Evaluation of high-level languages for general FPGA acceleration

Antonio Saavedra*, Gabriel L. Nazar†,
Nicolai Stawinoga*, Ben Juurlink*

* *Embedded System Architecture, Technische Universität Berlin, Berlin, Germany*
† *Informatics Institute, Federal University of Rio Grande do Sul, Porto Alegre, Brazil*

ABSTRACT

High-Level Synthesis (HLS) improves productivity compared to Register-Transfer Level (RTL) hardware descriptions. Despite its rise in popularity, there is still a performance gap compared to RTL design flows. One promising approach to bridge this gap has been the use of Domain-Specific Languages (DSLs), which allow increasing performance by restricting the application domain. In recent years, two promising DSL-based HLS tools have been proposed, Spatial and HeteroCL, that aim to translate these benefits into the design of general accelerators. In this work we compare these tools with custom RTL designs over diverse benchmarks, showing that a reduction of, on average, 10x of lines of code comes with a performance gap that can be as low as 8%. Our main conclusion is that the tools are suitable for faster design flows, and that previous hardware development experience and algorithm properties should be the main deciding factors between both similar tools.

KEYWORDS: FPGA, High-Level Synthesis, hardware acceleration

1 Introduction

Although reconfigurable architectures, such as Field-Programmable Gate Arrays (FPGAs), enable performance and energy efficiency improvements, programmability has been a major limitation for their widespread adoption. High-Level Synthesis (HLS) tools allow developers to work at higher abstraction levels and have increasingly been used in the past years to address this issue. However, they fail to deliver fully optimized acceleration, and even with great recent improvements, this quality of results (QoR) gap is still present [LSVH19].

Traditionally, HLS has been focused on software-oriented languages like C/C++, including commercial tools from FPGA manufacturers. Aiming to improve QoR, Domain-Specific Languages (DSL) allow programmers to generate efficient hardware through HLS. Due to the constrained domains, DSLs reach high-level abstractions with less QoR penalty than general-purpose software languages. While most DSLs focus on DSP for image/video, there have been efforts to develop DSLs for general-purpose hardware acceleration [KKO+18, LCH+19]. The main idea of this new DSL approach is using generalized parallelization and optimization patterns to describe accelerators at an abstract level, to later compile and synthesize them to optimized low-level hardware descriptions.

Although the benefits of DSL-based HLS approaches have been demonstrated in comparison to traditional HLS tools and pure software implementations, it is still important to assess the QoR penalty of these tools and languages compared to lower-level design flows. We focused on understanding where these new approaches stand in the HLS landscape and on how further improvements can be achieved in the future. For our analysis, we tested Spatial [KKO+18] and HeteroCL [LCH+19] and compared the obtained results with hand-written RTL.

2 Hardware Implementations

To test the development process with these two HLS tools, we used 3 benchmarks algorithms:

Orthogonal Matching Pursuit (OMP): a greedy algorithm that is commonly used in Compressive Sensing (CS) for signal reconstruction. OMP operates mostly with matrix operations, such as column-vector product and QR decomposition, that can be solved through vector inner products. Each iteration has data dependencies with the previous one, thus limiting the parallelism. Our accelerators were designed for a 1024×128 CS matrix with a signal sparsity of 16.

General Matrix Multiplication (GEMM): is the standard algorithm for matrix multiplication. It has applications in a wide range of different linear algebra problems in areas including image processing and machine learning. The inputs for our designs are two 1024×1024 matrices.

K-means clustering: is an iterative algorithm that partitions data without supervision. It has applications in machine learning, data mining, and image processing. The algorithm works by assigning each data point to one of k partitions. The iterative process consists of assigning each data point to the centroid of the nearest partition and then updating the centroids with the mean of all its assigned points. We used as inputs 1024 3-dimensional data points and performed 3 iterations with 256 clusters.

To evaluate the HLS DSLs, we implemented the benchmark in RTL (with Verilog) and both DSLs: Spatial and HeteroCL. All the implementations were designed for the same problem size for each algorithm and were validated with each other.

RTL: our RTL implementations were developed with the SystemVerilog HDL. The core of the OMP's architecture is the vector inner product implemented as parallel multiplications, followed by a binary adder tree. Due to the data dependencies, this hardware is reused in each iteration. To fully exploit data and instruction parallelism, GEMM's accelerator is a two-dimensional systolic array. K-means core is a linear systolic array, where each processing element computes the distance of the input to a specific cluster centroid.

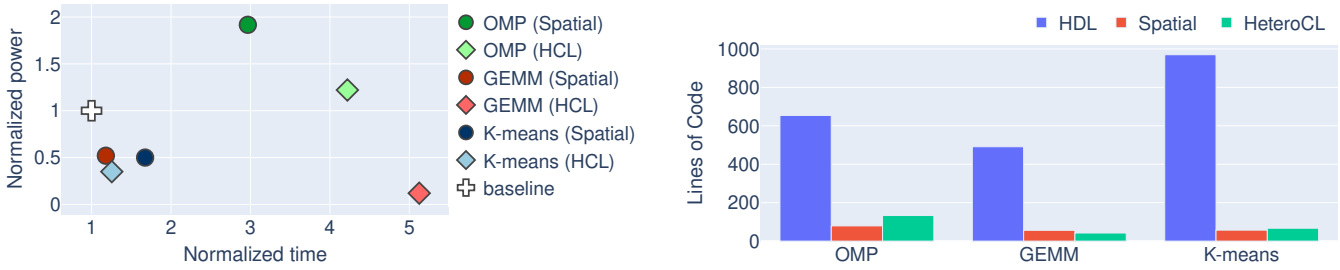
HLS DSLs: we used for GEMM and K-means the publicly available samples provided by each tool developer, only adjusting them to the required problem size. OMP implementation consists of two sequential, iterative loops that implement the greedy iterations, that internally exploit the parallelism of vector operations.

