Invited: The SODA Approach: Leveraging High-Level Synthesis for Hardware/Software Co-design and Hardware Specialization

Nicolas Bohm Agostini[*]
Serena Curzel[†]
Ankur Limaye, Vinay Amatya, Marco Minutoli,
Vito Giovanni Castellana, Joseph Manzano,
Antonino Tumeo
Pacific Northwest National Laboratory
Richland, WA, USA

Fabrizio Ferrandi
Politecnico di Milano
Milano, Italy

ABSTRACT

Novel "converged" applications combine phases of scientific simulation with data analysis and machine learning. Each computational phase can benefit from specialized accelerators. However, algorithms evolve so quickly that mapping them on existing accelerators is suboptimal or even impossible. This paper presents the SODA (Software Defined Accelerators) framework, a modular, multi-level, open-source, no-human-in-the-loop, hardware synthesizer that enables end-to-end generation of specialized accelerators. SODA is composed of SODA-Opt, a high-level frontend developed in MLIR that interfaces with domain-specific programming frameworks and allows performing system level design, and Bambu, a state-of-the-art high-level synthesis 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 a framework allows, including system-level design, profile driven optimization, and supporting new optimization metrics.

CCS CONCEPTS

• **Hardware → High-level and register-transfer level synthesis**.

KEYWORDS

High-level synthesis, hardware/software co-design

ACM Reference Format:
Nicolas Bohm Agostini, Serena Curzel, Ankur Limaye, Vinay Amatya, Marco Minutoli,, Vito Giovanni Castellana, Joseph Manzano, Antonino Tumeo, and Fabrizio Ferrandi. 2022. Invited: The SODA Approach: Leveraging High-Level Synthesis for Hardware/Software Co-design and Hardware Specialization. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC) (DAC '22), July 10–14, 2022, San Francisco, CA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3489517.3530628

[*]Also with Northeastern University.
[†]Also with Politecnico di Milano.

1 INTRODUCTION

Emerging scientific applications, including network (power grid, communication, transportation, etc) analysis, environmental monitoring, high energy physics, materials synthesis, and in general autonomous control of experimental workflows, require efficient processing of a combination of data analysis, machine learning (ML), and scientific computing algorithms. They need systems that can effectively support each phase of the computation and adapt in real-time to changes in the environment or the system under analysis, considering different, and often contrasting, energy, performance, area, and latency constraints. Systems able to meet all these requirements need 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 methods. In fact, the conventional hardware design cycle presents significant productivity limitations. Manually designing custom accelerators in hardware description languages (HDLs) is complex and time consuming, often requiring an entire new design cycle each time new algorithms or models appear and preventing a wide exploration of alternative architectures. General and automated solutions are needed to quickly transition from the formulation of an algorithm to the implementation of a dedicated accelerator.

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 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, with part manual coding and part automated processing, is 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

