SODA: An End-To-End Open-Source Hardware Compiler for Machine Learning Accelerators

Nicolas Bohm Agostini[†‡], Serena Curzel[§‡], Ankur Limaye[‡], Vinay Amatya[‡], Marco Minutoli[‡],
Vito Giovanni Castellana[‡], Joseph Manzano[‡], Fabrizio Ferrandi[§], Antonino Tumeo[‡]
[‡]*Pacific Northwest National Laboratory*, Richland, WA, USA
[†]*Northeastern University*, Boston, MA, USA
[§]*Politecnico di Milano*, Milan, Italy

I. Introduction

Modern scientific experimental workflows are diverse (e.g., environmental monitoring, high energy physics, materials synthesis), but all require specialized processing at the edge, near the sensors, to deal with the enormous amount of acquired data and perform *low latency reasoning* to enable autonomous control. Nowadays, designing and implementing specialized systems needs large teams of expert hardware designers, and is economically viable only for solutions that have broad commercial applications. The conventional process for designing specialized systems consists in identifying common computational patterns to accelerate, and developing highly customized accelerators for these patterns at the register transfer-level (RTL), considering all possible trade-offs in terms of energy, performance, area, energy, and more, depending on the specific application domain. This becomes impractical for areas such as data science, machine learning (ML), and artificial intelligence (AI), where algorithmic approaches and programming frameworks keep evolving at a fast pace, rendering previous methods quickly obsolete. A new generation hardware design tools that allow to transition from the formulation of the algorithms to the implementation of a domain-specific system is required.

The SODA (Software Defined Architectures) Synthesizer [1] aims to bridge this productivity gap. The SODA Synthesizer is an open-source, modular, and extensible end-to-end hardware compiler for the generation of specialized systems from algorithms specified in high-level programming frameworks. It is composed of a compiler-based frontend, which interfaces to ML and AI frameworks and applies high-level optimizations, and a compiler-based backend, which generates RTL code and interfaces to physical design tools that produce the final implementation for field programmable gate arrays (FPGAs) or application-specific integrated circuits (ASICs). The frontend, SODA-OPT [2], available at: https://gitlab.pnnl.gov/sodalite/soda-opt, is implemented with the MLIR framework. The backend, PandA- Bambu [3], leverages state-of-the-art High-Level Synthesis (HLS) techniques and is available at: https://panda.dei.polimi.it.

II. SODA Synthesizer Overview

The SODA Synthesizer is composed of two main parts: the frontend and the hardware generation engine. The framework accepts input descriptions from high-level Python frameworks,

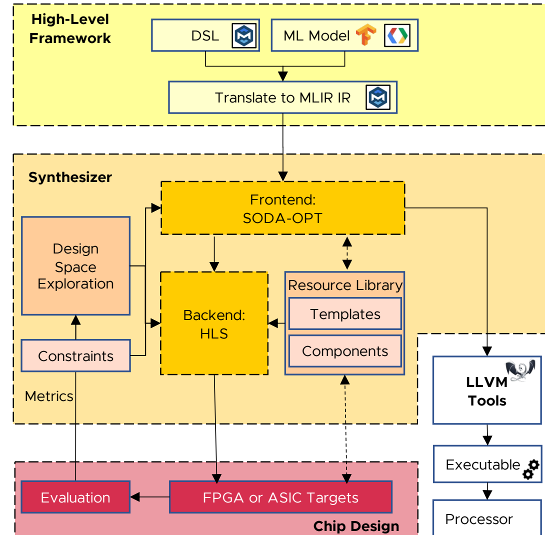

Fig. 1: The SODA Synthesizer.

translated by the frontend into a high-level intermediate representation (IR). The frontend exploits the Multi-Level Intermediate Representation (MLIR) [4] to perform hardware/software partitioning of the algorithm specifications, and architecture-independent optimizations. Subsequently, it generates a low-level IR (LLVM IR) for the HLS engine Bambu. A key difference between SODA and other frameworks that use HLS is that the interaction between frontend and backend happens through specialized compiler intermediate representations (IRs) and their progressive lowerings, allowing to perform optimizations at the right level of abstraction, and to pursue new research opportunities. Optimizations at all levels of the toolchain are implemented as compiler passes, which impact the generated hardware designs in terms of performance, area, and power. The exploration of the design space is made possible by enabling and disabling compiler passes or tuning their options through Python bindings.

