SODA Synthesizer: an Open-Source, End-to-End Hardware Compiler

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

***Abstract*—Enabling autonomous control in novel scientific experimental workflows requires the ability to generate highly specialized systems for data analysis and artificial intelligence, enabling the low-latency reasoning capabilities needed to take real-time decisions. This paper presents the SODA (Software Defined Accelerators) framework, an open-source modular, multi-level, no-human-in-the-loop, hardware compiler that enables end-to-end generation of specialized accelerators from high-level data science frameworks. SODA is composed of SODA-Opt, a high-level frontend developed in MLIR that interfaces with domain-specific programming environments and allows performing system level design, and Bambu, a state-of-the-art high-level synthesis (HLS) engine that can target different device technologies. The framework implements design space exploration as compiler optimization passes. We show how the modular, yet tight, integration of the high-level optimizer and lower-level HLS tools enables the generation of accelerators optimized for the computational patterns of converged applications. We then discuss some of the research opportunities that such an open-source framework allows.**

I. Introduction

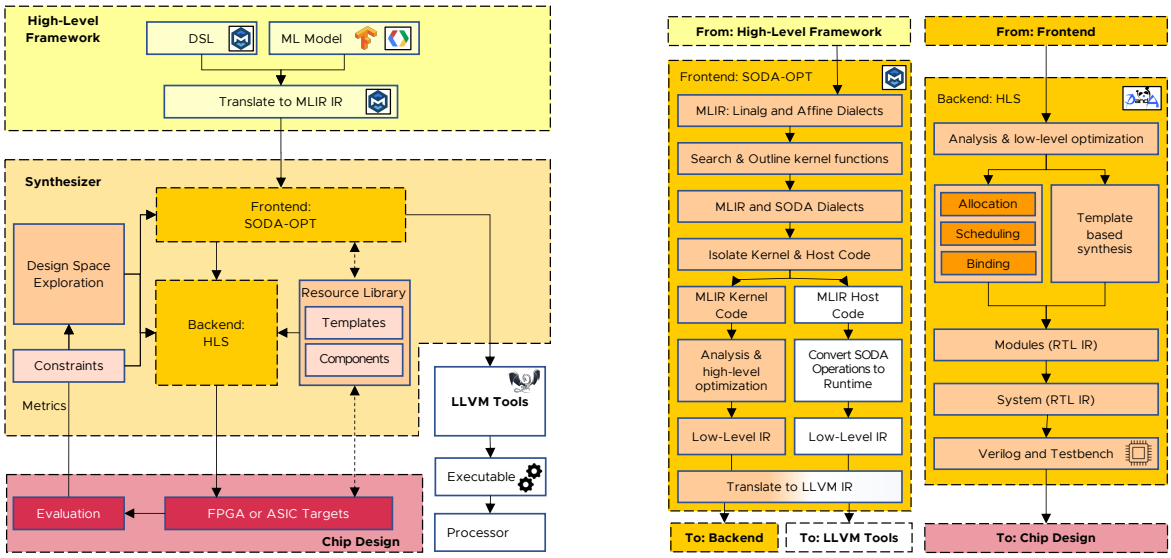
Emerging scientific applications, including network (power grid, communication, transportation, etc) analysis, environmental monitoring, high energy physics, material synthesis, and, in general, experimental scientific workflows, require efficient processing of a combination of data analysis, machine learning (ML), and scientific computing algorithms. Systems for all these areas need to analyze in-situ continuous streams of multi-modal data and take decisions in real-time to enable autonomous control of the experiments at very different scales. Achieving all the metrics that these diverse *edge* systems need to meet in terms of energy, performance, area, size and latency, is only possible through domain-specialized accelerators.

Domain scientists design and validate their algorithms in high-level programming frameworks, most of which are based on Python. Both algorithmic methods and programming frameworks are evolving quickly, especially in the data science area, making it extremely difficult to design specialized accelerators that are efficient with new approaches. In fact, the conventional hardware design cycle presents significant productivity limitations, often requiring an entire new design cycle each time new algorithms or models appear and preventing a wide exploration of alternative architectures.

