Optimizing OpenCL applications on Xilinx FPGA

[Extended Abstract of Technical Presentation]

Jeff Fifield Ronan Keryell Hervé Ratigner
Henry Styles Jim Wu
Xilinx, USA

ABSTRACT

In this presentation we focus on current Xilinx FPGA (Field-Programmable Gate Array) platforms with the SDAccel OpenCL environment. FPGA have the unique feature of a reconfigurable architecture by opposition to CPU, GPU or DSP which have a fixed architecture and are only programmable. For example the elementary functions in an FPGA can be configured according to an addressable memory, as such the interconnection among them, the internal memory organization, but also even the ultra high-speed input/output of the chip to interface with the outside world. This fine grain configurability allows high performance and power efficiency. We introduce the architecture of modern FPGA with their main building blocks and how functional operations can be expressed. The translation of imperative languages down to the hardware level is done through High-Level Synthesis. It can be done in several ways with different time/surface trade-off, for example by playing on parallelism and pipelining.

CCS Concepts

•General and reference → Performance; •Computing methodologies → Parallel programming languages; Concurrent programming languages; •Hardware → Reconfigurable logic and FPGAs; Hardware accelerators; Reconfigurable logic applications; *Hardware-software codesign; Best practices for EDA;*

Keywords

OpenCL; FPGA; Optimizations

1. INTRODUCTION

OpenCL is a portable standard from Khronos Group with an API to execute kernel programs on some heterogeneous accelerators. Kernels are written in OpenCL C, C or C++ and can run under the control of a host program on various architectures: CPU, GPU, FPGA, DSP...

IWOCL '16 April 19–21, 2016, Vienna, Austria

ISBN 978-1-4503-4338-1/16/04.
DOI: 10.1145/2909437.2909447

Unfortunately, as for a sequential program running on a CPU, there is no performance portability when it is about reaching the maximum performance and power efficiency on a given architecture. Some execution parameters may have to be tweaked (as simple as changing the work-group layout) or the architecture of the software has to be deeply changed accordingly. Since there are more parameters under control in an heterogeneous platform compared to a CPU, the exploration space is quite wider. But the OpenCL model provides a consistent framework allowing this design exploration.

In this presentation we focus on current Xilinx FPGA (Field-Programmable Gate Array) platforms with the SDAccel environment, the OpenCL implementation from Xilinx. FPGA have the unique feature of a reconfigurable architecture by opposition to CPU, GPU or DSP which have a fixed architecture and are only programmable. For example the elementary functions in an FPGA can be configured according to an addressable memory, as such the interconnection among them, the internal memory organization, but also even the ultra high-speed input/output of the chip to interface with the outside world. This fine grain configurability allows high performance and power efficiency.

We introduce the architecture of modern FPGA with their main building blocks and how functional operations can be expressed. The translation of imperative languages down to the hardware level is done through High-Level Synthesis. It can be done in several ways with different time/surface trade-off, for example by playing on parallelism and pipelining.

2. OPTIMIZATION OF OPENCL ON XILINX FPGA

The OpenCL model defines the concept of work-group (coarse-grain parallelism) with some work-items (fine-grain parallelism). SDAccel synthesizes 1 hardware compute-unit per work-group. The number of work-groups is the first way to control the amount of generated hardware for a kernel. OpenCL defines also a hierarchy of different memories that are mapped on the different memory elements found on the FPGA with different performance, size and power-consumption trade-off.

On heterogeneous platform, optimization is mostly done by managing carefully the memory hierarchy to minimize expensive memory transfers by recycling data while doing computations with intensive parallelism. It often requires reorganizing the computation with some loop transformations such as loop tiling and loop fusions. All these transformations are of course also the keystones of FPGA optimizations, but on FPGA there are quite more opportunities to play with.

But since FPGA are dynamically reconfigurable, it is possible to load different kernels in the FPGA according to the different phases of the application execution, to have the best optimized kernels for each phase.