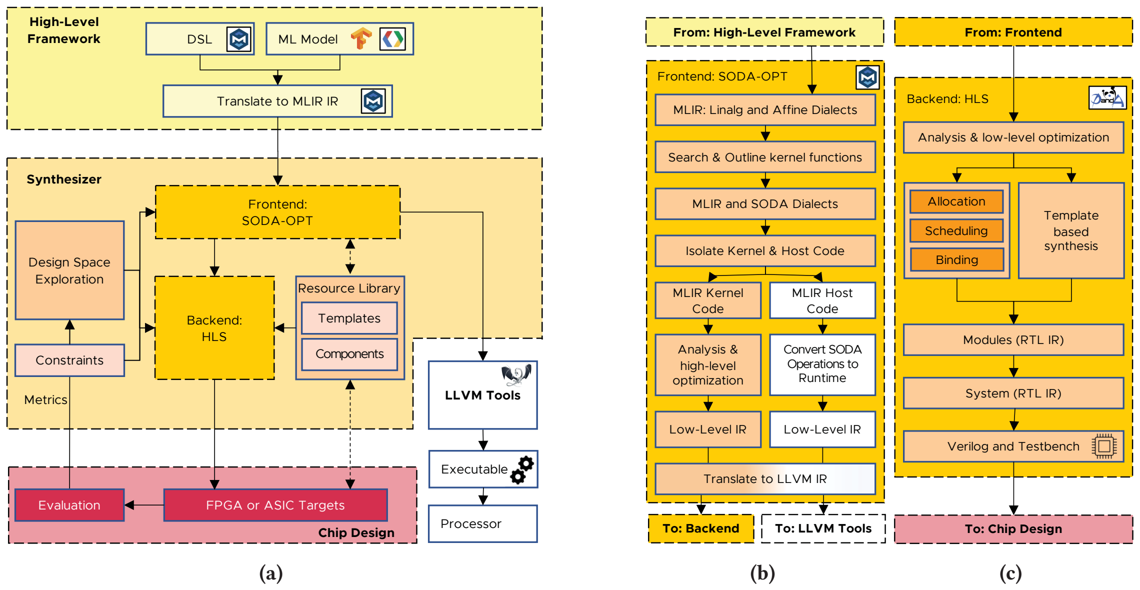

Figure 1: The SODA framework is an open-source, multi-level, modular, extensible, hardware generator composed of a high-level compiler and a lower-level HLS backend

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 introduce 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], is implemented with the MLIR compiler infrastructure, while the backend leverages a state-of-the-art HLS tool, Bambu [3], 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.

2 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) [5] 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 [3], 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.

2.1 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 and extends 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. We refer to dialects that are maintained in tree, along with the MLIR framework, as *built-in* dialects. 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, optimizations passes, and lowering methods to translate their programs into built-in MLIR dialects. Built-in dialects are entry points to the SODA Synthesizer, enabling high-level programming frameworks to integrate with our toolchain.

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

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 scheduled and combined together, these optimizations provide the following benefits for the HLS backend: 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 (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.

2.2 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 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 [8]: 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 [1], and in high-throughput, dynamically scheduled, multithreaded parallel templates [7]. 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

3 EVALUATION

We demonstrate the SODA approach by outlining and optimizing kernels and generating their hardware implementation.

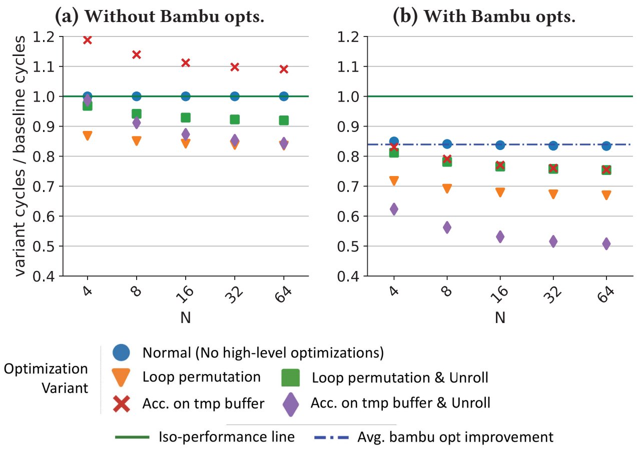

Figure 2: Relative difference between execution times of Matmul kernels with different MLIR and Bambu optimizations.

Benefits of high-level optimizations - High-level optimizations performed at the MLIR level have a direct impact on the final performance of the generated RTL code, by enabling further optimization opportunities within the HLS backend. For example, task level parallelism information exposes instruction and data level parallelism, which HLS can exploit for better operation scheduling. As an example, we synthesize matrix multiplication kernels kernels with different input sizes, while allowing SODA-OPT to apply several high-level optimizations: *permuting loop iterations*, *unrolling innermost loops*, and *allocating a temporary buffer on which to perform accumulations*. Figure 2a compares the execution latency of the optimized variants against the non-optimized variant (Normal). The solid line in bold represents the iso-performance line. Points under this line represent faster execution than the non-optimized baseline.

Figure 2b shows the average contribution of Bambu optimizations (18% clock cycles reduction) to the high-level optimizations. Points under the dashed line represent designs with lower execution delays than without any high-level optimization. In these designs, the selected high-level optimizations expose code structures that facilitate Bambu HLS process.

Neural Network case study - To demonstrate SODA end-to-end 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.

Figure 3 shows the SODA implementations of several different layers from the LeNet convolutional neural network model, in the standard GDSII format for ASIC manufacturing. Exploring available compiler options enables us to reach higher performance at the cost of increased area utilization.

