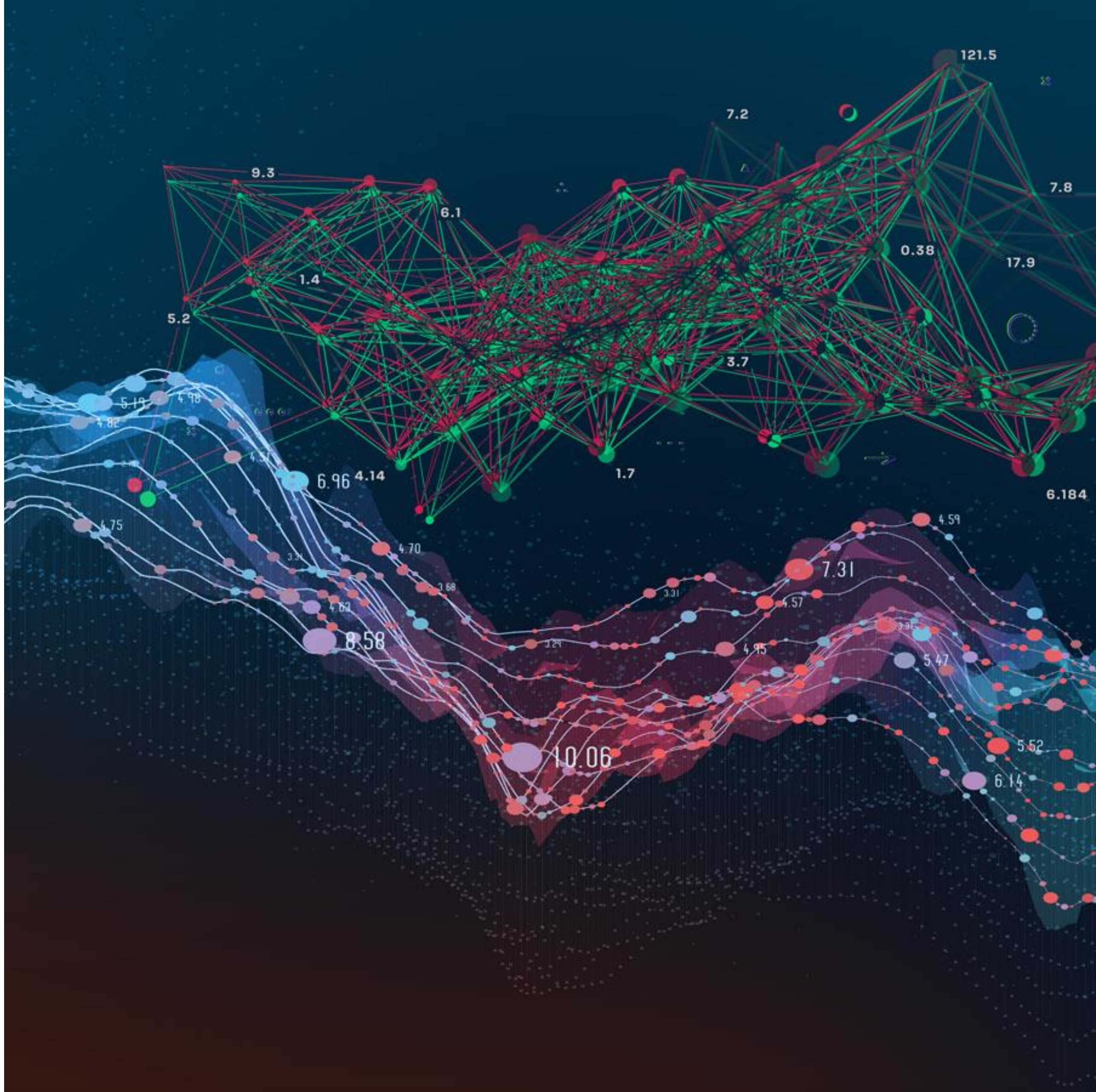
SODA Synthesizer

An End-to-End Compiler from High-Level Frameworks to Silicon

June 19, 2022

Nicolas Bohm Agostini, Serena Curzel, Reece Neff, **Ankur Limaye**, Vinay Amatya, Marco Minutoli, Vito Giovanni Castellana, Joseph Manzano, Antonino Tumeo
Pacific Northwest National Laboratory

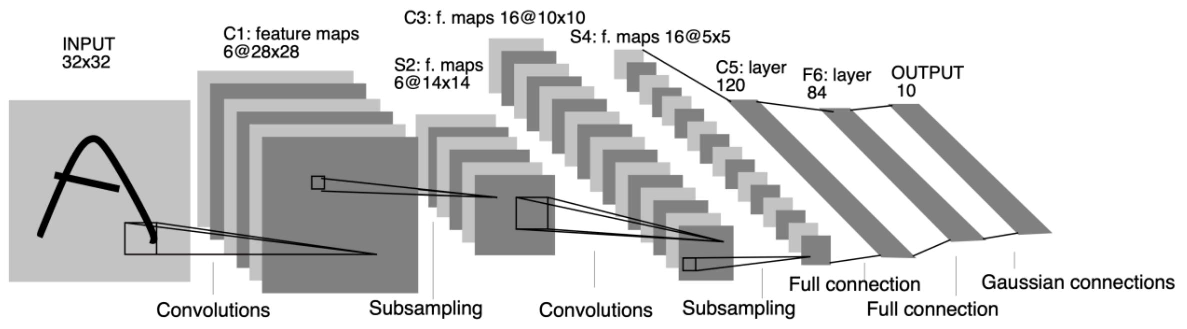
Michele Fiorito, Fabrizio Ferrandi
Politecnico di Milano

