Understanding and Optimizing OpenCL Kernels on FPGAs

ABSTRACT

As a complementary guideline for the existing optimization efforts. But we argue this work should be considered as the first step to decide, as it decides the potential of other optimization combinations can reach.

ACM Reference Format:

. 2019. Understanding and Optimizing OpenCL Kernels on FPGAs. In *Proceedings of ACM Woodstock conference (DAC'19).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

The parallel programming languages, e.g., OpenCL, are based on BSP (Bulk Synchronous Parallel) model and are widely adopted by GPUs, which features a massive number of cores for the computation task. Due to the fact that FPGA is embarrassingly parallel, it is a natural attempt to use the existing parallel programming languages to program FPGA [JGL: Also because of increased programmability and lower learning curve with respect to RTL]. For instance, Intel has launched one OpenCL SDK to program FPGAs [7]. Plenty of research work [3, 9, 13] are aim at optimizing the *conventional OpenCL* kernels on FPGAs, where the conventional OpenCL kernel is the NDRange kernel which employs a multi-thread approach to explore the thread-level parallelism to accelerate the computing task.

[JGL: I think an interesting motivation for this work can be achieving performance portability across different types of accelerators (i.e., GPU to FPGA). In this regard, a paper to cite is [2].]

[JGL: Somewhere in the introduction, it is necessary to talk about the characteristics of the GPU OpenCL codes that are going to be ported to FPGA in this work. In Chai [4], there are benchmarks that use inter-kernel communication (e.g., CEDD) and multi-pass schemes (e.g., BFS, SSSP). These two are typical techniques in GPU programming. In Chai, there is also extensive use of atomic operations. In the past, GPU programmers avoided atomic operations [10] because they had high overhead. However, in recent GPU generations (e.g., NVIDIA Kepler, Maxwell, Pascal) the hardware has greatly improved. They provide improved programmability that GPU programmers nowadays leverage.]

Despite the preliminary success from directly adopting conventional OpenCL programs on FPGAs, we still identify a significant gap from achieving the near-to-optimal performance on FPGAs due to the factor that the conventional OpenCL kernel is not able to fully utilize two distinguished architectural features of FPGAs [JGL: Any related work that has compared the performance of optimized RTL with OpenCL?]. We refer this kind of direction as a go-beyond OpenCL on FPGAs.

F1: Single Work-Item (SWI) Kernel. Intel OpenCL SDK provides a new programming model for FPGAs: *single work-item kernel*. It relies on the off-line compiler to explore the pipelined parallelism at the compilation time. Therefore, it only contains only one work item to do the computation, indicating significantly different from data-parallel programming model of conventional OpenCL kernel.

F2: Direct Kernel-to-Kernel Communication. The communication between two kernels can be done via a Fifo (i.e., OpenCL channel) on FPGAs, not via memory subsystem like GPUs. It can potentially reduce memory traffic and achieve more parallelism by concurrently executing multiple OpenCL kernels on FPGAs.

Actually, several research work [1, 12] have already employed the above two techniques to significantly accelerate a broad range of OpenCL applications. For instance, the performance of database partitioning [12] is improved by 10.7 times, indicating a great potential of go-beyond kernel on FPGAs. Together with the optimization methods from conventional OpenCL kernel, a huge design space needs to be explored to determine the optimal (or near-to-optimal) optimization combination on FPGAs. Unfortunately, there is no comprehensive study to guide how to optimize the OpenCL kernels on FPGAs, especially about go-beyond OpenCL kernels. Therefore, the burden of choosing the right OpenCL optimization methods still falls on the users without any rule-of-thumb guidelines. In this paper, we try to answer the following question: *Can we narrow the gap between conventional OpenCL programmer and performance potential of FPGAs?*

In this paper, we make the following contributions to narrow the gap for the conventional OpenCL programmer by connecting three common OpenCL patterns and four high-level execution models.

[JGL: To me, the most significant contribution of this work can be providing some guidelines on how to transform optimized GPU OpenCL code into optimized FPGA OpenCL.]

