Advantages of High-Level Synthesis in an OpenCL Based FPGA Programming Methodology

ALEXANDROS BARTZAS, GEORGE E. ECONOMAKOS and DIMITRIOS J. SOUDRIS
National Technical University of Athens

1. EXTENDED ABSTRACT

OpenCL (Open Computing Language) is an open standard for the development of parallel applications on a variety of heterogeneous multi-core architectures. Its execution model consists of a *host machine* connected and controlling a *compute device*, which performs calculations with a number of parallel computational intensive *kernels*. Since its introduction, it has been reported to support different CPUs, DSPs and GPUs, in a variety of heterogeneous configurations. Recently, the technological advances in *Field Programmable Gate Array* (FPGA) devices, with hundreds of GFLOPs, maximum power efficiency and low cost, has turned the parallel processing community towards them, with a number of publications proposing OpenCL as a programming language for FPGAs also. FPGAs, parallel processing and GPUs co-existed for many years, however they were considered isolated and disjoint fields, offering optimizations at different levels in system design. While indeed parallel processing is aiming at a higher optimization level than FPGAs, the main reason for this isolation has been the different programming models and languages used in each case, with FPGA programming considered a difficult and delicate job. This is starting to change however. As FPGAs made hardware design wider accepted (compared to ASICs), a new generation of FPGA programming tools based on *High-Level Synthesis* (HLS) [Coussy and Morawiec 2008], like Calypto's CatapultC, Xilinx's Vivado HLS, Cadence's C-to-Silicon and Synopsys' Synphony (to name just a few), is promising to bring hardware closer to software.

This paper presents a methodology for the adoption of OpenCL as an FPGA programming environment, based on CatapultC. CatapultC accepts C/C++/SystemC behavioral untimed system descriptions that should follow specific coding guidelines, and through a number of directives (or GUI commands) applies HLS transformations to produce optimized bit-accurate *Register Transfer Level* (RTL) architectural descriptions. The methodology of this paper is a systematic application of each HLS transformation, by a meta-engine placing and tuning CatapultC directives into OpenCL code. The main concern in this process is that even though CatapultC can produce hardware from C, efficient hardware needs effort and architectural synthesis expertise. This expertise is coded in the meta-engine, which iterates through different possible and feasible HLS directive applications to generate optimal hardware implementations of OpenCL kernels. With this approach, the opportunities as well as the obstacles imposed to the application developer by the FPGA computing platform and the adoption of C/C++ as input language are investigated, and a systematic way to explore instruction-level, data-level and thread-level parallelism is given. Furthermore, HLS offers deep design space exploration opportunities and is not used as a special purpose, one phase compiler, passing from software to hardware.

The advantages offered with the proposed methodology cover both fields of parallel processing and hardware design. First, by using OpenCL, a programmer can fine tune its algorithm for parallel processing, take simulation results and make critical decisions about data-level and thread-level parallelism. Second, a hardware designer can take kernels and produce optimized implementations through HLS, making critical decisions about instruction-level parallelism and FPGA device limitations. Finally, the proposed methodology uses a common input language for the whole development cycle, which can improve collaboration, ease integration of CPUs, DSPs, GPUs and FPGAs into a common platform and reduce application development time and cost (which was one of the main goals of the recent DARPA HPCS [Dongarra et al. 2008] program).