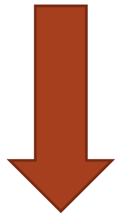


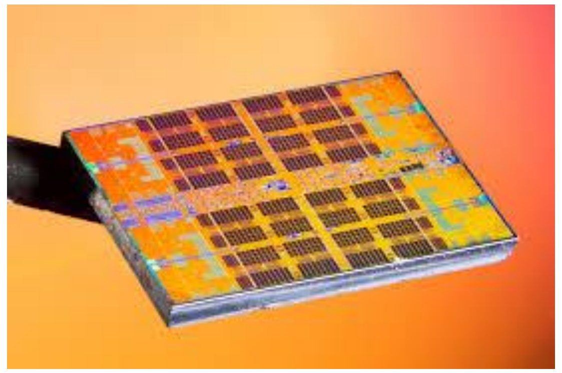
Motivations

- Data Science algorithms, Machine Learning models & frameworks are quickly evolving
  - To keep increasing the performance within tight constraints: Domain-specific accelerators
- Existing accelerators start from specific models (e.g., DNNs) or only try to accelerate specific computational patterns
  - Designing hardware accelerators by hand is complex and time-consuming
  - Hardware designer may want to explore design space for trade-offs, depending on the applications
- Agile Hardware Design & Prototyping required
  - Tools to quickly transition from algorithm formulation to the accelerator implementation, having sufficient design space exploration knobs and needing minimal human interaction

LeNet architecture [1]

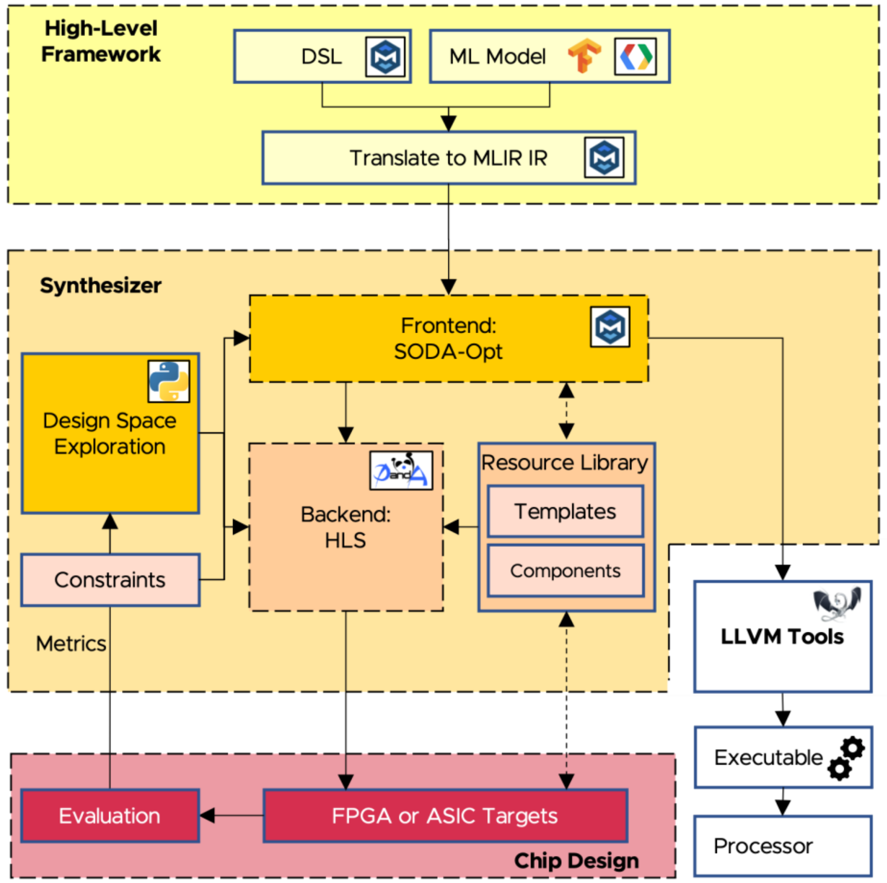






[1] Y. Lecun, *et al.*, "Gradient-based learning applied to document recognition," *Proc. IEEE*, 1998

SODA Synthesizer: Overview



[N. Bohm Agostini, *et al.*, "Bridging Python to Silicon: The SODA Toolchain," *IEEE Micro*, 2022]