The typical process requires hardware designers to distill key computational patterns from the algorithms that need to be accelerated, identify parallelism and data reuse opportunities, and design by hand custom functional units for specific kernels at the register-transfer level (RTL) with an HDL. A common alternative is to implement the functional units in C/C++ and convert them to HDL through commercial High-Level Synthesis (HLS) tools (Vitis HLS, Catapult C or Stratus HLS). In both cases, after functional verification, the HDL kernels are passed to downstream logic synthesis and physical design tools, and finally integrated into a system. This kind of design flow that combines manual coding with some automated processing is the standard practice for designing hardware. However, it still requires tremendous effort, and the quality highly depends on the designers' expertise. Moreover, the interactions between multiple Computer-Aided Design (CAD) tools at different levels of abstractions make the design process tedious and error-prone, introducing significant verification overheads and forcing manual propagation of changes across different stages of the design flow.

To address these issues, we introduced the SODA (Software Defined Accelerators) Synthesizer, an open-source, modular, and extensible end-to-end hardware compiler for the generation of highly specialized accelerators from algorithms designed in high-level programming frameworks. SODA is composed of a compiler-based frontend, to interface with high-level programming frameworks and apply high-level optimizations, and a compiler-based backend, to generate Verilog code and interface with external tools that compile the final design (either application-specific integrated circuits - ASICs - or field programmable gate arrays - FPGAs). The frontend, SODA-OPT[1] [1], is implemented with the MLIR compiler infrastructure, while the backend leverages a state-of-the-art HLS tool, Bambu [2], from the Panda framework[2]. Differently from other frameworks that use HLS, the interaction between frontend and backend happens through specialized compiler intermediate representations (IRs) and their progressive lowerings. Such a modular, yet tight, integration, allows performing optimizations at the right level of abstractions, and pursuing new research opportunities by adding new compiler representations and passes.

[1]SODA-OPT is available at: https://gitlab.pnnl.gov/sodalite/soda-opt
[2]Bambu is available at: https://panda.dei.polimi.it



(a) (b) (c)

Fig. 1: The SODA framework is an open-source, multi-level, modular, extensible, hardware generator composed of a high-level compiler and a lower-level HLS backend

II. The SODA Framework

Figure 1a provides an overview of the SODA framework, which can be divided in two parts: the frontend and the hardware generation engine. The framework accepts input descriptions from high-level Python frameworks, translated by the frontend into a high-level intermediate representation (IR). The frontend exploits the Multi-Level Intermediate Representation (MLIR) [3] 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, PandA-Bambu [2], a state-of-the-art open-source tool which, differently from most commercial alternatives, can also accept LLVM IR as input. Optimizations at all levels of the SODA toolchain are implemented as compiler passes, significantly influencing the generated hardware designs in terms of performance, area, and power. An exhaustive exploration of the design space is made possible by enabling and disabling compiler passes or tuning their options.

A. SODA-OPT Frontend

SODA-OPT (Figure 1b) is the high-level compiler frontend of the SODA framework. It performs *search*, *outlining*, *optimization*, *dispatching*, and *acceleration* passes on the input program, preparing it for hardware synthesis targeting FPGAs or ASICs. SODA-OPT leverages the MLIR framework.

MLIR is a framework that allows building reusable, 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. Several dialects of general use are maintained along with the MLIR framework. We refer to these as *built-in* dialects. They 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, optimizations 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 introduces a custom dialect to partition input applications into an orchestrating host program and custom hardware accelerators. SODA-OPT passes ingest MLIR inputs from high-level frameworks, identify key code regions, and outline them into separate MLIR modules. Code regions that are selected for hardware acceleration undergo an optimization pipeline with progressive lowerings through different MLIR dialects (`linalg` → `affine` → `scf` → `cf` → `llvm`), 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.

