Dynamics	A solver for Euler equations

than the CUDA implementation. The OpenACC programming style itself affected the data sensitivity. For memory coalescing, whether or not neighboring elements can be accessed mainly depends on loop order in OpenACC implementation, but the lack of architecture-specific directives in the OpenACC programming style limits programmers' ability in defining data attribution such as *shared* memory. In contrast, CUDA programmers can explicitly declare all data used as a shared type so that all results measured with different input sizes were less sensitive than the results in OpenACC implementation.

Through analyzing the PRoDS above, we concluded that the CUDA implementation was more sensitive to data than the OpenACC implementation without optimizations while the OpenACC implementation was more sensitive to data than the CUDA implementation with optimizations.

Table 2. Input size for selected benchmarks

Benchmark	Input Size
MatMul	Matrix: 1024x1024
Jacobi	Matrix: 1024x1024
BFS	4096x4096
Gaussian	Matrix: 1024x1024
Pathfinder	Row: 10000, Col:100, Iteration:100
NN	Number of records: 15690
CFD	File size: 0.2M

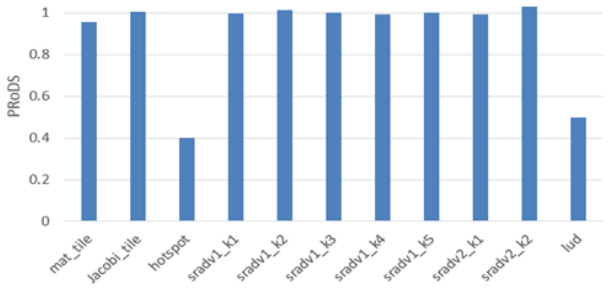

Figure 1. Performance comparison of data sensitivity with optimization

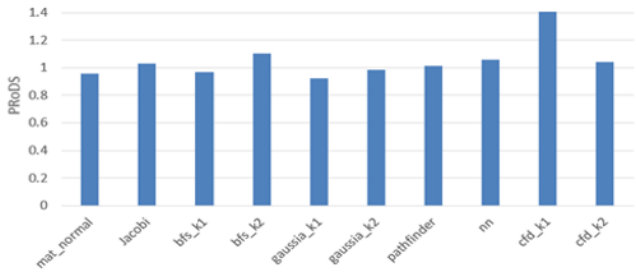

Figure 2. Performance comparison of data sensitivity without optimization

5.2 Comparision on programming model

Gang and *vector* parameters in *kernels* construct or *num_gangs* and *vector_length* parameters in parallel construct can assist OpenACC to achieve a similar dimension setup of Grid and Block in CUDA so that OpenACC and CUDA can implement and achieve similar acceleration in parallel loops. But as far as device memory hierarchy is concerned, CUDA can explicitly support specific kinds of memory in devices while OpenACC fails to do it because it is designed to be portable across platforms from multiple vendors [2]. For example, the use of architecture-specific device memory shared is impossible in OpenACC while this kind of memory can be explicitly declared and used in CUDA. Although cache directive can, to some extent, compensate performance loss caused by the lack of shared memory, the lack of barrier synchronization extremely limits effects of the use of this directive to read-only data [2].

6. Background

The goal of high-performance parallel languages is to accelerate run time and to reduce the unnecessary waiting time of computing operations and resources schedule by utilizing optimization technologies such as kernel parameters tuning, the use of local memory, data layout in memory and avoiding CPU-GPU data transfer. In this section, we simply introduce languages used in this paper: CUDA and OpenACC.

CUDA is a parallel computing programming model that can be run only under NVIDIA's GPUs. It fully utilizes hardware architecture and software algorithms to accelerate high-performance computation with architecture-specific directives such as shared, or global. But this utilization requires programmers to fully understand each detail of hardware and software. For example, in order to utilize memory-coalescing or tiling technology, the software-managed on-chip and off-chip memory knowledge should be gained by users before programming, which is a big challenge, especially for programmers who have little knowledge of software and hardware. Also, in order to successfully launch CUDA kernels, all configurations, such as Grid/Block dimensions, computing behaviors of each thread, and synchronization problems need to carefully be tuned.

OpenACC is a cross-platform programming model and a new specification for compiler directives that allow for annotation of the compute region and data region that are offloaded to accelerators [2] such as GPUs. The OpenACC Application Programming Interface [8] offers a directive-based approach for describing how to manage data and execute sections of codes on the device, where parallelism often appears in regular repetition constructs such as Fortran "DO" loops and C "FOR" loops. In OpenACC, porting of legacy CPU-base code only

requires to add several lines of annotations before the sections where they need to be accelerated, without changing code structures [2]. But over-simplification parallel programming model also brings to users limitations which may prevent full use of the available architectural resources, potentially resulting in greatly inferior performance when compared to highly manually tuned CUDA code [2]. For example, programmers can use synchronization to manage all threads of one block in CUDA while they fail in OpenACC. Some architecture-specific directives such as global, shared and private have been provided by CUDA while OpenACC does not provide them.

7. Conclusions and future work

The analysis of performance gaps has been shown above in 19 kernels of 10 benchmarks. We used kernel execution time and data sensitivity as main standards referenced when conclusions were made. A comparison of data sensitivity is a new index to explore an easily-ignored problem that how both programming models are sensitive to changes of data sizes. The appearance of the PRoDS formula brings us an objective comparison rather than a subjective comparison. Through the analysis of Figure 1 and Figure 2, the OpenACC programming model is more sensitive to data than the CUDA with optimizations while CUDA is more sensitive than OpenACC without optimizations. Overall, the OpenACC performance is similar to CUDA under a fair comparison and OpenACC can be a good alternative to CUDA especially for beginners in high-level parallel programming.

