SODA Synthesizer: Status & Updates

June 30, 2024

Nicolas Bohm Agostini, Ankur Limaye, Reece Neff,

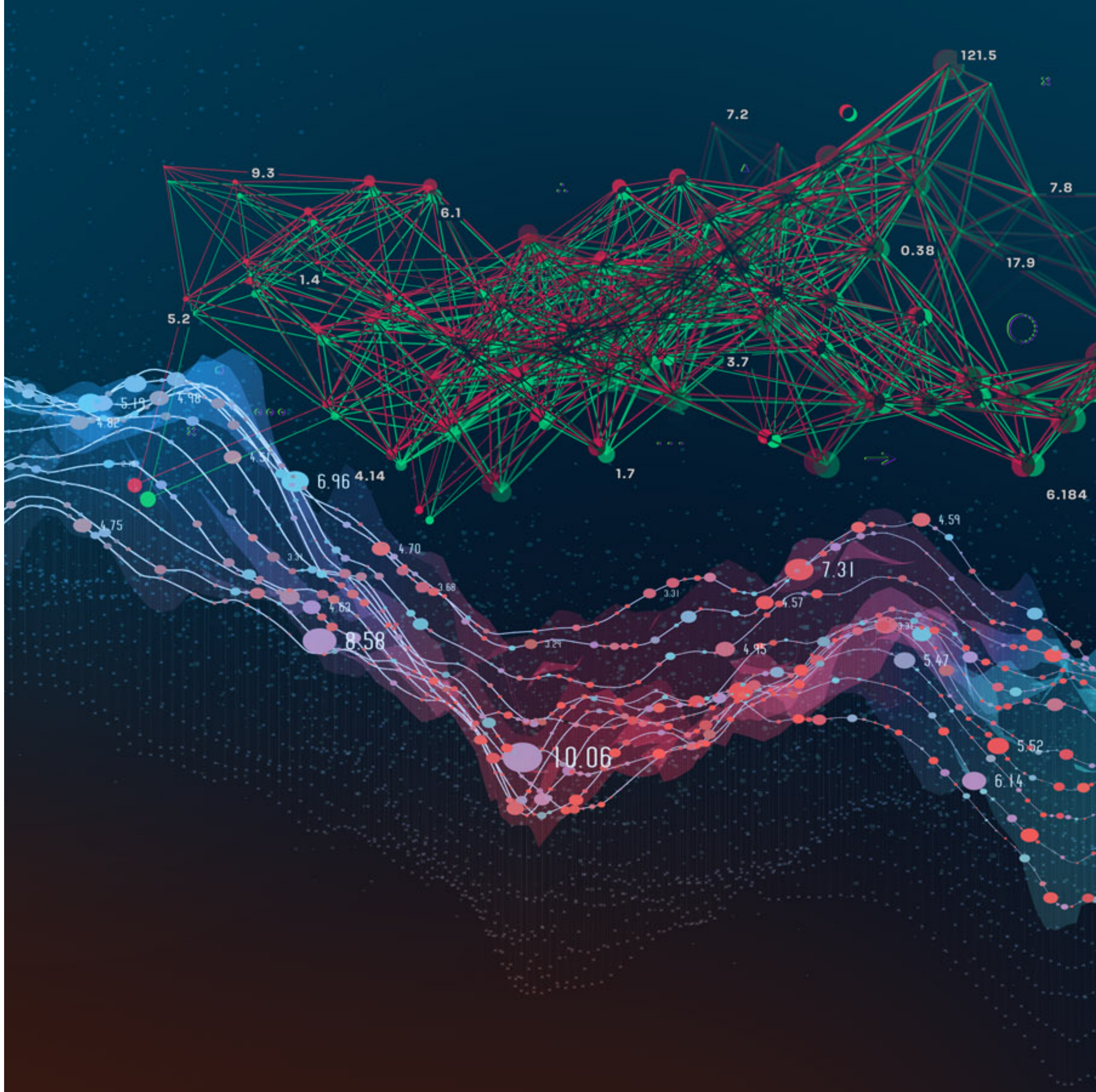
Varshika Mirtinti, Claudio Barone,

Vito Giovanni Castellana, Marco Minutoli,

Joseph Manzano, **Antonino Tumeo**,

Giovanni Gozzi, Michele Fiorito,

Serena Curzel and **Fabrizio Ferrandi**

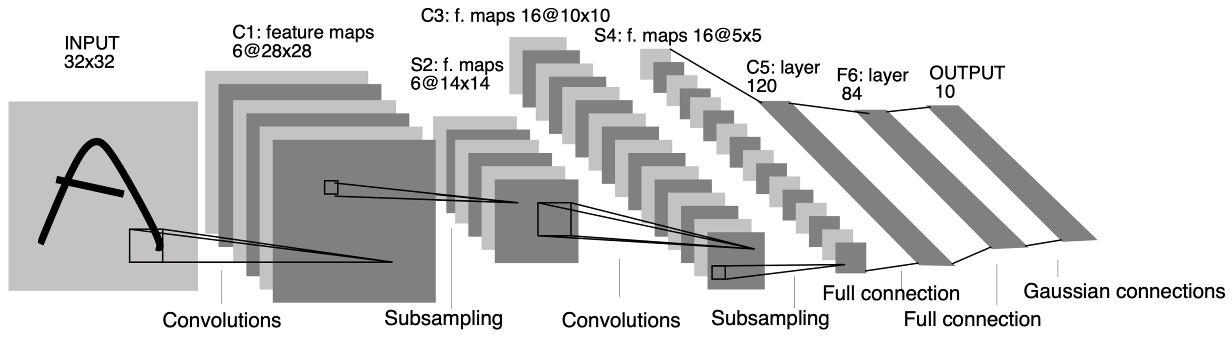


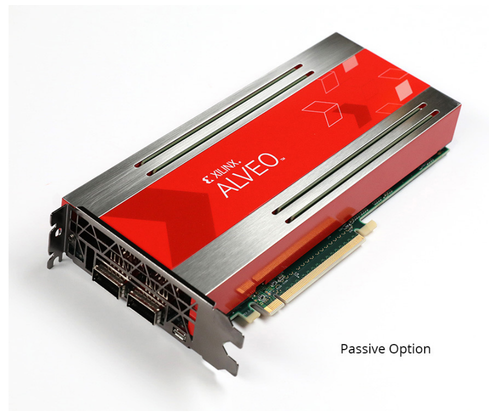
Motivations

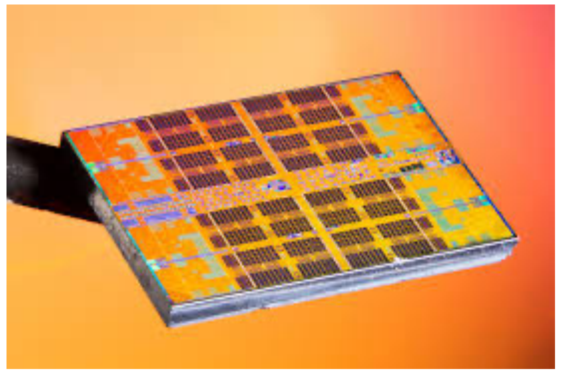
- Data science algorithms, approaches, and frameworks are quickly evolving
- Domain-specific accelerators are the only possible approach to keep increasing performance in tight constraints
- Existing accelerators start from specific models (i.e., mostly deep neural networks) or only try to accelerate specific computational patterns coming from high-level frameworks
- Designing hardware by hand is complex and time-consuming
- Depending on the application, a designer may want to explore performance, area, energy, accuracy, and more…