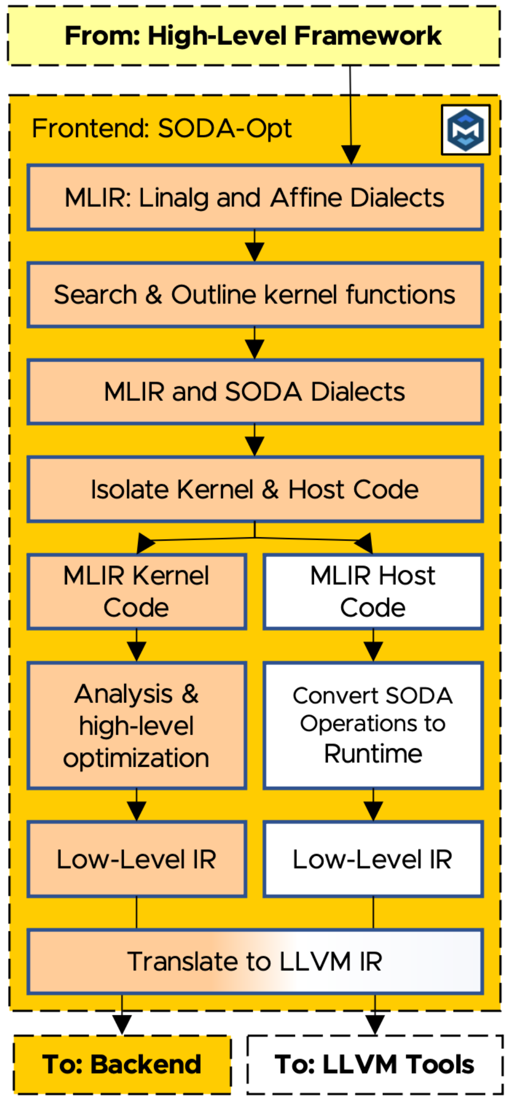
[J. J. Zhang, *et al.*, "Towards Automatic and Agile AI/ML Accelerator Design with End-to-End Synthesis," *ASAP 2021*]

[M. Minutoli, *et al.*, "SODA: a New Synthesis Infrastructure for Agile Hardware Design of Machine Learning Accelerators," *ICCAD 2020*]

- A modular, multi-level, interoperable, extensible, open-source hardware compiler from high-level programming frameworks to silicon
  - Optimizations at all levels are performed as compiler optimization passes
- Compiler-based frontend (SODA-Opt): leverages the Multi-Level Intermediate Representation (MLIR)
- Compiler-based backend (PandA-Bambu): leverages state-of-the-art High-Level Synthesis (HLS) techniques, as well as a Coarse-Grained Reconfigurable Array (CGRA) generator
  - Generates synthesizable Verilog for a variety of targets, from Field Programming Gate Arrays (FPGAs) to Application-Specific Integrated Circuits (ASICs)

SODA Synthesizer: Frontend (SODA-Opt)

- SODA-Opt: Search, Outline, Dispatch, Accelerate frontend optimizer "generates" the SODA High-Level IR
- Employs and embraces the MLIR framework
  - 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)
  - Generates interfacing code and runtime calls for microcontroller

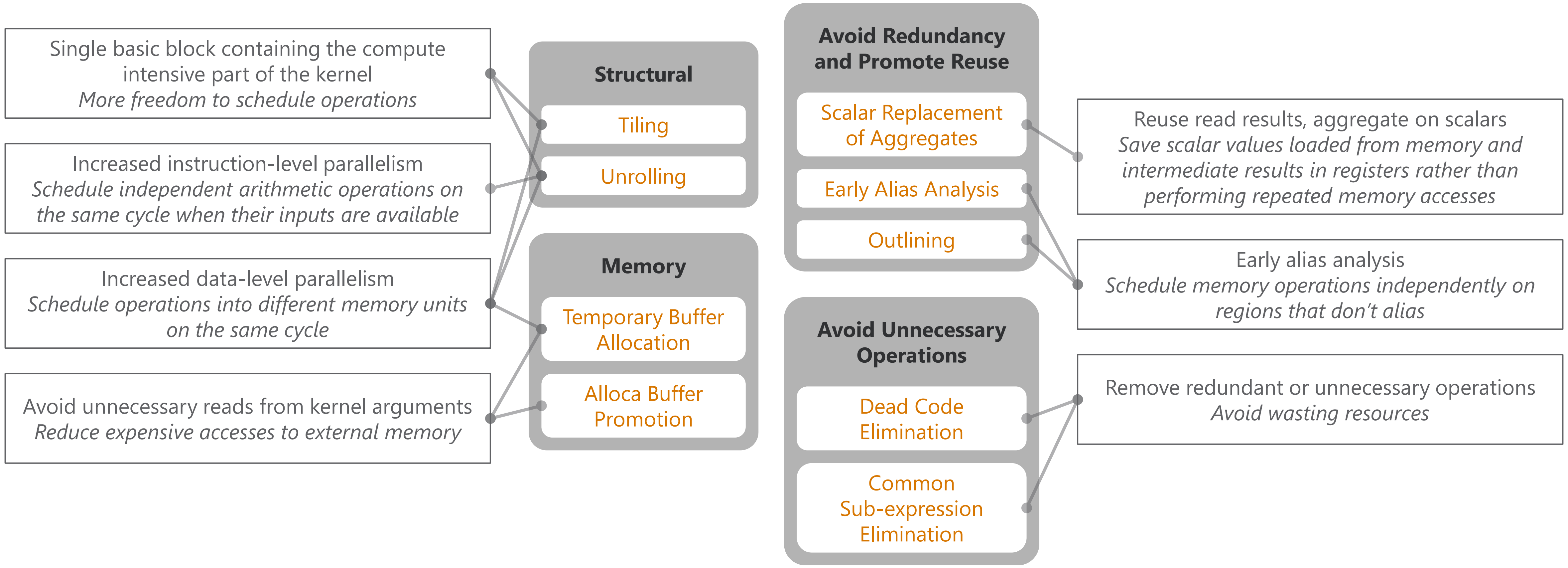

From: High-Level Framework
Frontend: SODA-Opt
MLIR: Linalg and Affine Dialects
Search & Outline kernel functions
MLIR and SODA Dialects
Isolate Kernel & Host Code
MLIR Kernel Code
MLIR Host Code
Analysis & high-level optimization
Convert SODA Operations to Runtime
Low-Level IR
Low-Level IR
Translate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