C1: Three OpenCL Patterns on FPGAs (Section 4). We identify three popular patterns from OpenCL kernels: atomic operation, multi-pass scheme and kernel-to-kernel communication, which are worth to be heavily revisited on FPGAs, since their performance can be significantly improved with the help of go-beyond-OpenCL features enabled by FPGAs. These patterns are well understood by the conventional OpenCL programmers such that they can easily identify the potentials of their original OpenCL program when mapping them onto FPGAs.

C2: Four High-level Execution Models on FPGAs (Section 5). We identify four high-level OpenCL execution models: NDRange, SWI, NDRange+Channel and SWI+Channel, which serve as main guidelines on how to optimize the OpenCL program on FPGAs. Choosing the right high-level execution model is vital for OpenCL program to achieve excellent performance on FPGAs. We also generally compare the characteristics of four execution models, which are aware of the advanced features of go-beyond OpenCL kernel.

DAC'19, June 2019, Las Vegas, NV, USA

ACM ISBN 123-4567-24-567/08/06.
https://doi.org/10.475/123_4

C3. Intensive Experiment to Verify Our Observation (Section 6). We implement and optimize 11 OpenCL applications on a Terasic's DE5A-Net Arria 10 FPGA board. Actually, we try our best to implement all the execution models[1], each of which contains multiple optimization combinations, for each application. Based on the experimental result, we can make two observations. First, different high-level execution model leads to up to two orders of magnitude performance difference[2], as shown in Figure **??**. Second, we identify a obvious connection between three proposed OpenCL channels and four high-level execution models.

Future Directions and Limitation. To the best of our knowledge, this paper is the first attempt to narrow the gap between conventional OpenCL programmer and FPGAs, and we believe it opens a few exciting research directions which aim at closing the gap so that it will attract more and more people to devote to FPGA acceleration. One interesting future direction is to provide an end-to-end performance analytical model to predict the performance of each execution model. Unfortunately, only the NDRange kernel is well studied in terms of the analytical model [9, 13]. Another interesting future direction is to design an automatic optimization process for each execution model with the help of performance analytical model. As such, more and more people without any FPGA background can take advantage of computing power of FPGAs.

The limitation is that we manually try plenty of optimization combinations for each execution model.

2 BACKGROUND

2.1 Traditional OpenCL Programming Model

2.1.1 OpenCL Concepts. OpenCL is an open parallel programming language for heterogeneous computing environments. It aims for a host-accelerator model of program execution, where a host processor runs control-intensive task and offloads computation-intensive code (i.e., *kernel*) onto an external accelerator.

Recently, Altera provides the OpenCL SDK to abstract the hardware complexities from the traditional FPGA design. The Altera's SDK can translate the high-level OpenCL description to low-level hardware implementation by creating the circuits for each operation of the kernel and interconnect them together to achieve the whole data path.

From the perspective of OpenCL, the FPGA memory hierarchy contains three layers. First, the *global memory* with high latency and low bandwidth which often refers to the DDRs of the FPGA board. Second, the *local memory* utilizes on-chip low-latency and high-bandwidth BRAMs, and it has four banks in general. Third, the *private memory*, storing the variables or small arrays for each work-item, is constructed using completely-parallel registers. Compared to CPU/GPU, the private memory is rather plentiful in FPGA.

Different from viewing FPGAs as pure hardware resources like LUTs, DSP blocks and memory blocks, the OpenCL SDK regards the FPGA as a large-scale parallel architecture. Most commonly, OpenCL partitions a problem into equally loaded chunks of work, i.e., *work-item*, which represent the basic unit of execution. A set of work-items are organized into a *work-group* and multiple work-groups are combined to form a three-dimensional index space called *NDRange*. *NDRange kernel* is the default OpenCL kernel model which achieves the pipelined parallelism across multiple work-items. We can configure multiple kernel pipelines, i.e., Compute Units (CUs) for the NDRange kernel. Then each CU executes the OpenCL program in a pipelined fashion. Although NDRange execution pattern is the most common pattern of using accelerators with OpenCL, it is not mandatory. Actually, in many cases, single work-item kernels are employed as a preferred execution pattern in Altera's OpenCL implementation as explained below.

2.1.2 Main Optimization Methods. We introduce three main optimization methods, which are widely adopted by normal and go-beyond OpenCL kernels. We refer the reader to the related work [13] for more optimization methods on FPGAs.