- ***Need tools to quickly transition from formulation of an algorithm to the accelerator implementation and explore the accelerator design along different dimensions***

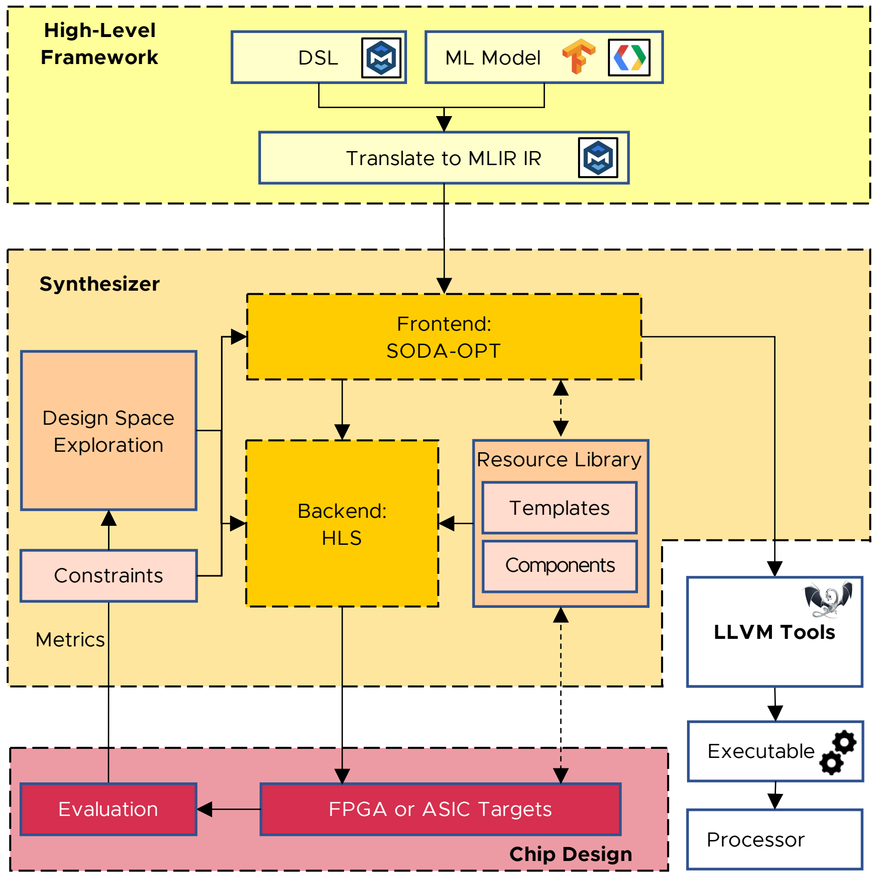
LeNet architecture from the original paper







SODA Synthesizer: Overview



[M. Minutoli, V. G. Castellana, C. Tan, J. Manzano, V. Amatya, A. Tumeo, D. Brooks, G-Y. Wei: SODA: a New Synthesis Infrastructure for Agile Hardware Design of Machine Learning Accelerators. ICCAD 2020: 98:1-98:7]

[J. Zhang, N. Bohm Agostini, S. Song, C. Tan, A. Limaye, V. Amatya, J. Manzano, M. Minutoli, V. G. Castellana, A. Tumeo, G-Y. Wei, D. Brooks: Towards Automatic and Agile AI/ML Accelerator Design with End-to-End Synthesis. ASAP 2021: 218-225]

- A modular, multi-level, interoperable, extensible, **open-source hardware compiler** from **high-level programming frameworks to silicon**
- Compiler-based frontend, leveraging the MultiLevel Intermediate Representation (MLIR)
- **Compiler-based backend**, leveraging state-of-the-art High-Level Synthesis (HLS) techniques
- Generates **synthesizable Verilog** for a variety of targets, from Field Programmable Gate Arrays (FPGAs) to Application Specific Integrated Circuits (ASICs)
- Optimizations at all levels are performed as **compiler optimization** passes

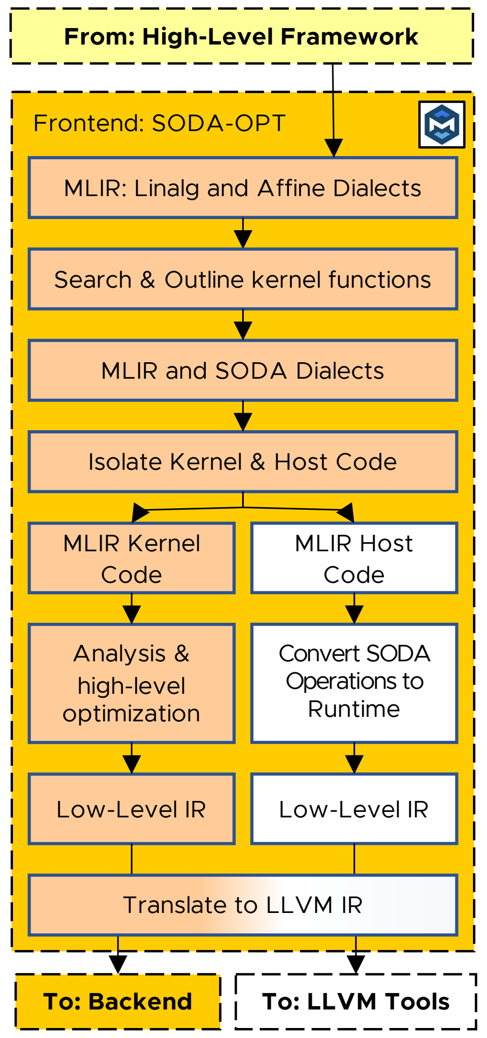
[N. Bohm Agostini, S. Curzel, J. Zhang, A. Limaye, C. Tan, V. Amatya, M. Minutoli, V.G. Castellana, J. Manzano, D. Brooks, G-Y. Wei, A. Tumeo: Bridging Python to Silicon: The SODA Toolchain. IEEE Micro Magazine 2022 - **Best Paper for 2022]**
[N. Bohm Agostini, S. Curzel, V. Amatya, C. Tan, M. Minutoli, V. G. Castellana, J. Manzano, D. Kaeli, A. Tumeo : An MLIR-based Compiler Flow for System-Level Design and Hardware Acceleration. ICCAD 22]
[Fabrizio Ferrandi, Vito Giovanni Castellana, Serena Curzel, Pietro Fezzardi, Michele Fiorito, Marco Lattuada, Marco Minutoli, Christian Pilato, Antonino Tumeo: Invited: Bambu: an Open-Source Research Framework for the High-Level Synthesis of Complex Applications. DAC 2021: 1327-1330]