SODA-OPT: System Overview

[N. Bohm Agostini, *et al.*, "SODA-OPT: an MLIR-based flow for co-design and high-level synthesis," *CF 2022 - Best Poster Award*]

SODA-Opt Optimization Passes

- The SODA-Opt optimization passes:



Single basic block containing the compute intensive part of the kernel
More freedom to schedule operations

Increased instruction-level parallelism
Schedule independent arithmetic operations on the same cycle when their inputs are available

Increased data-level parallelism
Schedule operations into different memory units on the same cycle

Avoid unnecessary reads from kernel arguments
Reduce expensive accesses to external memory

Structural
- Tiling
- Unrolling

Memory
- Temporary Buffer Allocation
- Alloca Buffer Promotion

Avoid Redundancy and Promote Reuse
- Scalar Replacement of Aggregates
- Early Alias Analysis
- Outlining

Avoid Unnecessary Operations
- Dead Code Elimination
- Common Sub-expression Elimination

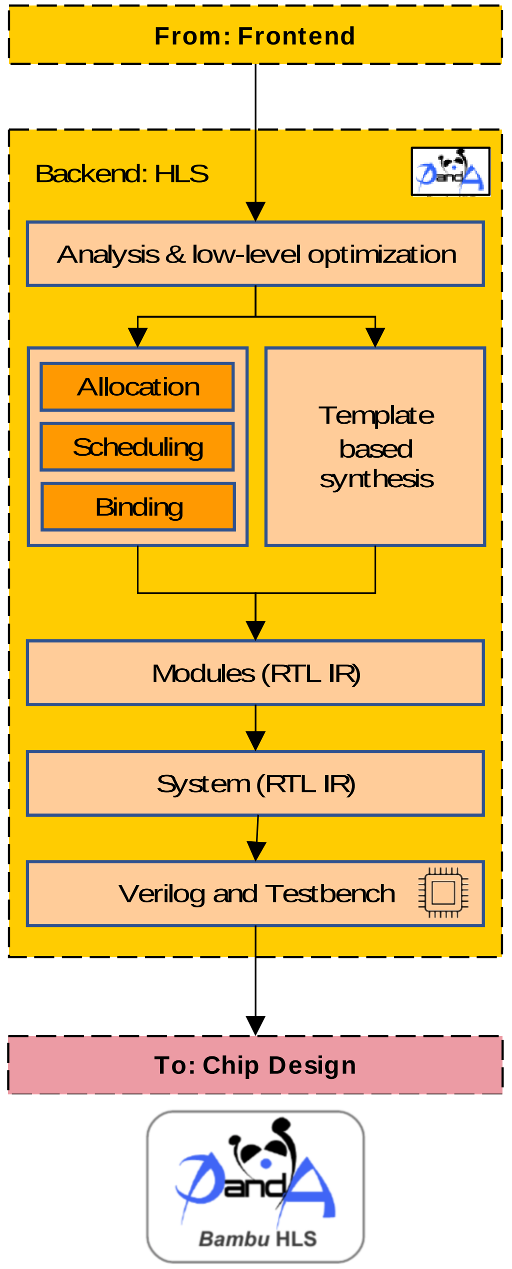
Reuse read results, aggregate on scalars
Save scalar values loaded from memory and intermediate results in registers rather than performing repeated memory accesses

Early alias analysis
Schedule memory operations independently on regions that don't alias

Remove redundant or unnecessary operations
Avoid wasting resources

SODA Synthesizer: HLS Backend (Bambu)