SODA-OPT performs the following high-level optimizations at the `affine` or lower dialects: tiling, unrolling, temporary buffer allocation, alloca buffer promotion, scalar replacement of aggregates (SRoA), early alias analysis, common sub-expression elimination (CSE), and dead code elimination (DCE). When properly combined together, these optimizations provide several benefits to the HLS backend, including: easier operation scheduling, increase of instruction-level and data-level parallelism, reduction of the number of accesses to external memory, favoring reuse of previously read values

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)

(storing them in registers), aggregation on local registers instead of external memory accesses, concurrent scheduling of independent memory operations on arrays, removal of redundant or unnecessary operations improving resource utilization.

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. These can expose instruction- and data-level parallelism, perform loop transformations, and apply various other steps such as buffer hoisting or accumulation on temporary variables. SODA-OPT leverages the `linalg` dialect to identify operations and separate hardware and software partitions, then it optimizes loops through the `affine` dialect, and finally performs CSE, DCE, and SRoA optimizations through the `cf`, `arith`, and `memref` dialects.

B. SODA Synthesizer Backend

The SODA framework backend, shown in Figure 1c, is Bambu, a state-of-the-art HLS tool that generate the accelerators 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 can also automatically produce 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 (Synopsis Design Compiler) and open-source (OpenROAD flow) logic synthesis and phisical layout 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 generates designs at the register transfer level (RTL) 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 [4]: 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 [5], and in high-throughput, dynamically scheduled, multithreaded parallel templates [6]. MLIR descriptions are naturally parallel and hierarchical, making possible 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. Synthesis Example

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. The optimized LLVM IR generated by SODA-OPT is then passed down to Bambu to generate the different specialized accelerators. 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. The HLS process is specifically optimized for the target technology beforehand, by performing resource characterization and extraction of metrics that are then used to drive the execution of the HLS algorithms. 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.

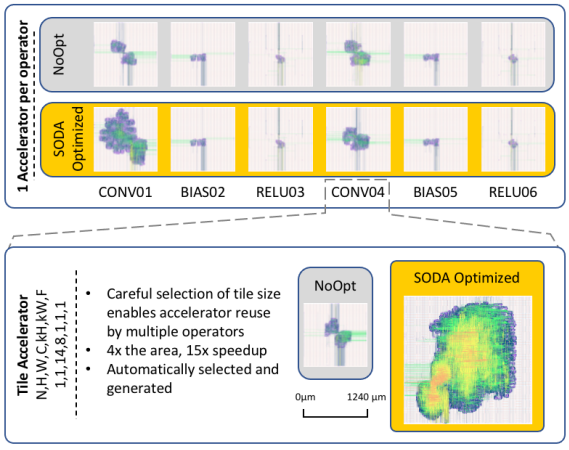


Fig. 2: ASIC implementations of LeNet layers.

Figure 2 shows the layouts (extracted from the standard GDSII format for ASIC manufacturing) at the end of the entire SODA flow for each different layer in Table I, without and with high-level optimizations. We can visually see how the optimized, and faster, designs, occupy a larger area of the die.