SODA-OPT: Frontend and High-Level IR

- **SODA-OPT: S**earch, **O**utline, **D**ispatch, **A**ccelerate frontend **opt**imizer "generates" the SODA High-Level IR
- Employs and embraces the MLIR framework
  - MLIR: Multi-Level Intermediate Representation
  - Used in TensorFlow, TFRT, ONNX-MLIR, NPComp, others
  - Several architecture independent dialects (Linalg, Affine, SCF) and optimizations
- Interfaces with high-level ML frameworks through MLIR "bridges" (e.g., libraries, rewriters)
- Defines the SODA MLIR dialect and related compiler passes to:
  - Identify dataflow segments for hardware generation
  - Perform high-level optimizations (dataflow transformations, data-level and instruction-level parallelism extraction)
  - Generate interfacing code and runtime calls for microcontroller

[N. Bohm Agostini, S. Curzel, V. Amatya, C. Tan, M. Minutoli, V. G. Castellana, J. Manzano, D. Kaeli, A. Tumeo : An MLIR-based Compiler Flow for System-Level Design and Hardware Acceleration. ICCAD 22]

[N. Bohm Agostini, S. Curzel, D. Kaeli, A. Tumeo: SODA-OPT an MLIR based flow for co-design and high-level synthesis. CF 2022: 201-202 - **Best Poster Award**.]

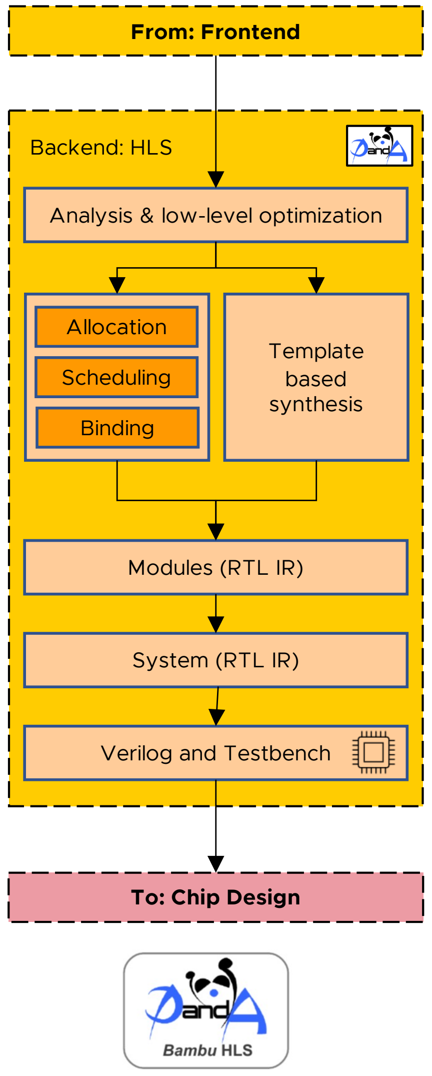

From: High-Level Framework
Frontend: SODA-OPT
MLIR: Linalg and Affine Dialects
Search & Outline kernel functions
MLIR and SODA Dialects
Isolate Kernel & Host Code
MLIR Kernel Code
MLIR Host Code
Analysis & high-level optimization
Convert SODA Operations to Runtime
Low-Level IR
Low-Level IR
Translate to LLVM IR
To: Backend
To: LLVM Tools

SODA-OPT: System Overview

https://github.com/pnnl/soda-opt

SODA Synthesizer: HLS Backend

- The synthesizer backend take as input the properly optimized low-level IR and generate the hardware descriptions of the accelerators
- The HLS backend is PandA-Bambu, an open-source state-state-of-the-art high-level synthesis (HLS)
  - Key features: **parallel accelerator designs**, **modular HLS**, and **ASIC support**
- The HLS backend provides automated testing and verification of the generated designs

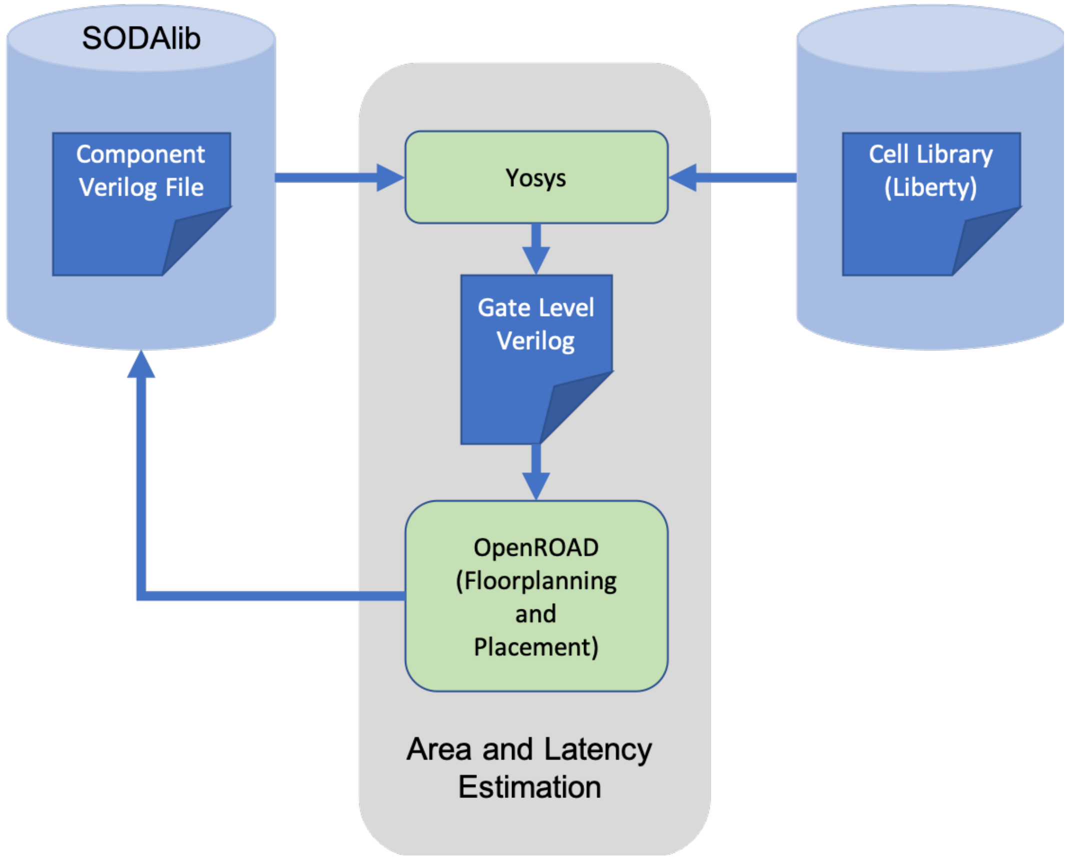


[Fabrizio Ferrandi, Vito Giovanni Castellana, Serena Curzel, Pietro Fezzardi, Michele Fiorito, Marco Lattuada, Marco Minutoli, Christian Pilato, Antonino Tumeo: Invited: Bambu: an Open-Source Research Framework for the High-Level Synthesis of Complex Applications. DAC 2021: 1327-1330]