A. SODA-OPT Frontend

SODA-OPT performs *search*, *outlining*, *optimization*, *dispatching*, and *acceleration* passes on the input program, preparing it for hardware synthesis. It leverages and extends MLIR. MLIR is a framework that allows building reusable,

Kernel	No Optimizations			Optimizations			Speedup
	Cycles	Area(um^2)	GF/W	Cycles	Area(um^2)	GF/W	
CONV_01	10,262,618	29,073	4.43	4,627,982	124,255	2.68	2.22
BIAS_02	251,694	10,395	11.48	40,826	60,048	9.01	6.17
RELU_03	151,342	7,385	41.55	38,446	35,695	38.39	3.94
CONV_04	85,380,948	36,814	3.32	83,380,180	37,556	3.34	1.02
BIAS_05	62,932	10,409	11.00	10,222	60,007	8.41	6.16
RELU_06	37,844	7,464	41.75	9,620	35,950	37.04	3.93

TABLE I: Evaluation of non optimized and optimized LeNet operators in ASIC technology (FreePDK 45 nm at 500 MHz)

extensible, and modular compiler infrastructure by defining *dialects*, i.e., self-contained IRs that respect MLIR's meta-IR syntax. Dialects allow modeling code at different levels of abstraction, enabling the use of specialized representations to facilitate compiler optimizations. These include abstractions for linear algebra, polyhedral analysis, structured control flow, and others. Several high-level programming frameworks for various domains such as machine learning (TensorFlow, ONNX-MLIR, TORCH-MLIR), scientific computing (NPCOMP), and general-purpose languages (e.g., the FLANG frontend for Fortran) started leveraging MLIR to implement their own specific dialects, optimization passes, and lowering methods to translate their programs into built-in MLIR dialects. Built-in dialects are entry points to SODA, enabling high-level programming frameworks to integrate with our toolchain.

SODA-OPT passes ingest MLIR inputs from high-level frameworks, identify key code regions, and outline them into separate MLIR modules, introducing a custom dialect to partition input applications into an orchestrating host program and custom hardware accelerators. Code regions that are selected for hardware acceleration undergo an optimization pipeline with progressive lowerings through different MLIR dialects, until they are translated into an LLVM IR restructured for hardware synthesis. Instead, the host module is lowered into an LLVM IR file that includes runtime calls to control the generated custom accelerators. Traditional HLS design flows expect manual code modifications that restructure the original algorithm (to create internal buffers or apply profitable tiling strategies) or tool-specific pragma annotations (to guide unrolling or provide alias information). Instead, SODA-OPT exploits dedicated and context-specific MLIR dialects to apply *systematic* high-level transformations.

B. SODA Synthesizer Backend

Bambu generates the accelerator designs starting from the low-level LLVM IR produced by SODA-OPT. Bambu has several frontends based on standard compilers (GCC or CLANG), it builds an internal IR to perform HLS steps (including bitwidth analysis, loop optimizations, resource allocation, scheduling, and binding algorithms), and generates the designs in a hardware description language (Verilog or VHDL). Alongside synthesizable HDL, it also automatically produces testbenches for verification. Bambu enables SODA to target FPGAs (from Xilinx, Altera, Lattice, NanoXplore) and ASICs. For ASICs, SODA supports Verilog-to-GDSII generation with both commercial and open-source logic synthesis tools.

Bambu is optimized to support a wide set of C and C++ constructs, but it can also ingest LLVM IR through its internal Clang frontend; through SODA-OPT, we connect Bambu with MLIR code. The LLVM IR generated after SODA-OPT high-level optimizations is restructured for HLS, resulting in more efficient accelerators with respect to inputs directly translated from MLIR to LLVM IR.

Bambu produces RTL designs following the finite state machine with datapath (FSMD) model; the accelerators can subsequently be integrated in larger system-level designs, with or without microcontrollers driving the execution. Bambu also exposes modular synthesis methodologies [5]: differently from other HLS tools, it can generate modules representing functions that may be reused or replicated across an entire design and composed in a complex multi-accelerator system.

We have extended Bambu with new HLS methodologies that can integrate FSMD modules as processing elements in coarse-grained dataflow designs [6], and in high-throughput, dynamically scheduled, multithreaded parallel templates [7]. MLIR descriptions are naturally parallel and hierarchical, so it is practical to instantiate such architectural templates from SODA-OPT: rather than requiring manual annotations on the input code, we can define the design hierarchy at a higher level of abstraction by exploiting MLIR.