IV. Research Opportunities

An open-source, modular compiler infrastructure provides several research opportunities. SODA-OPT already enables system level design. In fact, it can perform code partitioning, high-level optimizations for custom hardware generation, and composition of an entire system architecture by generating glue code for control processors or by assembling accelerators in dynamically scheduled architectures. Such an approach could be further extended by integrating with rapid prototyping platforms in the open-source hardware ecosystem, such as the Embedded Scalable Platforms (ESP) [7]. Specifically, Bambu could provide an open-source synthesis backend for custom accelerators to ESP, while SODA-OPT could drive the system design, leveraging the rich set of services offered by ESP to invoke the accelerators. From a more general point view, SODA-OPT could easily support other types of specialized accelerators beside general purpose processors and HLS generated accelerators. A multi-level retargetable compiler framework provides opportunities to couple static with dynamic analysis, enabling to capture information on data-dependent patterns (typically involving memory accesses) through automated instrumentation and profiling that could then be feed back to the hardware generation engine to facilitate the exploration of the memory hierarchy and overall architecture design [8]. As presented in [9], the modularity of the framework even allows supporting novel computing paradigms, such as spiking neural networks. We have designed a new MLIR dialect to perform conversion and mapping of artificial neural networks on spiking neural networks. Digital versions of spiking neurons can then be synthesized through Bambu, enhancing what is currently done by hand with other FPGA platforms and HLS tools. An additional opportunity for SODA is to integrate with open-source tool that allow creating domain-specific FPGAs. SODA could, in fact, integrate with solutions like OpenFPGA [10], performing high-level analysis to identify patterns that might require additional hard macros in the hardware substrate while still leveraging fine-grained reconfigurability. The HLS backend could perform design space exploration, leveraging the hard macros through the resource library, or even synthesizing such macros. SODA would then be able to automatically provide the domain-specific FPGA organization and generate it using the logic synthesis and physical layout tools. Integration with domain-secific FPGAs allow exploring aspects such as custom memory interfaces, or new macros (containing for example memristors) that could simplify implementation of spiking neurons.

V. Conclusions

This paper overviews the SODA framework, an end-to-end, multi-level, open-source, hardware compiler composed of a frontend based on the MLIR infrastructure and a backend leveraging a state-of-the-art HLS engine. Through its frontend, SODA interfaces with a variety of high-level productive programming frameworks employed by domain scientists for novel "converged" applications. Through its backend, it can generate complete hardware designs targeting FPGAs from different vendors and ASICs. The end-to-end nature of the framework provides the agility needed to go from algorithmic formulation to hardware implementation. Its modularity and extensibility, coupled with its open-source nature, provide fundamental components to enable democratization of hardware design as well as unique research opportunities.

Acknowledgments

This research was partially supported by the Software Defined Accelerators for Data Analytics (SO(DA)^2) project in the Data Model Convergence (DMC) Initiative under the PNNL's Laboratory Directed Research and Development (LDRD) program, the Defense Advanced Research Projects Agency's (DARPA) Real-Time Machine Learning (RTML) program, and the Department of Defense / Department of Energy P38 project.

References

[1] N. Bohm Agostini, S. Curzel, V. Amatya, C. 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 '22: 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, p. To appear.

[2] 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: 58th Design Automation Conference*, 2021, pp. 1327–1330.

[3] 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: International Symposium on Code Generation and Optimization*, 2021, p. 2–14.

[4] M. Minutoli, V. G. Castellana, A. Tumeo, and F. Ferrandi, "Inter-procedural resource sharing in high level synthesis through function proxies," in *FPL 2015: 25th International Conference on Field Programmable Logic and Applications*, 2015, pp. 1–8.

[5] 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.

[6] 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.

[7] P. Mantovani, D. Giri, G. Di Guglielmo, L. Piccolboni, J. Zuckerman, E. G. Cota, M. Petracca, C. Pilato, and L. P. Carloni, "Agile soc development with open esp," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2020, pp. 1–9.

[8] A. Tumeo, "Architecture independent integrated early performance and energy estimation," in *IGSC '17: Eighth International Green and Sustainable Computing Conference*, 2017, pp. 1–6.

[9] S. Curzel, N. Bohm Agostini, S. Song, I. Dagli, A. Limaye, M. Minutoli, V. G. Castellana, V. Amatya, J. Manzano, A. Das, F. Ferrandi, and A. Tumeo, "Automated generation of integrated digital and spiking neuromorphic machine learning accelerators," in *ICCAD: International Conference On Computer Aided Design*, 2021, pp. 1–7.

[10] X. Tang, E. Giacomin, A. Alacchi, B. Chauviere, and P.-E. Gaillardon, "Openfpga: An opensource framework enabling rapid prototyping of customizable fpgas," in *29th International Conference on Field Programmable Logic and Applications*, ser. FPL'19, 2019, pp. 367–374.