https://panda.dei.polimi.it

SODA Synthesizer: ASIC targets

- The multi-level approach of the SODA toolchain allows supporting different target technologies (FPGA, ASIC) for actual generation of the designs
- SODA also supports ASIC targets:
  - **Commercial Tools** (Synopsys Design Compiler with Global Foundries 12/14 nm cells)
  - **OpenROAD suite** (OpenPDK 45nm and ASAP 7nm cell libraries)
- Backends' resources characterized for the target technology:
  - **HLS Backend: Eucalyptus** tool in Bambu, allows driving hardware synthesis algorithms to optimize for area, latency, etc.
- PandA-Bambu now also the opensource C frontend for **ZeroASIC' SIliconCompiler** (https://www.siliconcompiler.com)



SODA characterization flow. The characterization flow can be extended to synthesize HLS generated designs, or used to estimate their area-latency-power profiles to drive the Design Space Exploration engine

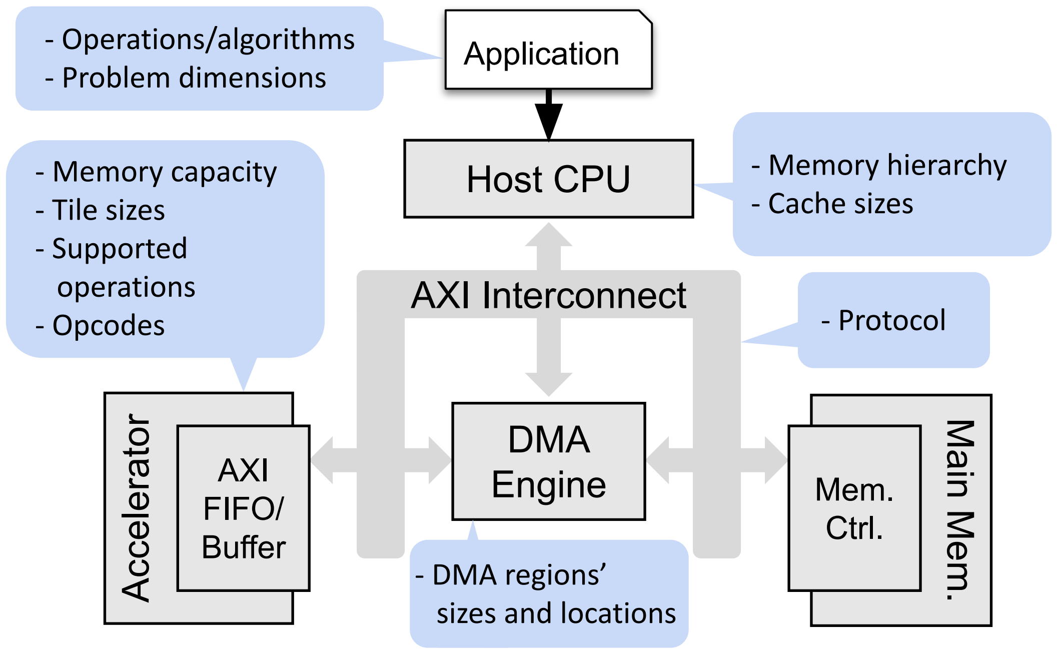
OpenROAD

https://theopenroadproject.org

SODA-Synthesizer Progress: AXI4MLIR - Host-Accelerator Communication

- Work in collaboration with University of Glasgow and Northeastern University
- AXI4MLIR implements **host code generation** to drive accelerators connected through an AXI-Stream Interface
  - Provides Efficient Host-Accelerator execution flow

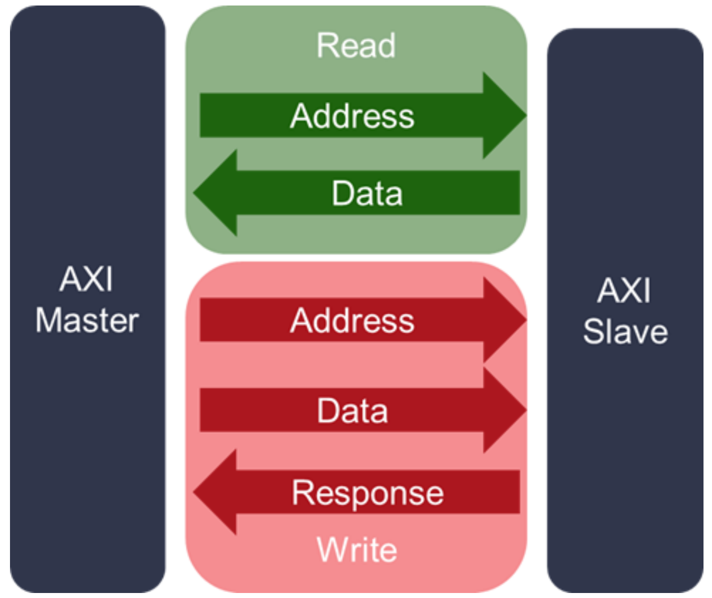
Before	Accelerators in isolation
Now	Host-Accelerator communication for **predefined** accelerators
Goal	Host-Accelerator communication for **generated** accelerators



[N. Bohm Agostini, J. Haris, P. Gibson, M. Jayaweera, N. Rubin, A. Tumeo, K. Abellan, J. Cano, D. Kaeli. AXI4MLIR: User-Driven Automatic Host Code Generation for Custom AXI-Based Accelerators. CGO 2024]

AXI4MLIR: User-Driven Automatic Host Code Generation for Custom AXI-Based Accelerators

- **MLIR extensions to describe** custom accelerators with arbitrary instructions
- Simple Host-Accelerator **communication abstraction** and **AXI library implementation**
- Currently implemented targeting Systems-on-Chip developed in with Xilinx Vitis, but generalizable to other open-source prototyping platforms