Shared Memory. Each CU has its own local memory, which is shared by all the work-items assigned to the CU, while all CUs share a global memory. The on-chip local memory is low-latency and high-throughput compared to global memory. Thus, the local memory can be employed to reduce the number of global memory accesses.

Loop Unrolling. If there are a large number of loop iterations in the kernel pipeline, the loop iterations could potentially be the critical path of the kernel pipeline. Unrolling the loop can increase the pipeline throughput at the expense of more hardware resources consumption. And it may have a side-product benefit on FPGAs. The load/store operations can coalesce so that the number of global memory accesses is reduced.

Multiple Compute Units. If the hardware resource is sufficient on the FPGA, the kernel pipeline can be replicated to generate multiple CUs for each kernel and the inner hardware scheduler automatically dispatches work-groups to available CUs. Kernel pipeline replication can achieve higher throughput at the expense of increasing hardware resource utilization, which often leads to a lower FPGA operating frequency than that of one CU. It means that two CUs cannot always double the performance. Another issue is that the global memory load/store operations from multiple CUs compete for the global memory bandwidth and the total number of global memory operations remains unchanged.

2.2 Go-beyond OpenCL on FPGAs

2.2.1 Single Work-Item Kernel. In addition to the standard NDRange kernel model, Altera OpenCL SDK also provides a *single work-item kernel* model. It follows a sequential model like C programming, thus executes the kernel on only one CU that contains only one work-item. In the single work-item execution pattern, the parallelism is implicit, and the OpenCL SDK will instead determine pipelined parallelism at the compilation time based on the dependency. The single work-item execution pattern more closely matches the traditional deep-pipeline approach of programming FPGAs.

2.2.2 OpenCL Channel. Altera OpenCL SDK provides the *OpenCL channel* feature, which can be used to pass data between two

[1]For few applications, it is impossible to implement few execution models.
[2]Based on our five years' experience on OpenCL-based FPGAs, we are pretty sure that our implementation have typically reached the near-to-optimal performance for each execution model. We will make all the related source code open-sourced.

OpenCL kernels (either NDRange or single work-item) and synchronize the kernels with high efficiency and low latency. In the conventional OpenCL implementation, the communication between two kernels is performed via global memory. However, global memory bandwidth could potentially be a performance bottleneck for the kernels. In contrast, the implementation of channels decouples kernel execution from the host processor, allowing the kernels to communicate directly with each other via on-chip FIFO buffers.

The OpenCL channel has two types: blocking channel and nonblocking channel. The write/read operation to blocking channel will not return if the operation does not successfully commit, while the write/read operation to nonblocking channel will return even when the operation does not successfully commit.

3 DESIGN OVERVIEW

In this section, We illustrate how the next three sections organized. Bridge the gap between OpenCL implementation and FPGAs.

4 OPENCL ON FPGA: ISSUE AND POTENTIAL

When migrating conventional OpenCL programs, which is originally designed for GPUs, to FPGAs, we identify three OpenCL patterns which can be significantly improved: atomic operation, multi-pass scheme and communication approach. In the following, we analyze the issue of each aspect on FPGAs, followed by the potential optimization direction on FPGAs.

4.1 Atomic Operation

For NDRange OpenCL, the atomic operation is required to guarantee the consistence of OpenCL kernel, when multiple work items try to update the same memory location, e.g., atomic_add [8, 11].

Issues on FPGAs. We have identified three issues regarding atomic operation on FPGAs. First, the mechanism of atomic operation is relatively complex, so massive FPGA resources are required to implement it, leading to a noticeable resource overhead on FPGAs. Second, with atomic operation in our OpenCL kernel, all the memory transactions, including both atomic and normal memory transactions, have to enter the atomic module to guarantee the correct execution, as atomic and normal memory transactions from OpenCL kernel can reference the same memory location [JGL: Every access goes through the atomic path even though it is clear that there are no atomic accesses to certain arrays? This is weird, but reminds me that something similar happened in old AMD GPUs]. Therefore, each memory transaction has a longer memory access latency, leading to potentially lower memory bandwidth. Third, to make things worse, the frequency of FPGA design is much lower than that on ASICs. We conclude that the atomic operation is not able to fit well on FPGAs.