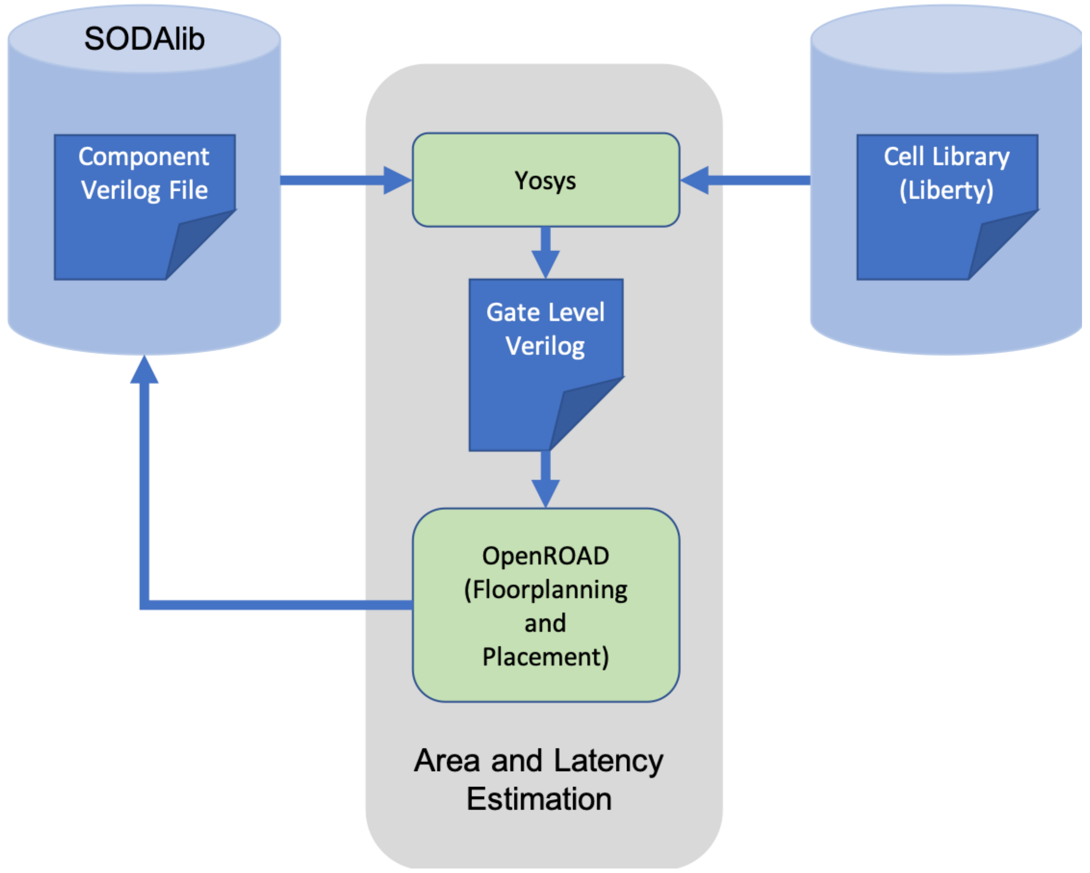
- Backend: optimized low-level IR inputs to generate hardware descriptions of the accelerators
- PandA-Bambu: open-source state-of-the-art high-level synthesis (HLS) tool as a backend
  - Key features: parallel accelerator designs, modular HLS, and ASIC support
- The HLS backend:
  - Provides automated testing and verification of the generated designs
  - Provides the necessary generality to deal with novel algorithms
  - Provides the opportunities for specialized and optimized templates by recognizing specific computational patterns
- SODA approach relies on progressive lowerings of compiler IRs, rather than rewriting annotated C/C++



[F. Ferrandi, *et al.*, Invited: "Bambu: an Open-Source Research Framework for the High-Level Synthesis of Complex Applications," *DAC 2021*]

SODA Synthesizer: Targets

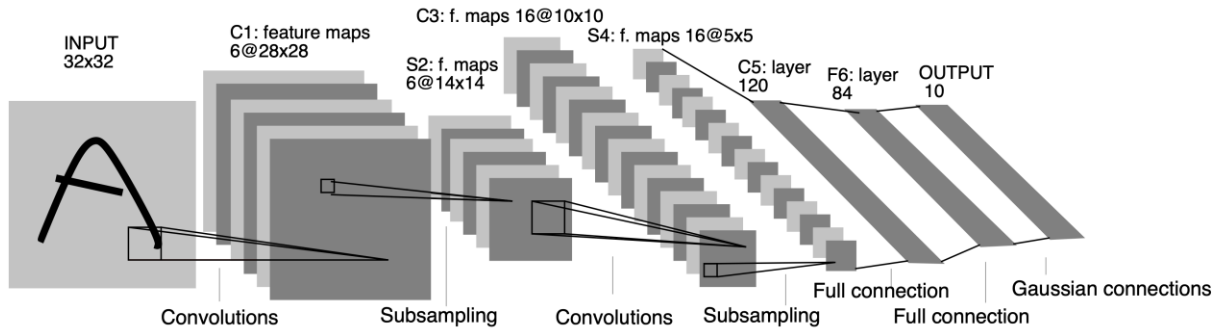
- Supports different target technologies (FPGA, CGRA, ASIC) for actual generated designs
- ASIC targets:
  - Commercial Tools (Synopsys Design Compiler with Global Foundries 12/14 nm cells)
  - OpenROAD suite (FreePDK 45nm and ASAP 7nm cell libraries)
- Backends' resources characterized for the target technology:
  - Eucalyptus tool in Bambu, allows driving hardware synthesis algorithms to optimize for area, latency, etc.
  - OpenCGRA: evaluation of the results, metrics for the design space exploration