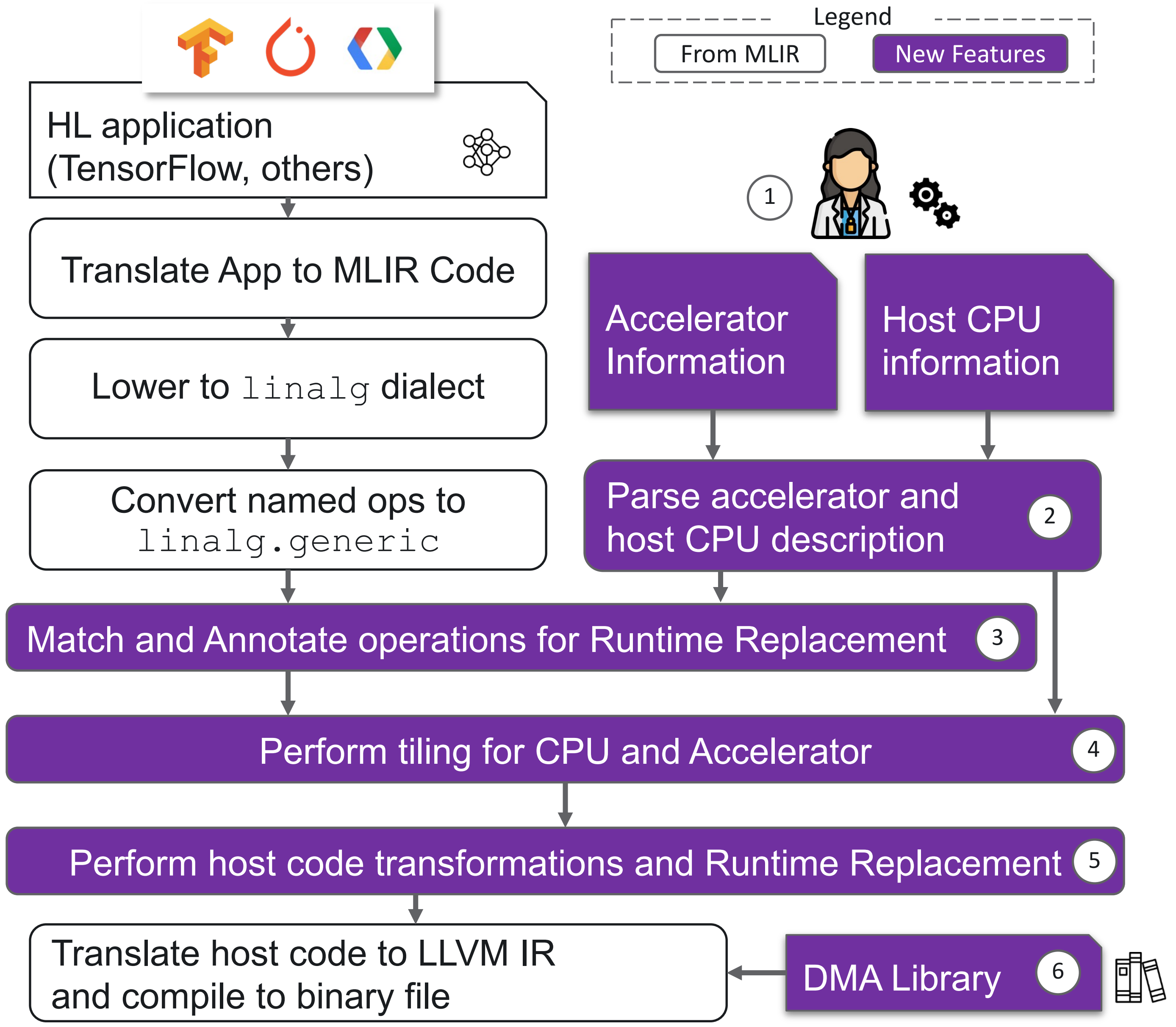
From: Werbrouck, F. “AXI Basics 1 - Introduction to AXI”. 2023. Available at: https://support.xilinx.com/s/article/1053914

AXI4MLIR: User-Driven Automatic Host Code Generation for Custom AXI-Based Accelerators

AXI4MLIR Approach

- **MLIR extensions to describe** custom accelerators with arbitrary instructions
- Simple Host-Accelerator **communication abstraction** and **AXI library implementation**
- Implements **host code generation** to drive accelerators connected through an AXI-Stream Interface

https://github.com/AXI4MLIR/axi4mlir



Legend
From MLIR
New Features
HL application (TensorFlow, others)
Translate App to MLIR Code
Lower to linalg dialect
Convert named ops to linalg.generic
1
Accelerator Information
Host CPU information
Parse accelerator and host CPU description
2
Match and Annotate operations for Runtime Replacement
3
Perform tiling for CPU and Accelerator
4
Perform host code transformations and Runtime Replacement
5
Translate host code to LLVM IR and compile to binary file
DMA Library
6

AXI4MLIR: Contributions

- An MLIR dialect to abstract **send** and **recv** transactions of data packages to the accelerator
  - The accel dialect:
    - `send, recv, send_literal, send_dim, send_idx`
- A communication library
  - Lightweight DMA Engine Library

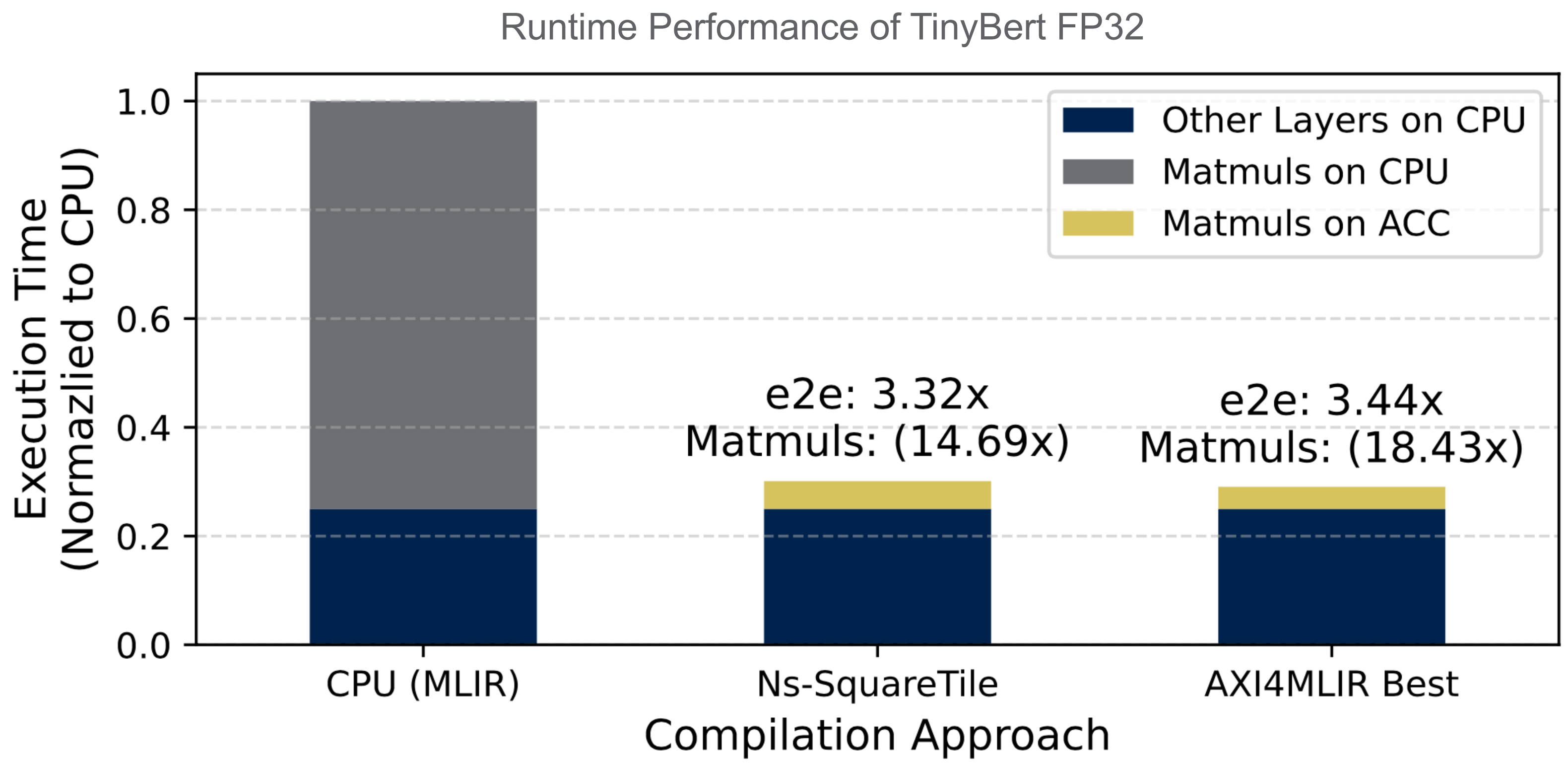
```
<final offset> = accel.send(<memref or subview>, <dma offset>)
```

```
%sA=memref.subview %A[%i, %j][%tile_size_I, %tile_size_J][%c1, %c1]
call @copy_to_dma_region (%sA, offset=4bytes)
call @dma_start_send(size=%tile_size_I*%tile_size_J*4bytes, offset=4bytes)
call @dma_wait_send_completion()
```