Potential on FPGAs. In order to achieve good performance of OpenCL kernel on FPGAs, one potential direction is to get rid of atomic operations. Fortunately, Intel OpenCL SDK [7] supports a new execution model: single work-item kernel. Essentially, it contains only one work item during the kernel execution, so there is no conflict from multiple work items, indicating that no atomic operation is required. In other words, it exploits the pipelined parallelism, not the thread-level parallelism which is widely exploited by GPUs.

4.2 Multi-pass Scheme

Typically, the multi-pass scheme is widely adopted by OpenCL kernel, since it enables a massive amount of cores in CPUs/GPUs to accelerate parallel algorithm, e.g., database operators [6, 14, 15]. Further, it improves the cache locality on GPUs, e.g., gather/scatter [5], so as to improve the overall performance at the cost of more memory traffic. We observe that the inevitable requirement for the adoption of multi-pass scheme is powerful memory subsystem, as the multi-pass scheme always requires more memory traffic to realize the algorithm. It works pretty well on GPUs, since the memory bandwidth of GPUs is almost an order of magnitude larger than the same-generation CPU or FPGA. For example, the memory bandwidth of Nvidia Tesla P100 GPU is able to reach 732GB/s, while Intel Skylake i9-7980XE CPU has the memory bandwidth of 85GB/s and Xilinx UltraScale+ FPGA board has 48GB/s.

Issues on FPGAs. We can predict that the multi-pass scheme, whose success heavily relies on high memory bandwidth, would be less popular on FPGAs than on GPUs, since the memory bandwidth of FPGAs is relatively low, e.g., 18GB/s on our tested FPGA board. Therefore, the multi-pass scheme of OpenCL kernel can suffer from its severe performance issue, especially for the memory-bound OpenCL kernel on FPGAs, even though the FPGA can provide massive thread-level parallelism.

Potential on FPGAs. Since OpenCL SDK for OpenCL supports the new single work-item kernel. Thus, the application which is originally optimized with multi-pass scheme is allowed to be implemented in a single-pass approach, without compromising any achievable pipeline parallelism. Therefore, the memory traffic is reduced to a minimum level just as the single-threaded implementation running on a single core.

Todo: add one concrete example to show the difference, like database filter.

4.3 Kernel-to-Kernel Communication

On GPUs, the typical communication approach between producer/-consumer kernels is done via global memory. In particular, after the producer kernel writes the data to the global memory, the consumer kernel reads the data from global memory. It is a common communication approach to exchange information among Stream Processors (SPs) in GPUs, as there is no physical communication path between any two SPs. This communication approach works well on GPUs since GPUs can execute each kernel relatively fast due to its massive thread-level parallelism and powerful memory subsystem. Suppose each OpenCL kernel is fully optimized, the overall performance of producer/consumer kernels is maximized. This communication approaches widely used in accelerating various applications, e.g., database [6, 15].

Issue on FPGAs. The memory bandwidth, e.g., 18GB/s on our tested FPGA board, is relatively low on FPGAs, the communication via external memory can be extremely expensive.

Potential on FPGAs. We can use OpenCL channel via which the producer kernel can directly send data to the consumer kernel at the register level (i.e., FIFO), without any memory traffic involved between the producer and consumer kernels. Besides, both kernels, each of which has the dedicated hardware resources to implement, execute concurrently, so not only intra-kernel parallelism

	NDRange	SWI	NDRange+Channel	SWI+Channel
Programmability	Moderate	High	Low	Low
Computing Parallelism	Moderate	Low	High	High
Memory Traffic	High	Low	Moderate	Low

Table 1: General comparison of execution models

(i.e., pipelined) is exploited, but also inter-kernel parallelism (i.e., multiple kernels).

5 OPENCL EXECUTION MODELS ON FPGA

In this section, we explore the design space of execution models of OpenCL kernel on FPGAs. In the following, we generally analyze the pros and cons of each execution model.

5.1 NDRange

It is the conventional OpenCL programming model, which is widely used in GPU programming. It explore massive theread-level parallelism.

Pros. The optimization methods which are used by GPUs can also apply to FPGAs, like SM, memory coalescing. Plenty of related work is available.

Cons. It may not achieve good performance on FPGAs, whose architecture is significantly different from GPUs. Suppose it works well on FPGAs, it also works well on GPUs.

Scope. Without any above three issues.