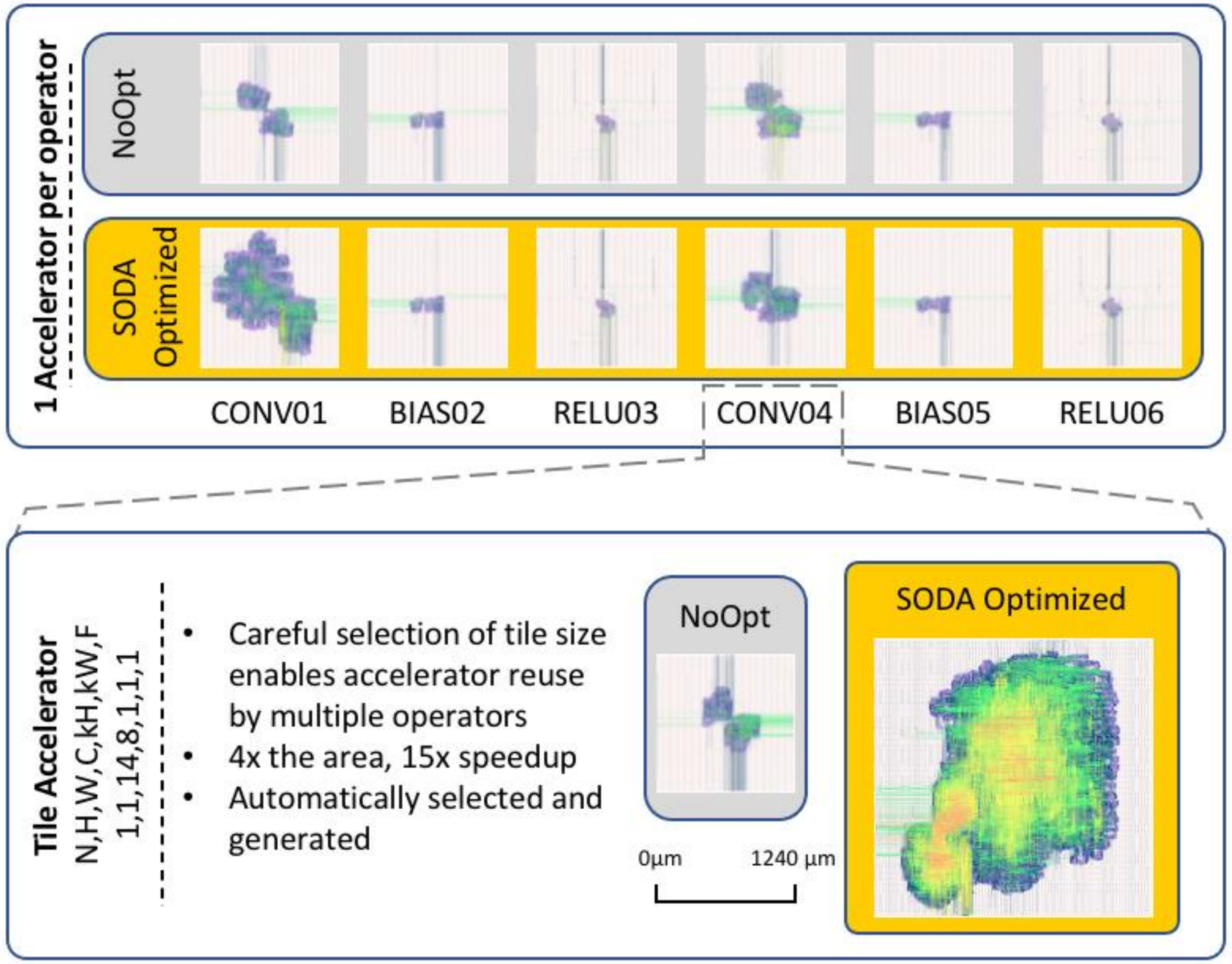
SODA characterization flow. The characterization flow can be extended to synthesize HLS generated designs, or used to estimate their area-latency-power profiles to drive the Design Space Exploration engine