Loop unrolling is a classical loop transformation to use more resource with less control, leading to potentially faster execution. The compiler has also more opportunities to do optimizations and vectorization. It can be done manually but SDAccel recognizes some attributes such as `__attribute__((opencl_unroll_hint(...)))` to do it in a simpler way.

A classical optimization in HPC back from the 70's with CDC and Cray computers is the pipelining of loop iterations. On hardware architectures it is very interesting because it requires only adding cheap registers to increase dramatically the throughput at the price of a moderate latency degradation. It can be done manually in a cumbersome way but can be done gracefully by using 2 different attributes, to pipeline at the work-group level `__attribute__ ((xcl_pipeline_workitems))` or more precisely at the loop level with `__attribute__((xcl_pipeline_loop))`.

As on other accelerators, carefully choosing where the data are allocated is very important. On a GPU, a common issue is to have some external memory bank conflicts, some conflicts on the internal crossbar or some conflicts on the local memory banks. Interestingly, an FPGA is programmable at the hardware level, so why not changing the architecture itself to avoid these conflicts for a specific application? This can be done for example by selecting the number of memory ports allocated to some kernel arguments or even defining exactly the memory layout of some local arrays `__attribute__((xcl_array_partition(...)) )` in block/cyclic/block-cyclic down to 1 register per array element for crazy memory need without any conflict, but at the price of more hardware resource usage.

Often applications can be decomposed in a data-flow or functional programming way, with consumer kernels of data produced by several kernels. The best optimization is to fuse the kernels to eliminate the costly data transfers when possible. If not possible, a first optimization is to use internal memory for the transfers or even OpenCL 2.0 pipes. Pipes are FIFO to send data between producer/consumer. It is very interesting on FPGA because it can be efficiently implemented in hardware and avoid communication through global memory compared to GPU. Unfortunately, OpenCL pipes are not very useful yet because of the OpenCL 2.0 execution model: since there is no independent forward progress guaranty in OpenCL 2.0, it may not possible to run at the same time a producer and a consumer. But since FPGA are programmable, SDAccel provides these features by having all the related kernels implemented in hardware and running at the same time without dead-locking on the pipe resources. So it is possible to write OpenCL programs on FPGA with some data-flow applications started by the host and running forever, such as networking, signal or video processing, without any further host interaction. This leads to very high performance and power efficiency.

Another interest of pipes is to use them to process data directly from I/O like 10 Gb/s Ethernet or HDMI in a streamlined data-flow way, minimizing the hardware and host interaction.

As a way to improve locality and better use hardware data transfers (DMA) on some specific bus such as connecting the DDR4 external memory storing OpenCL global buffers, the OpenCL kernel function `async_work_group_copy()` is very useful to transfer data between local and global storage at full speed and implement transfer and computation overlapping.

Vectorization by using some OpenCL vector types is helpful to take advantage of the full width of some external memory bus instead of accessing them just with the width of some scalar type, resulting in a waste of potential bandwidth.

For some extreme optimization cases, for example when the full hardware control and optimization is required or when some specific hardware components have to be used from inside a kernel, a kernel can be written with some C/C++ languages or even with lower level VHDL/Verilog. It is similar to the existing concept of built-in or native kernels in OpenCL but with the ability to provide its own crafted implementation. The advantage is that even if the kernel is very specifically optimized, it still benefits from the higher-level OpenCL host API for the integration in the whole application.

Of course all these optimizations are eased with the help of the profilers, performance simulators and emulators provided in the SDAccel development environment. This provides a good trade-off between precision, compilation and execution time since unfortunately on FPGA, the price for this extreme versatility is the important time of compilation when going down to the real hardware.

3. CONCLUSION

FPGA are rather new comers in the world of generic heterogeneous accelerators with the availability of new higher-level programming models such as OpenCL or C/C++ with `#pragma` for FPGA and not only GPU. The development of higher level languages such as OpenCL C++ and SYCL 2.2 or OpenMP 4.5 will help their spread. The ability to craft an application down to the hardware level can lead to very high-performance and power-efficient execution. This opens a new GP-FPGA revolution in the same way we saw the GPU to GP-GPU (General-Purpose GPU) revolution 10 years ago.