5.2 Single Work-item

Only one work-item is alive during the execution.

Pros. Easy to write the program.

Cons.

Scope. With Atomic or multi-pass, or both. But it works only for a simple OpenCL kernel, since it can only explore limited parallelism.

5.3 NDRange + OpenCL Channel

Pros. More information

Cons. Additional information, e.g., work item id, is communicated to guarantee the correctness. Otherwise, the producer/consumer execution pattern, enabled by OpenCL channel, may run out-of-order.

Scope. For compute-bound application, needs more data parallel.

5.4 Single Work-item + OpenCL Channel.

Pros.

Cons.

6 EXPERIMENT

Hello Introduction.

Hypotheses to validate: 1, different execution patterns lead to significantly different performance.

7 CONCLUSION

Hello Conclusion.

Take-away messages:

1, Direct adoption of OpenCL does not work well on FPGA. It does not take advantage of FPGA properties.

2, The necessary background of FPGA is required to guarantee the good performance of OpenCL kernels on FPGAs.

3, for the beginner, OpenCL is much easier than RTL .

Two observations are orthogonal.

1, Multi-pass and atomic operations can benefit from single-work item.

2, Multiple kernels (producer-consumer) can benefit from channel.

3, NDRange can explore more pipelined parallelism than single-work-item. (Using one example (MM) to show).

8 ACKNOWLEDGEMENT

We thank Intel which has donated Terasic's DE5A-Net Arria 10 FPGA board for our research.

REFERENCES

[1] M. S. Abdelfattah, A. Hagiescu, and D. Singh. Gzip on a chip: High performance lossless data compression on fpgas using opencl. In *IWOCL*, 2014.
[2] L.-W. Chang, J. Gómez-Luna, I. El Hajj, S. Huang, D. Chen, and W.-m. Hwu. Collaborative computing for heterogeneous integrated systems. In *International Conference on Performance Engineering (ICPE), 8th ACM/SPEC*. ACM/SPEC, 2017.
[3] T. S. Czajkowski and et al. Opencl for fpgas: Prototyping a compiler. In *ERSA*, 2012.
[4] J. Gómez-Luna, I. El Hajj, V. Chang, Li-Wen Garcia-Flores, S. Garcia de Gonzalo, T. Jablin, A. J. Pena, and W.-m. Hwu. Chai: Collaborative heterogeneous applications for integrated-architectures. In *Performance Analysis of Systems and Software (ISPASS), 2017 IEEE International Symposium on*. IEEE, 2017.
[5] B. He, N. K. Govindaraju, Q. Luo, and B. Smith. Efficient gather and scatter operations on graphics processors. In *SC*, 2007.
[6] B. He, M. Lu, K. Yang, R. Fang, N. K. Govindaraju, Q. Luo, and P. V. Sander. Relational query coprocessing on graphics processors. *TODS*, 2009.
[7] Intel. Intel SDK for OpenCL Optimization Guide. 2018.
[8] Khronos OpenCL Working Group. The OpenCL Specification. 2009.
[9] Y. Liang, S. Wang, and W. Zhang. Flexcl: A model of performance and power for opencl workloads on fpgas. *TC*, 2018.
[10] R. Nasre, M. Burtscher, and K. Pingali. Atomic-free irregular computations on gpus. In *Proceedings of the 6th Workshop on General Purpose Processor Using Graphics Processing Units*, GPGPU-6, pages 96–107, New York, NY, USA, 2013. ACM.
[11] N. Ramanathan, J. Wickerson, F. Winterstein, and G. A. Constantinides. A case for work-stealing on fpgas with opencl atomics. In *FPGA*, 2016.
[12] Z. Wang, B. He, and W. Zhang. A study of data partitioning on opencl-based fpgas. In *FPL*, 2015.
[13] Z. Wang, B. He, W. Zhang, and S. Jiang. A performance analysis framework for optimizing opencl applications on fpgas. In *HPCA*, 2016.
[14] Z. Wang, J. Paul, H. Y. Cheah, B. He, and W. Zhang. Relational query processing on opencl-based fpgas. In *FPL*, 2016.
[15] S. Zhang, J. He, B. He, and M. Lu. Omnidb: Towards portable and efficient query processing on parallel cpu/gpu architectures. *VLDB*, 2013.