- A way to describe custom instructions and how to use them in an MLIR operation

AXI4MLIR Results

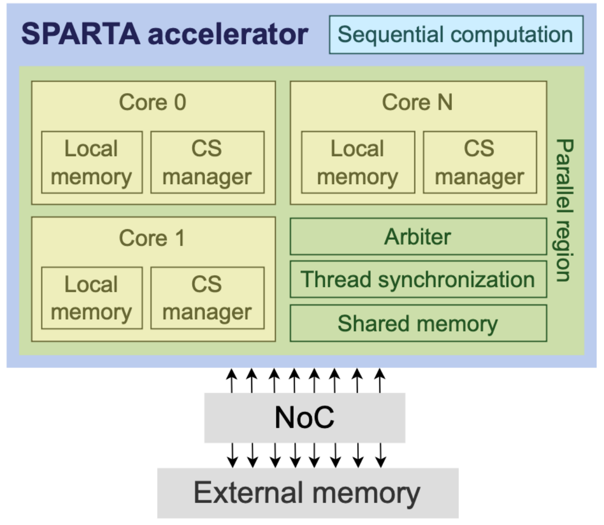
- Accelerating Matrix-Multiplication Layers of TinyBERT language model
- Presenting results for
  - CPU execution
  - Worst case (Nothing stationary)
  - Best case (oracle selection of best dataflow)
- Improves overall performance due to use of the accelerator
- Improves improves performance due to better orchestration of data
- Improves productivity

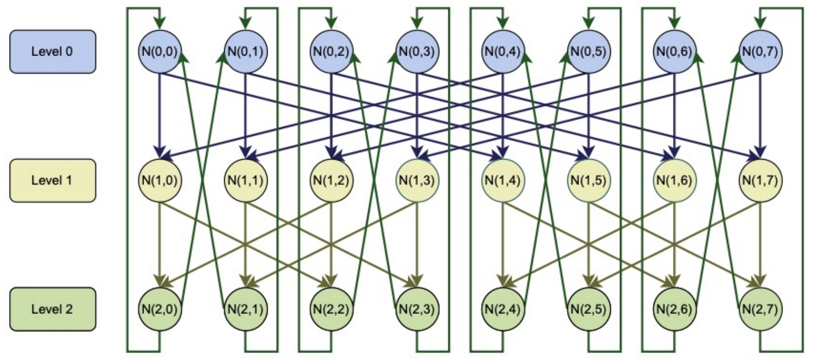


SPARTA: High-Level Synthesis of Parallel Multi-Threaded Accelerators

- A design methodology to perform High-Level Synthesis of Parallel Multi-Threaded Accelerators (SPARTA)
  - Improves our previous approach (SVELTO) for generating multithreaded parallel accelerators starting from OpenMP-annotated shared memory codes in flexibility and performance
  - Uses a library of synthesizable components to directly map OpenMP runtime calls using LLVM
  - Implements a deflection-based fine-grained network-on-chip to connect accelerators to external memory
  - Supports memory-side caches

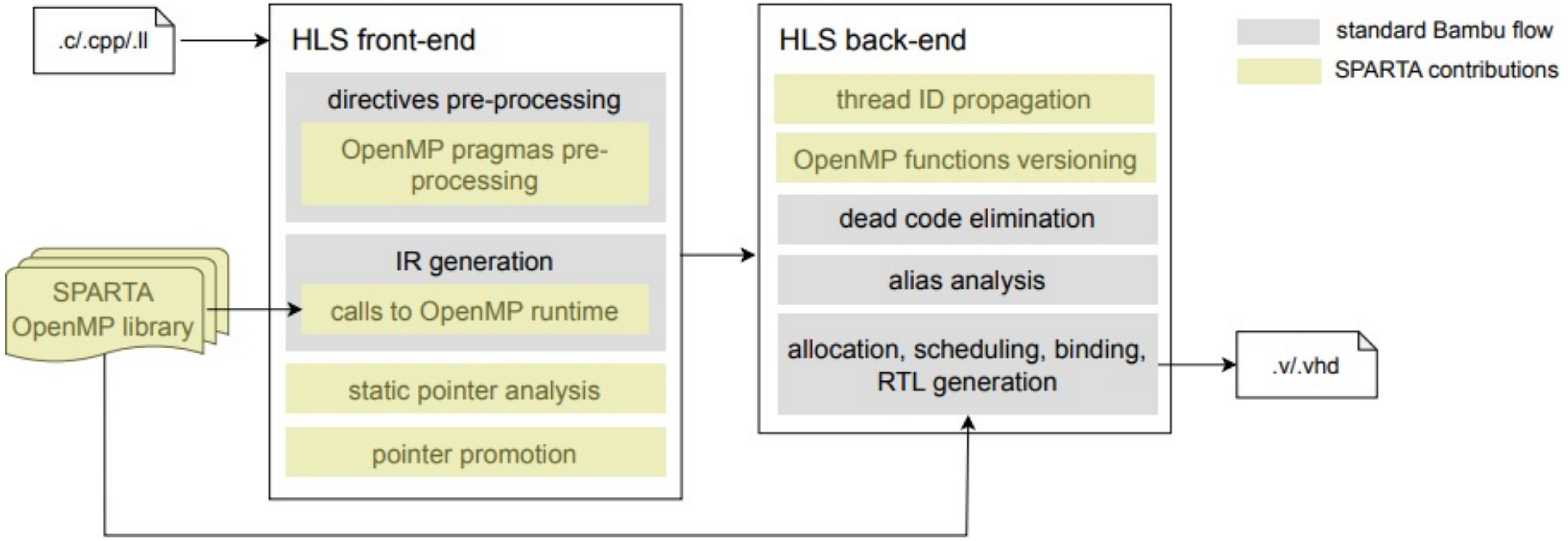
Before	Template based mapping from OpenMP codes
Now	New methodology that directly maps OpenMP runtime calls through LLVM
Goal	Ability to map designs from high-level languages through MLIR OpenMP dialect