3 Experimental Results

Results were obtained targeting a Xilinx Ultrascale+ XCZU9EG FPGA. Spatial was configured to target the Ultrascale MPSoC. For HeteroCL, we targeted Vivado HLS C/C++ code generator.

Table 1: Resource utilization and performance results of different workflows

	OMP			GEMM			K-means		
Language	HDL	Spatial	HCL	HDL	Spatial	HCL	HDL	Spatial	HCL
LUTs	61031	171265	189435	8302	27827	105	3353	26271	17504
Regs	94221	215367	25245	18417	21811	156	6619	21787	14720
DSP	256	2360	10	507	54	3	3	21	18
BRAM	11	11	71	4	8	0	44	1	3
Power (mW)	673	1290	822	743	387	88	516	263	180
Time (ms)	230.22	683.22	971.56	8388	9061	42984	792	1326	994



(a) HLS performance relative to RTL

(b) Lines of code

Figure 1: Performance and power (a) and lines of code (b) for each implementation.

3.1 Performance

Table 1 presents the FPGA resource utilization, latency, and power for each benchmark implementation. The performance and power of HLS designs are also compared to the RTL baseline in Fig 1a.

In general, HDL implementations have higher performance, at the cost of a higher power. QoR results vary greatly for different benchmarks. The most significant QoR gaps occur when the DSLs are unable to take full advantage of specialized data types or structures. A notable gap is observable for both HLS OMP implementations. The main reason being that the HDL accelerator uses a custom floating-point representation with 17-bit mantissae, while the smallest available on the HLS tools were the standard 32-bit floating-point. Another great performance gap is observed for HeteroCL version of GEMM, due to the limitation of the publicly available backend to implement systolic arrays. As a result, a purely sequential implementation is obtained.

For applications that better suit the tools, performance gaps can be as low as 8 % (Spatial GEMM) and 25 % (HCL K-means). However, even when a QoR gap is observable, the HLS designs are still extremely efficient. For instance, the RTL OMP accelerator reaches speedups up to 67x when compared to software designs [SPZHF17]. HLS performances are still a highly efficient alternative to software approaches, as is also reported on [LCH+19].

3.2 Programmability and portability

The two DSLs present different approaches to customize the hardware designs. Spatial allows for more detailed customization, e.g., for defining the memory hierarchy. This could benefit programmers searching for more control, but it also reduces the abstraction level, as they are responsible for describing part of the architecture. On the other hand, in HeteroCL the customization of the

architecture is decoupled from the algorithm description. This gives the programmer the ability to describe the algorithm abstracted from the architecture, to later apply hardware customizations. This facilitates the exploration of optimizations to programmers unfamiliar with hardware design. Nevertheless, for designs with many computing stages, it can increase the complexity. Fig. 1b shows that both DSLs greatly reduce the lines of codes for each benchmark. On average, the reductions were 89.64% and 87.86% for Spatial and HeteroCL respectively.

The approaches are also different when it comes to portability. Spatial provides highly specialized workflows generating bitstream for selected FPGAs, while HeteroCL generates code to be implemented using device-specific tools, namely Vivado or Intel HLS. Porting Spatial code to a different platform is very simple and does not require code changes, if and only if the new target is also supported. In contrast, HeteroCL is not completely automatized and glue logic has to be provided by the developer, but supports a wider range of FPGAs.

4 Conclusions and future work

HLS presents a promising perspective for the development of FPGA-based platforms. The use of DSLs for HLS seems to be a step in the right direction and the tested tools appear to be suitable to increase design productivity. Experimental results show that, while requiring nearly 10× fewer lines of code, the time penalty can be as low as 8% and 25% compared to RTL design.

Three factors should be analyzed in order to choose between the tools. First, experience in hardware development, as it is important to fully use Spatial optimizations, while HeteroCL could be better suited for software developers. Secondly, the algorithm should be analyzed, as for larger sizes, the decoupled algorithms and optimizations could increase the complexity of HeteroCL code, while it is a powerful tool in smaller designs. Finally, available target platforms are also to be considered.

For future works, the design-space exploration functionalities available on the tools should be taken into account for a more in-depth analysis. Additionally, with a fully automatized workflow in both DSLs, the generated memory interfaces could also be analyzed.

References

[KKO+18] D. Koeplinger, C. Kozyrakis, K. Olukotun, M. Feldman, R. Prabhakar, Y. Zhang, S. Hadjis, R. Fiszel, T. Zhao, L. Nardi, and A. Pedram. Spatial: a language and compiler for application accelerators. In *39th ACM SIGPLAN Conf. on Programming Language Design and Implementation - PLDI*, 2018.

[LCH+19] Y. Lai, Y. Chi, Y. Hu, J. Wang, C. Yu, Y. Zhou, J. Cong, and Z. Zhang. HeteroCL: A Multi-Paradigm Programming Infrastructure for Software-Defined Reconfigurable Computing. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2019.

[LSVH19] S. Lahti, P. Sjövall, J. Vanne, and T.D. Hämäläinen. Are We There Yet? A Study on the State of High-Level Synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019.

[SPZHF17] A. Saavedra, J. Pezoa, P. Zarkesh-Ha, and M. Figueroa. An embedded system for face classification in infrared video using sparse representation. In *Applications of Digital Image Processing XL*. SPIE, 2017.