Compared to recent publications that are considering using parallel programming models (OpenCL and CUDA) as a programming language for FPGAs also, our work takes full advantage of HLS. In [Mingjie et al. 2010] and [Owaida et al. 2011] two methodologies are given for mapping OpenCL kernels to reconfigurable hardware. The methodologies involve compiler optimizations that map kernel code into fixed hardware templates, which are then written in hardware description languages. While both methodologies are complete and cover many different issues (computations, memory hierarchies and interfacing), the resulting hardware cores are template-based and do not cover in detail lower level design issues. In [Jaaskelainen et al. 2010], the authors present another similar methodology, targeting *Application-Specific Processors* (ASPs). They use a custom design environment and map OpenCL kernels into either common or custom ASP instructions. Another approach, closer to this paper is reported in [Papakonstantinou et al. 2009], where CUDA code is passed though another HLS tool. Directives and pragmas are used to control the tool but no systematic and iterative application is reported, as in the proposed methodology. HLS is rather considered as a single pass procedure. From the industrial point of view, FPGA vendors have been actively involved in the use of OpenCL for FPGA programming (Altera SDK for OpenCL [Czajkowski et al. 2012]), offering a specific framework that takes advantage of the parallelism expressed in OpenCL code and utilize a custom HLS step. As in [Papakonstantinou et al. 2009] however, no systematic HLS design space exploration is performed. On the contrary, HLS is considered a time consuming task in the whole design process so, HLS iterations are avoided. On the contrary, our work is based on HLS iterations for better design space exploration and improved instruction-level parallelism opportunities.

The main idea of this paper is the proposal of a semi-automated methodology to translate OpenCL code into a form suitable for CatapultC, with which hardware is synthesized using HLS. Since OpenCL is based on C99, which is also recognized by CatapultC, this translation does not bring major changes to the input code. The whole process is performed by a custom source-to-source translator (at this time implemented as a preliminary version through script files), that either infers (if possible) or accepts by the user (this justifies the term semi-automated) details to OpenCL code like pointer sizes, loop boundaries, input parameters and expected return values. The basic steps are the following.

—Each kernel is isolated and HLS synthesizes a hardware components for it.
—Pointers used as formal parameters in functions are converted to arrays with specific dimensions, for correct memory allocation.
—Return values are inserted as formal pointer parameters in the kernel function. This coding technique generates output registers for them.
—Barrier OpenCL instructions are converted into CatapultC I/O transactions with ready/acknowledge interfaces.
—Array sizes are enlarged to reach powers of 2, when feasible. This simplifies synthesis of memory access related hardware.
—Data types are changed into bit accurate and simulation efficient types supported by CatapultC.
—Conditional statements are supplemented so that all mutually exclusive paths are clearly defined. This helps CatapultC schedule them correctly.
—OpenCL specific directives are temporary removed. They are taken into account later, during system integration.
—CatapultC pragmas and directives are inserted. These pragmas and directives control all HLS transformations.

After translation, an iterative procedure is initiated, which works as a meta-engine modifying CatapultC pragmas and directives. At each iteration, which is performed with a predefined scenario (i.e. a loop's initiation interval is decreased by one in each meta-engine iteration), a new solution is produced. The meta-engine finishes when no new solutions can be produced (further modification of pragmas and directives produces invalid solutions) and the best solution with respect to performance and resource usage is selected for FPGA implementation.

In order to prove the efficiency of our methodology and support our main differentiating characteristic (HLS based design space exploration), a number of OpenCL kernels found in the NVIDIA OpenCL SDK version 4.1 have been synthesized, the parallel matrix multiplication, parallel 2D discrete cosine transform (DCT) and parallel 2D inverse discrete cosine transform (IDCT). Table I shows different solutions achieved (through different HLS optimizations) for the parallel 2D DCT in terms of maximum performance (as throughput period in ns, the time required before a new input set can be processed by the resulting pipeline architecture) and required FPGA resources (Look-Up Table (LUT) function generators, D-type Flip-Flops (DFF), Block RAM (BRAM) and special purpose DSP blocks). For all solutions, the largest FPGA of the Xilinx Virtex-6 family was used, the 6VLX760 (with 758784 LUTs, 948480 DFFs, 720x36KB BRAM and 864 DSPs) at 200MHz. Solutions S1, S2 and S3 work directly with global memory and utilize fast BRAMs (nonzero in BRAM column), which is a common block for all kernels. This offers advantages at the circuit level (smaller memory controllers, less DFFs) but performance is low because of the large number of global memory accesses (barrier commands blocks every kernel before writing its result). Furthermore, solution S1 corresponds to no HLS optimizations selected, solution S2 corresponds to main loop initiation interval set to 4 (the minimum achieved) and solution S3 corresponds to minimum initiation interval and full loop unrolling. Solutions S4 and S5 are use the same HLS optimizations with S2 and S3 but utilize double width local memories (64 bit I/O ports with 32 bit operands) and solution S6 is like S5 with subfunctions implemented as hardware components and not as inlined code. As it can be seen, each solution offers specific advantages and disadvantages and the different HLS optimizations performed can greatly improve performance and area requirements. Solutions S1 or S2 are the slowest. All other solutions are sorted so that each one is better than the previous with respect to performance. Looking at resources, in many solutions less than 1% of the available hardware is used, so there is room to implement large number of kernels. The only resources that limit the number of kernels are the DSP blocks, which increase up to a significant percentage as more parallelization is attempted. Furthermore, preliminary comparisons between a system composed of the above fastest kernel (solution S6) at 600MHz clock speed and a system based on the Radeon HD 6970 GPU at 850MHz, show a system level speedup ranging from 1.8x (256x256 image size) to 3.4x (2048x2048 image size) [Bartzas and Economakos 2012]. Also, a system with S6 kernels (larger and thus fewer, but optimized) is found to be faster than one with S1 (more but not optimized), which is a justification of our approach.