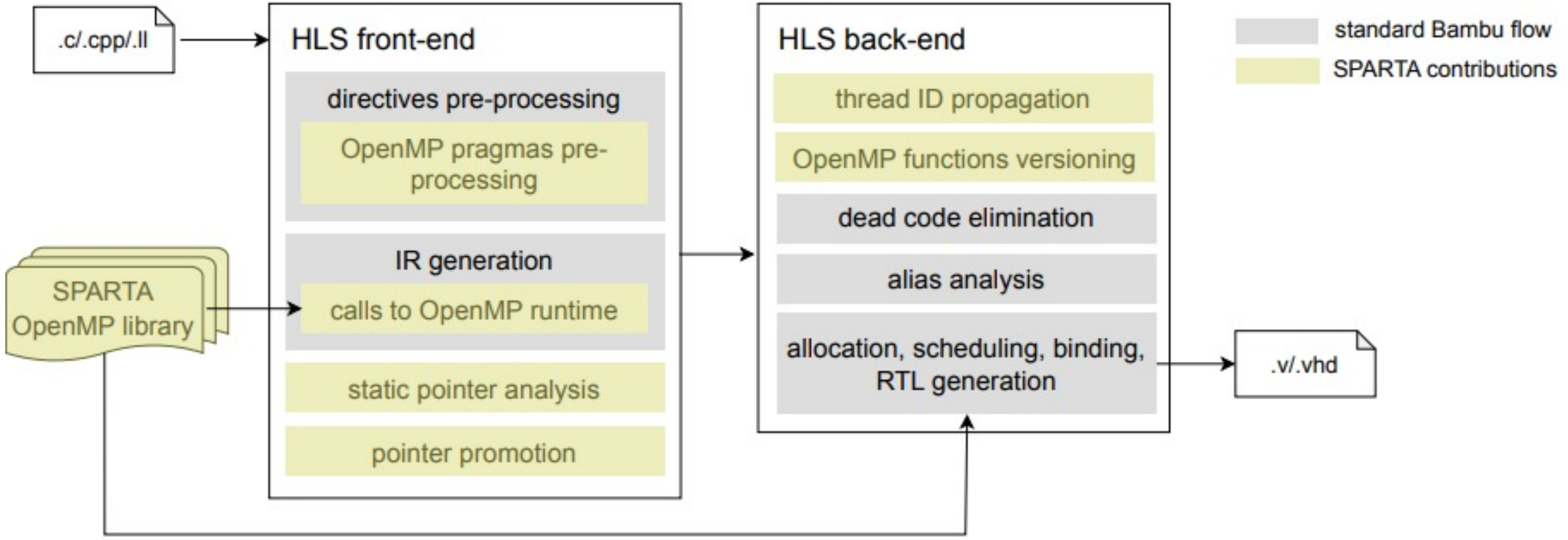
Bambu modified execution flow

- SPARTA modifies the execution flows of Bambu
- It implements the operation of the OpenMP runtime with ad-hoc hardware components.



Bambu modified execution flow

- Performs argument promotion to the arguments passed to the OpenMP runtime.
- It replicates the hardware creating multiple parallel cores.

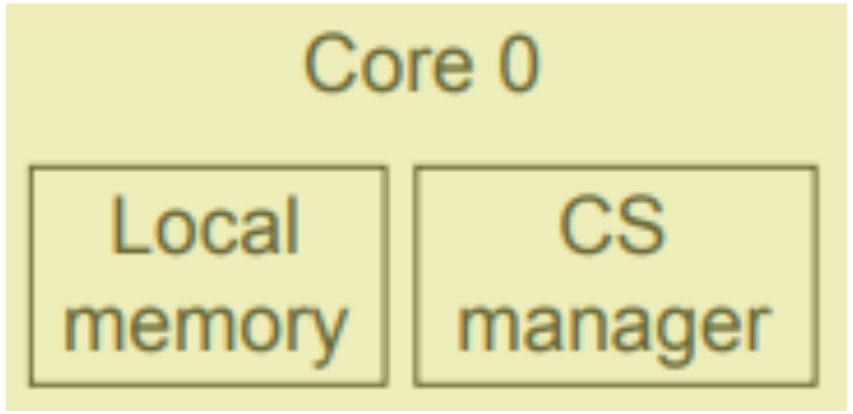


SPARTA architecture

Cores are the base elements of the SPARTA architecture:

- Each core implements a software function
- It has its own local memory (easy to access)