III. End-to-end Example

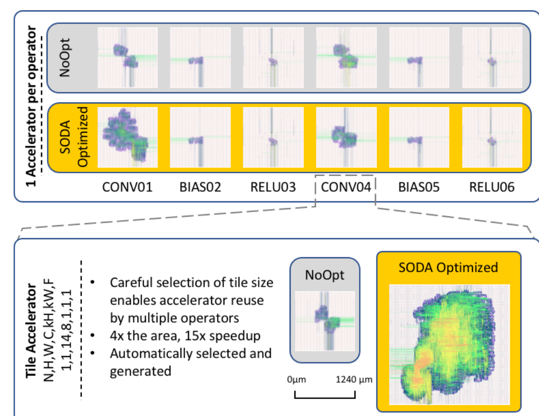

Fig. 2: ASIC implementations of LeNet layers.

To demonstrate SODA end-to-end synthesis capabilities, we automatically translate a LeNet model trained in TensorFlow to the `linalg` dialect and employ SODA-OPT to search, outline, and optimize different regions of the network, then

generating different specialized accelerators with Bambu. Table I reports the evaluation of the SODA implementations of different layers from the LeNet convolutional neural network model, synthesized with the OpenROAD flow targeting the FreePDK 45 nm cell library and a frequency of 500 MHz. For these layers, we also obtained floorplans in standard GDSII format, shown in Figure 2. All accelerators employ 32-bit floating point units. Optimizations provide a performance increase (speedup) proportional to the increase in area. Power efficiency (GF/W) may slightly reduce due to increase in power consumption of the faster solutions.

IV. Conclusion

The modern experimental workflow requires domain-specialized systems at the edge and on the instruments to enable low latency data analytics, reasoning, and autonomous control. However, the conventional hardware design flow requires significant efforts from large teams of expert hardware designers. We provided an overview of the SODA Synthesizer, an open-source compiler toolchain that enables automated generation of domain specialized systems from high-level programming frameworks to silicon, that aims to bridge the design productivity gap. The toolchain is composed of a frontend that interfaces with high-level programming frameworks and performs high-level optimizations, and a backend based on state-of-the-art HLS techniques to generate the RTL description of custom accelerators. SODA interfaces to open-source logic synthesis and physical layout tools to generate chip designs, providing an end-to-end solution for agile hardware design.

References

[1] N. Bohm Agostini, S. Curzel, J. Zhang, A. Limaye, C. Tan, V. Amatya, M. Minutoli, V. G. Castellana, J. Manzano, D. Brooks, G.-Y. Wei, and A. Tumeo, "Bridging Python to Silicon: The SODA Toolchain," *IEEE Micro*, pp. 1–1, 2022.

[2] N. Bohm Agostini, S. Curzel, C. Amatya, Vinay Tan, , M. Minutoli, V. G. Castellana, J. Manzano, D. Kaeli, and A. Tumeo, "An MLIR-based Compiler Flow for System-Level Design and Hardware Acceleration," in *ICCAD 2022*, 2022, p. to appear.

[3] F. Ferrandi, V. G. Castellana, S. Curzel, P. Fezzardi, M. Fiorito, M. Lattuada, M. Minutoli, C. Pilato, and A. Tumeo, "Bambu: an open-source research framework for the high-level synthesis of complex applications," in *DAC 2021*, 2021, pp. 1327–1330.

[4] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, and O. Zinenko, "Mlir: Scaling compiler infrastructure for domain specific computation," in *CGO*, 2021, p. 2–14.

[5] M. Minutoli, V. G. Castellana, A. Tumeo, and F. Ferrandi, "Interprocedural resource sharing in high level synthesis through function proxies," in *FPL 2015*, 2015, pp. 1–8.

[6] V. G. Castellana, A. Tumeo, and F. Ferrandi, "High-level synthesis of parallel specifications coupling static and dynamic controllers," in *IPDPS '21: IEEE International Parallel and Distributed Processing Symposium*, 2021, pp. 192–202.

[7] M. Minutoli, V. Castellana, N. Saporetti, S. Devecchi, M. Lattuada, P. Fezzardi, A. Tumeo, and F. Ferrandi, "Svelto: High-Level Synthesis of Multi-Threaded Accelerators for Graph Analytics," *IEEE Transactions on Computers*, no. 01, pp. 1–14, 2021.