From Python to optimized ASIC

LeNet architecture

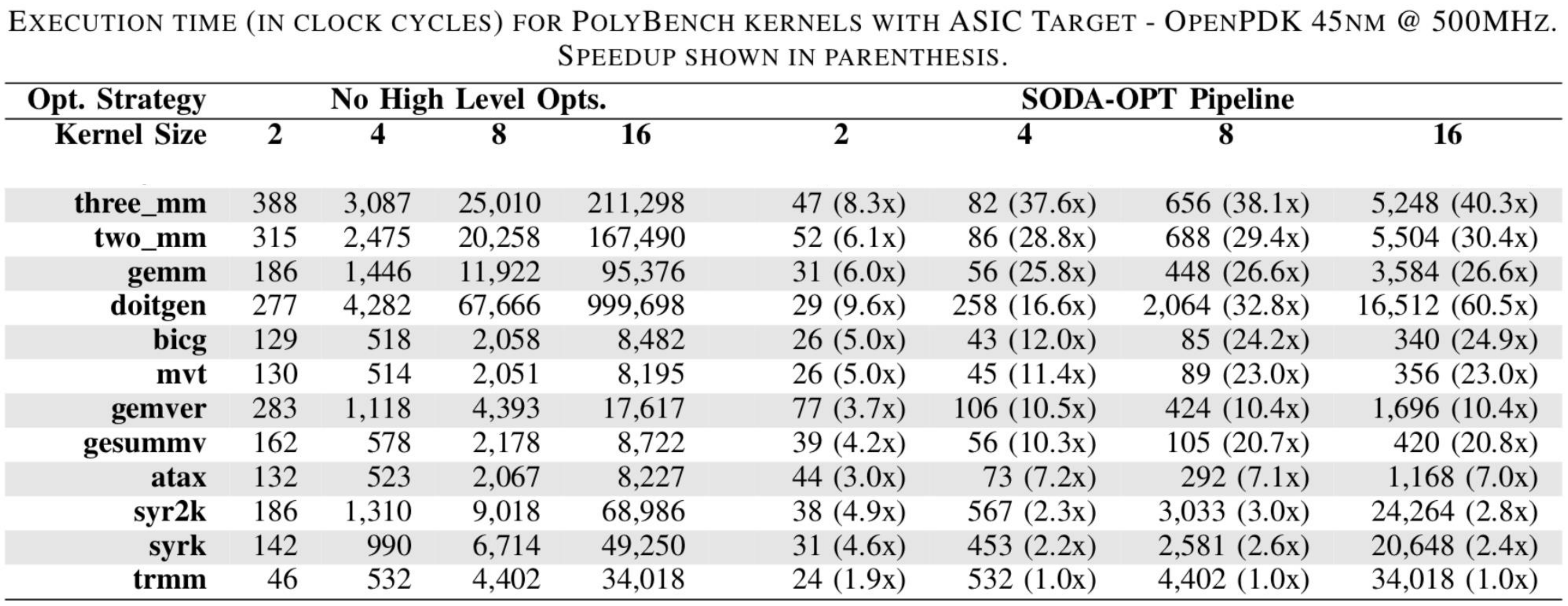


- LeNet architecture: each of the operators are synthesized to an ASIC accelerator
- SODA-Opt optimized accelerators are bigger, but also much faster



- Careful selection of tile size enables accelerator reuse by multiple operators
- 4x the area, 15x speedup
- Automatically selected and generated

ASIC Generation: Linear Algebra Kernels



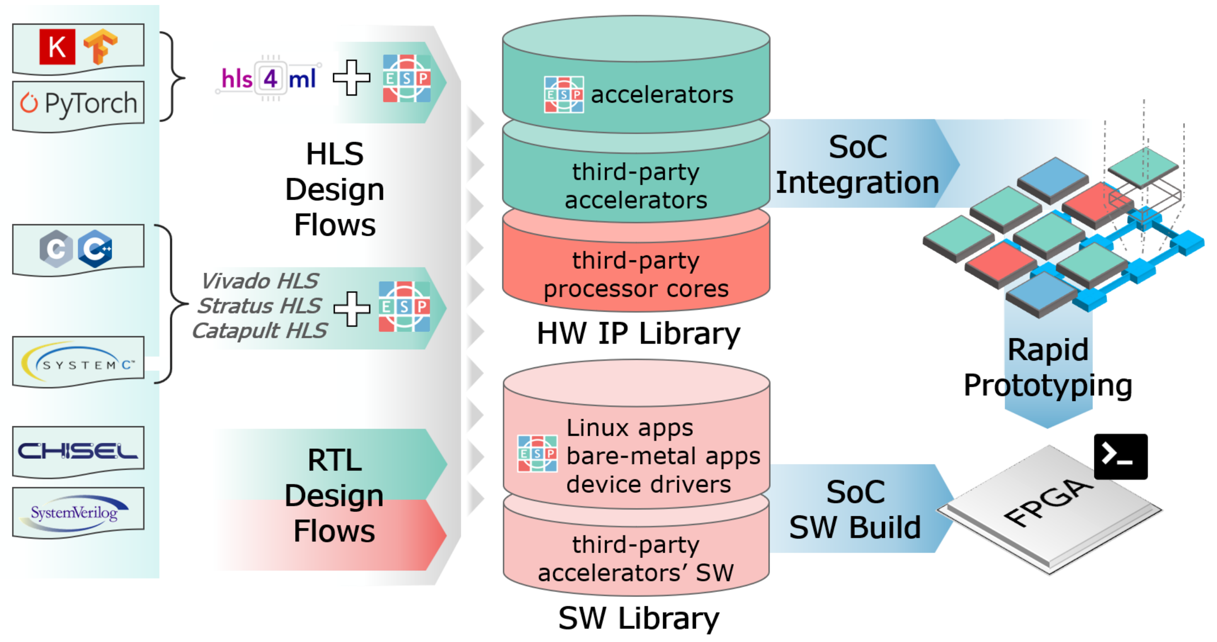
EXECUTION TIME (IN CLOCK CYCLES) FOR POLYBENCH KERNELS WITH ASIC TARGET - OPENPDK 45NM @ 500MHZ. SPEEDUP SHOWN IN PARENTHESIS.

Opt. Strategy	No High Level Opts.				SODA-OPT Pipeline			
Kernel Size	**2**	**4**	**8**	**16**	**2**	**4**	**8**	**16**
three_mm	388	3,087	25,010	211,298	47 (8.3x)	82 (37.6x)	656 (38.1x)	5,248 (40.3x)
two_mm	315	2,475	20,258	167,490	52 (6.1x)	86 (28.8x)	688 (29.4x)	5,504 (30.4x)
gemm	186	1,446	11,922	95,376	31 (6.0x)	56 (25.8x)	448 (26.6x)	3,584 (26.6x)
doitgen	277	4,282	67,666	999,698	29 (9.6x)	258 (16.6x)	2,064 (32.8x)	16,512 (60.5x)
bicg	129	518	2,058	8,482	26 (5.0x)	43 (12.0x)	85 (24.2x)	340 (24.9x)
mvt	130	514	2,051	8,195	26 (5.0x)	45 (11.4x)	89 (23.0x)	356 (23.0x)
gemver	283	1,118	4,393	17,617	77 (3.7x)	106 (10.5x)	424 (10.4x)	1,696 (10.4x)
gesummv	162	578	2,178	8,722	39 (4.2x)	56 (10.3x)	105 (20.7x)	420 (20.8x)
atax	132	523	2,067	8,227	44 (3.0x)	73 (7.2x)	292 (7.1x)	1,168 (7.0x)
syr2k	186	1,310	9,018	68,986	38 (4.9x)	567 (2.3x)	3,033 (3.0x)	24,264 (2.8x)
syrk	142	990	6,714	49,250	31 (4.6x)	453 (2.2x)	2,581 (2.6x)	20,648 (2.4x)
trmm	46	532	4,402	34,018	24 (1.9x)	532 (1.0x)	4,402 (1.0x)	34,018 (1.0x)