In the future work, we will continue to explore the influence of optimization techniques on CUDA and OpenACC models. Also, we will also use the autotuner [12] to optimize parameters in OpenACC so that the best performance can be used in the comparisons.

References

1. X. Li, C. H, Li. and Y. Xie. 2011. “A Retrieval System of Vehicles Based on Recognition of License Plates”. Proceedings of 2011 International Conference on Machine Learning and Cybernetics (ICMLC), IEEE, Guilin, pp.1453-1459.
2. Hoshino, T., Maruyama, N., Matsuoka, S. and Takaki, R. 2013. “CUDA vs OpenACC: Performance Case Studies with Kernel Benchmarks and a memory-bound CFD Application”. 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 136-143.
3. Herdman, J.A. Gaudin, W.P. Mclntosh-Smith, S. and Boulton, M. 2012. “Accelerating Hydrocodes with OpenACC, OpenCL and CUDA”. 2012 SC Companion: High Performance Computing, Networking, Storage and Analysis (SCC). pp. 465-471.
4. Christgau, S., Spazier, J., Schnor, B., Hammitzsch, M., Babeyko, A. and Waechter, J. 2014. “A comparison of CUDA and OpenACC: Accelerating the Tsunami Simulation EasyWave”. 27th International Conference on Architecture of Computing Systems (ARCS). pp. 1-5.
5. Che, S., Jeremy, W., Sheaffer, Michael, B., Lukasz G. S., Liang. W., and Kevin, S. 2010. "A characterization of the rodinia benchmark suite with comparison to contemporary cmp workloads", in Proceedings of the IEEE International Symposium on Workload Characterization (IISWC'10), pp. 1-11
6. Fang, J., Varbanescu, A and Sips, H. 2011. “A Comprehensive Performance Comparison of CUDA and OpenCL”. International Conference of Parallel Processing (ICPP), pp. 216-225.
7. Krste, A. Ras, B. Bryan, C. Joseph, G. Parry, H. Kurt, K. David, P. William, P. John, S. Samuel, W. and Katherine, Y. 2006. “The landscape of parallel computing research: a view from Berkeley”. Tech. Rep. UCB/EECS-2006-183, Electrical Engineering and Computer Science, University of California at Berkeley
8. The OpenACC Application Programming Interface, Version1.0, November 2011.
9. Kamran, K. Neil, D. and Firas, H. A Performance Comparison of CUDA and OpenCL.
10. http://aqua.dwavesys.com
11. Po-Yu Chen ; Chun-Chieh Lan ; Long-Sheng Huang and Kuo-Hsuan Wu. Overview and Comparison of OpenCL and CUDA Technology for GPGPU. IEEE Asia Pacific Conference on Circuits and Systems (APCCAS). pp. 448 – 451. Dec. 2012
12. Calvin,M, Jeffrey, O. and Xuechao, L. Autotuning OpenACC Work Distribution via Direct Search.To appear at Extreme Science and Engineering Discovery Enviroment (XSEDE15), Jul. 2015
13. CUDA, “NVIDIA CUDA [online]. available: http://developer.nvidia.com/category/zone/cuda-zone,” 2012.
14. OpenACC. https://www.openacc.org/
15. PGI Accelerator, “The Portland Group, PGI Fortran and C Accelarator Programming Model [Online]. Available: http://www.pgroup.com/resources/accel.htm,” 2009
16. HMPP, “HMPP Workbench, a directive-based compiler for hybrid computing [Online]. Available: www.caps-entreprise.com/hmpp.html,” 2009.
17. J. C. Beyer, E. J. Stotzer, A. Hart, and B. R. de Supinski, “OpenMP for Accelerators.” in IWOMP’11, 2011, pp. 108–121
18. T. D. Han and T. S. Abdelrahman, “hicuda: High-level gpgpu programming,” IEEE Transactions on Parallel and Distributed Systems, vol. **22**, no. 1, pp. 78–90, 2011
19. A. Leung, N. Vasilache, B. Meister, M. Baskaran, D. Wohlford, C. Bastoul, and R. Lethin, “A mapping path for multi-GPGPU accelerated computers from a portable high level programming abstraction,” in Proceedings of the 3rd Workshop on General-

Purpose Computation on Graphics Processing Units, ser. GPGPU '10. New York, NY, USA: ACM, 2010, pp. 51–61

20. J. Dongarra, P. Beckman, T. Moore, P. Aerts, G. Aloisio, J.-. Andre, D. Barkai, J.-. Berthou, T. Boku, B. Braunschweig, F. Cappello, B. Chapman, X. Chi, A. Choudhary, S. Dosanjh, T. Dunning, S. Fiore, A. Geist, B. Gropp, RobertHarrison, M. Hereld, M. Heroux, A. Hoisie, K. Hotta, Y. Ishikawa, Z. Jin, F. Johnson, S. Kale, R. Kenway, D. Keyes, B. Kramer, J. Labarta, A. Lichnewsky, T. Lippert, B. Lucas, B. Maccabe, S. Matsuoka, P. Messina, P. Michielse, B. Mohr, M. Mueller, W. Nagel, H. Nakashima, M. E. Papka, D. Reed, M. Sato, E. Seidel, J. Shalf, D. Skinner, M. Snir, T. Sterling, R. Stevens, F. Streitz, B. Sugar, S. Sumimoto, W. Tang, J. Taylor, R. Thakur, A. Trefethen, M. Valero, A. van der Steen, J. Vetter, P. Williams, R. Wisniewski, and K. Yelick, "The International Exascale Software Project RoadMap," Journal of High Performance Computer Applications, vol. **25**, no. 1, 2011