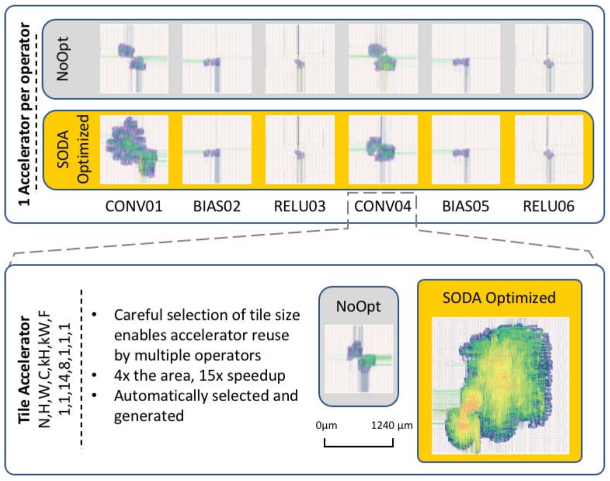
1 Accelerator per operator
NoOpt
SODA Optimized
CONV01
BIAS02
RELU03
CONV04
BIAS05
RELU06
Tile Accelerator
N,H,W,C,kH,kW,F
1,1,14,8,1,1,1
• Careful selection of tile size enables accelerator reuse by multiple operators
• 4x the area, 15x speedup
• Automatically selected and generated
NoOpt
SODA Optimized
0µm
1240 µm

Figure 3: ASIC implementations of LeNet layers.

4 RESEARCH OPPORTUNITIES

A modular compiler infrastructure provides several new research opportunities. SODA-OPT can already reason about the system level design, performing code partitioning, optimizations specific for the generation of custom hardware generation, and the composition of a system architecture by generating glue code for control processors or 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) [6]. 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 [9]. As presented in [2], the modularity of the framework even allows supporting novel computing paradigms, such as spiking neural networks. We have designed a new MLIR dialect able to deal with spiking neural networks, and the framework easily allows mapping spiking neurons on digital equivalents that can be synthesized through Bambu (enhancing what is currently done by hand with other FPGA platforms and conventional HLS tools). This approach could be further extended to support other aspects for the generation of custom circuits. For example, in the context of security, there already are initiatives to integrate homomorphic encryption into the MLIR framework [4]. By designing appropriate lowering passes, this information can be conveyed to the LLVM IR generated by SODA-OPT for both synthesizer backend and general purpose processors.

5 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. The modularity and extensibility of the framework provide unique new opportunities for reasoning at the system level, support profile driven hardware generation, and generation architectures with new properties (e.g., security).

REFERENCES

[1] V. G. Castellana, A. Tumeo, and F. Ferrandi. 2021. High-Level Synthesis of Parallel Specifications Coupling Static and Dynamic Controllers. In *IPDPS '21: IEEE International Parallel and Distributed Processing Symposium.* 192–202.
[2] 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. 2021. Automated Generation of Integrated Digital and Spiking Neuromorphic Machine Learning Accelerators. In *ICCAD: International Conference On Computer Aided Design.* 1–7.
[3] F. Ferrandi, V. G. Castellana, S. Curzel, P. Fezzardi, M. Fiorito, M. Lattuada, M. Minutoli, C. Pilato, and A. Tumeo. 2021. Bambu: an Open-Source Research Framework for the High-Level Synthesis of Complex Applications. In *DAC: 58th Design Automation Conference.* 1327–1330.
[4] S. Govindarajan and W. S. Moses. 2020. SyFER-MLIR: Integrating Fully Homomorphic Encryption Into the MLIR Compiler Framework. https://math.mit.edu/research/highschool/primes/materials/2020/Govindarajan-Moses.pdf
[5] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, and O. Zinenko. 2021. MLIR: Scaling Compiler Infrastructure for Domain Specific Computation. In *CGO: International Symposium on Code Generation and Optimization.* 2–14.
[6] P. Mantovani, D. Giri, G. Di Guglielmo, L. Piccolboni, J. Zuckerman, E. G. Cota, M. Petracca, C. Pilato, and L. P. Carloni. 2020. Agile SoC development with open ESP. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD).* IEEE, 1–9.
[7] M. Minutoli, V. Castellana, N. Saporetti, S. Devecchi, M. Lattuada, P. Fezzardi, A. Tumeo, and F. Ferrandi. 2021. Svelto: High-Level Synthesis of Multi-Threaded Accelerators for Graph Analytics. *IEEE Trans. Comput.* 01 (2021), 1–14.
[8] M. Minutoli, V. G. Castellana, A. Tumeo, and F. Ferrandi. 2015. Inter-procedural resource sharing in High Level Synthesis through function proxies. In *FPL 2015: 25th International Conference on Field Programmable Logic and Applications.* 1–8.
[9] A. Tumeo. 2017. Architecture independent integrated early performance and energy estimation. In *IGSC '17: Eighth International Green and Sustainable Computing Conference.* 1–6.