Table I. Parallel discrete cosine transform

Sol.	Perf. (ns)	LUTs	DFFs	BRAMs	DSPs
S1	455	4158	1702	1	37
S2	640	4194	2084	1	48
S3	110	3563	2354	1	23
S4	30	3649	2261	0	46
S5	15	5273	4339	0	62
S6	10	5453	6292	0	64

REFERENCES

BARTZAS, A. AND ECONOMAKOS, G. 2012. Methodology for Efficient Use of OpenCL, ESL and FPGAs in Multi-Core Architectures. In *5th Workshop on UnConventional High Performance Computing*.

COUSSY, P. AND MORAWIEC, A. 2008. *High-level Synthesis: From Algorithm to Digital Circuit*. Springer-Verlag.

CZAJKOWSKI, T. S., AYDONAT, U., DENISENKO, D., FREEMAN, J., KINSNER, M., NETO, D., WONG, J., YIANNACOURAS, P., AND SINGH, D. P. 2012. From OpenCL to High-Performance Hardware on FPGAs. In *22nd International Conference on Field Programmable Logic and Applications*. IEEE, 531–534.

DONGARRA, J., GRAYBILL, R., HARROD, W., LUCAS, R., LUSK, E., LUSZCZEK, P., MCMAHON, J., SNAVELY, A., VETTER, J., YELICK, K., ALAM, S., CAMPBELL, R., CARRINGTON, L., CHEN, T. Y., KHALILI, O., MEREDITH, J., AND TIKIR, M. 2008. DARPA's HPCS Program: History, Models, Tools, Languages. *Advances in Computers 72*, 1–100.

JAASKELAINEN, P. O., DE LA LAMA, C. S., HUERTA, P., AND TAKALA, J. H. 2010. OpenCL-based Design Methodology for Application-Specific Processors. In *10th International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*. IEEE, 223–230.

MINGJIE, L., LEBEDEV, I., AND WAWRZYNEK, J. 2010. OpenRCL: Low-Power High-Performance Computing with Reconfigurable Devices. In *20th International Conference on Field Programmable Logic and Applications*. IEEE, 458–463.

OWAIDA, M., BELLAS, N., ANTONOPOULOS, C. D., DALOUKAS, K., AND ANTONIADIS, C. 2011. Massively Parallel Programming Models Used as Hardware Description Languages: The OpenCL Case. In *International Conference on Computer-Aided Design*. IEEE/ACM, 326–333.

PAPAKONSTANTINOU, A., GURURAJ, K., STRATTON, J. A., CHEN, D., CONG, J., AND HWU, W.-M. W. 2009. FCUDA: Enabling Efficient Compilation of CUDA Kernels onto FPGAs. In *7th Symposium on Application Specific Processors*. IEEE, 35–42.