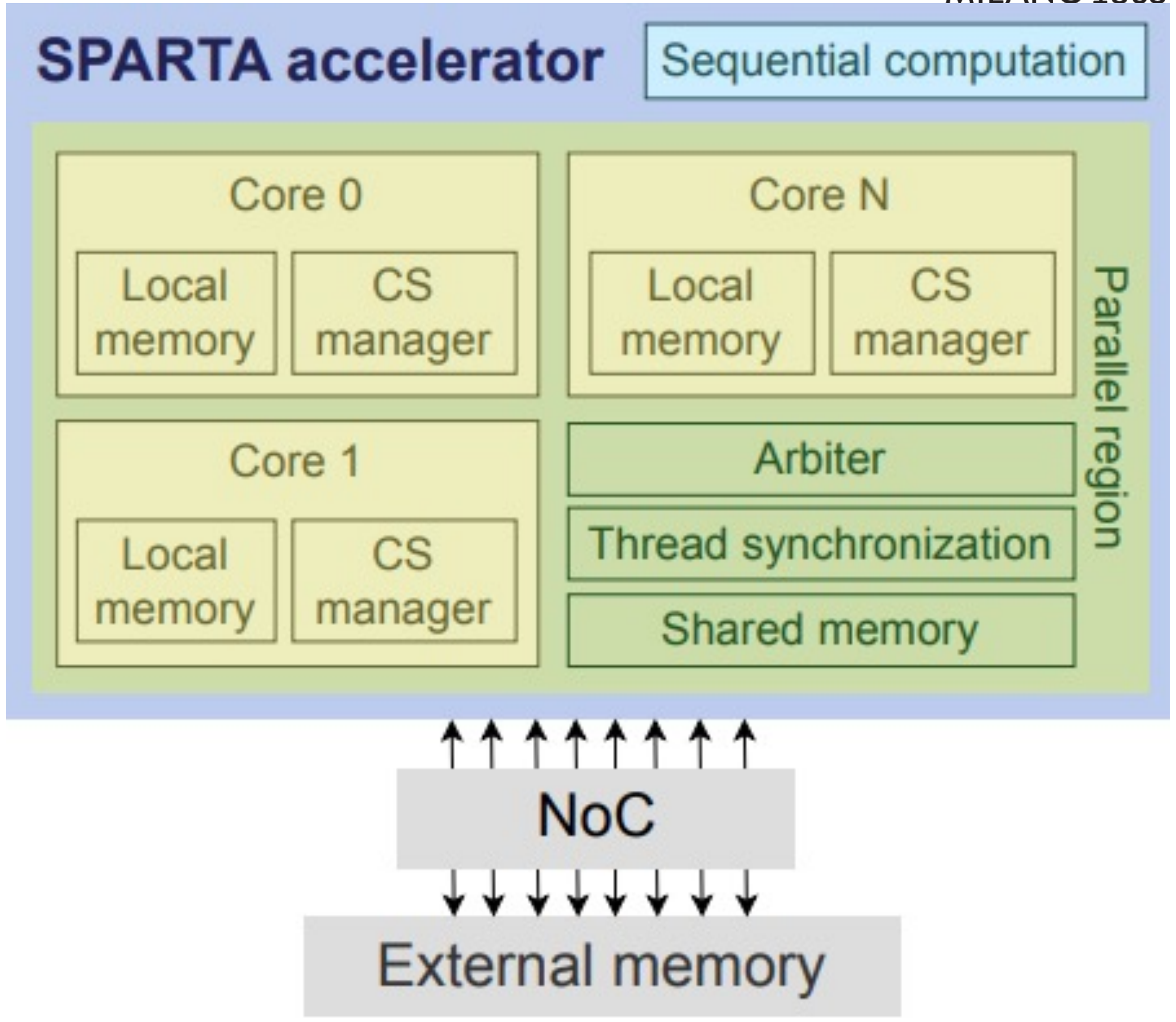
Multiple threads can execute on the same core: when an external operation requires multiple cycles the context active in the core changes increasing the utilization of the hardware resources.



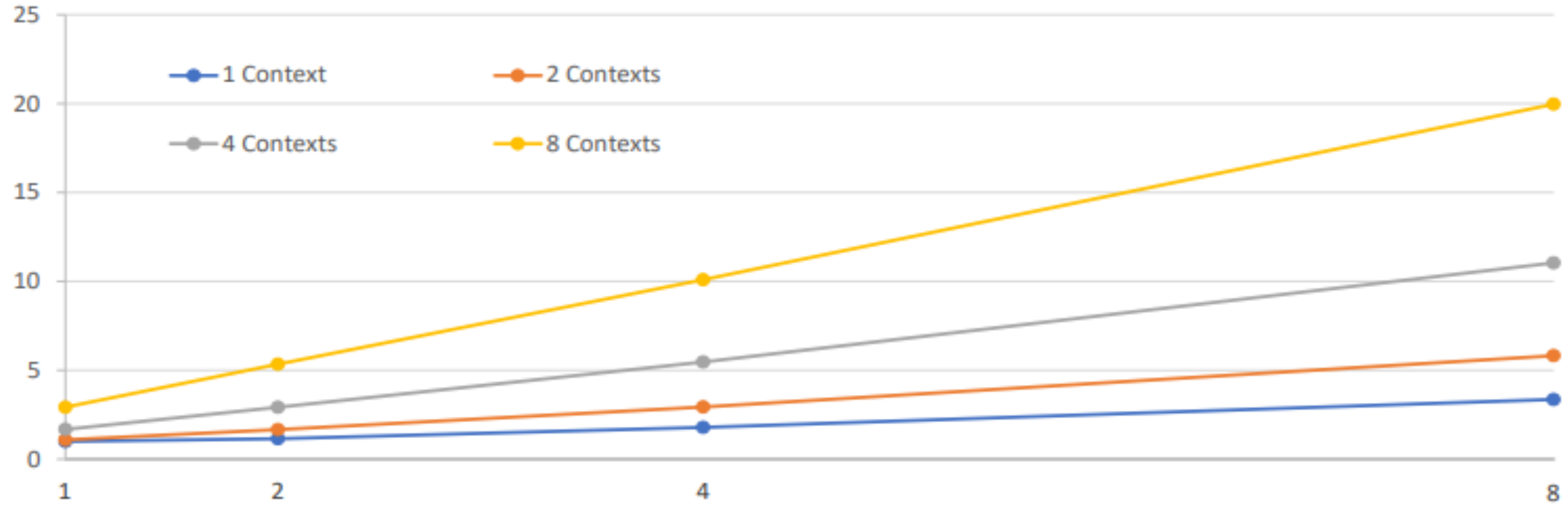
SPARTA architecture

Each parallel region can contain multiple cores:

- SPARTA adds hardware for arbitration and synchronization (critical sections, barriers…)
- It allocates the variable shared between the different cores like the reduction variables.



SPARTA Results



Speed up of SPARTA accelerators over the sequential baseline for the Triangle Count benchmark with different cores and contexts

Public Software Repositories

- SODA-Opt: https://github.com/pnnl/sodaopt
- Panda-Bambu HLS: https://panda.dei.polimi.it (latest release 2023.10)
- OpenROAD: https://theopenroadproject.org (external tool, leveraged by SODA toolchain to achieve end-to-end synthesis to ASIC in a fully opensource compiler toolchain)
- SODA docker image: https://hub.docker.com/r/agostini01/soda

SODA-OPT

PandA-Bambu HLS (2023.10)

SODA Docker Image

SODA Tutorial: *ISCA 2024 (tomorrow afternoon)*

Conclusions

- SODA implements an **end-to-end** (high-level frameworks to silicon) **compiler-based toolchain** for the generation of domain-specific accelerators
  - Modular, multi-level, extensible
  - All based on interoperating open-source technologies
  - Targets reconfigurable architectures FPGAs as well ASICs
  - Considers system-level implications
  - Enables automated design space exploration and agile hardware design
- **The SODA Synthesizer provides a no-human-in-the-loop toolchain from algorithmic formulation to hardware implementation for complex workloads**
- **SODA is also a research tool to explore novel generation methodologies for domain-specific systems**
  - Discussed AXI4MLIR and SPARTA

Thank you!

- This work has been partially supported by:
  - The AT SCALE Initiative at PNNL
  - The ASCR Project Compiler Frameworks and Hardware Generators in Support of Innovative US Government Designs
  - The Spoke 1 "FutureHPC & BigData” of the Italian Research Center on High-Performance Computing, Big Data and Quantum Computing (ICSC) funded by MUR Missione 4 - Next Generation EU (NGEU)

- Questions?
  - antonino.tumeo@pnnl.gov
  - fabrizio.ferrandi@polimi.it