- Results for 14 linear algebra kernels from PolyBench demonstrates the effectiveness of the end-to-end flow and high-level optimizations
- The SODA Synthesizer generates ASICs for all the provided kernels
- In most cases, SODA-Opt optimization pipeline provide significant speedups

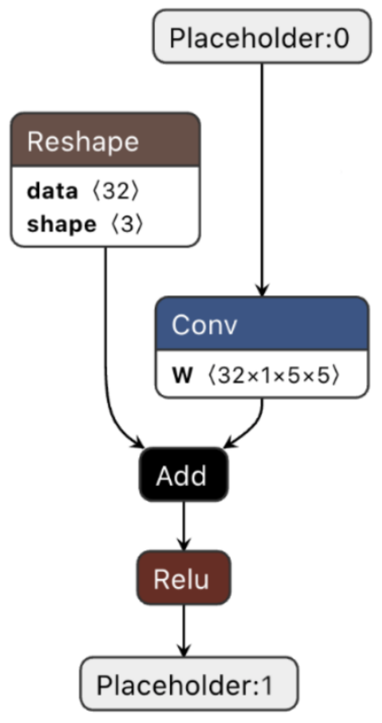
Research Opportunities: System-Level Design

- Integrating with open-source fast prototyping platforms: Columbia University Embedded Scalable Platform (ESP)
- SODA-Opt
  - MLIR is naturally modular and hierarchical
  - Can lower to multiple targets, including runtimes
- Bambu
  - Provides a fully open-source HLS backend for ESP
- Can enable end-to-end fast prototyping from algorithmic concept to system implementation



[P. Mantovani, *et al.*,: "Agile SoC Development with Open ESP," *ICCAD 2020*]

Research Opportunities: Dataflow architecture



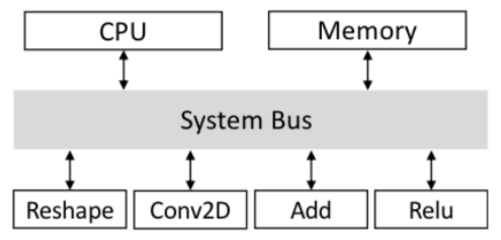
(a) Input model

```
func @main(...) {
  ...
  linalg.generic #reshape_trait ... {
  ^bb0(...):
    linalg.yield %arg2 : f32
  }
  linalg.generic #conv2d_trait ... {
  ^bb0(...):
    %7 = arith.mulf %arg2, %arg3 : f32
    %8 = arith.addf %arg4, %7 : f32
    linalg.yield %8 : f32
  }
  linalg.generic #bias_trait ... {
  ^bb0(...):
    %7 = arith.addf %arg2, %arg3 : f32
    linalg.yield %7 : f32
  }
  linalg.generic #relu_trait ... {
  ^bb0(...):
    %7 = arith.cmpf olt, %arg2, %cst_0 : f32
    %8 = select %7, %cst_0, %arg2 : f32
    %9 = arith.cmpf olt, %cst, %arg2 : f32
    %10 = select %9, %cst, %8 : f32
    linalg.yield %10 : f32
  }
  return
}
```

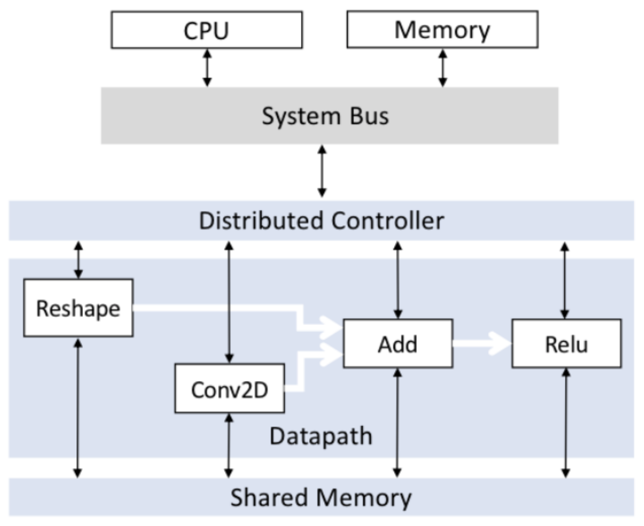
(b) MLIR lowering (simplified)

```
func @main(...) {
  ...
  soda.launch_func @df0_reshape::@f args(...)
  soda.launch_func @df1_conv2d::@f args(...)
  soda.launch_func @df2_add::@f args(...)
  soda.launch_func @df3_relu::@f args(...)
return

  soda.func @f(...) {
    linalg.generic #reshape_trait // ...
    soda.return
  }

  soda.func @f(...) {
    linalg.generic #conv2d_trait // ...
    soda.return
  }

  soda.func @f(...) {
    linalg.generic #bias_trait // ...
  }

  soda.func @f(...) {
    linalg.generic #relu_trait // ...
    soda.return
  }
```

(c) After search and outlining (simplified)



(d) Centralized architecture

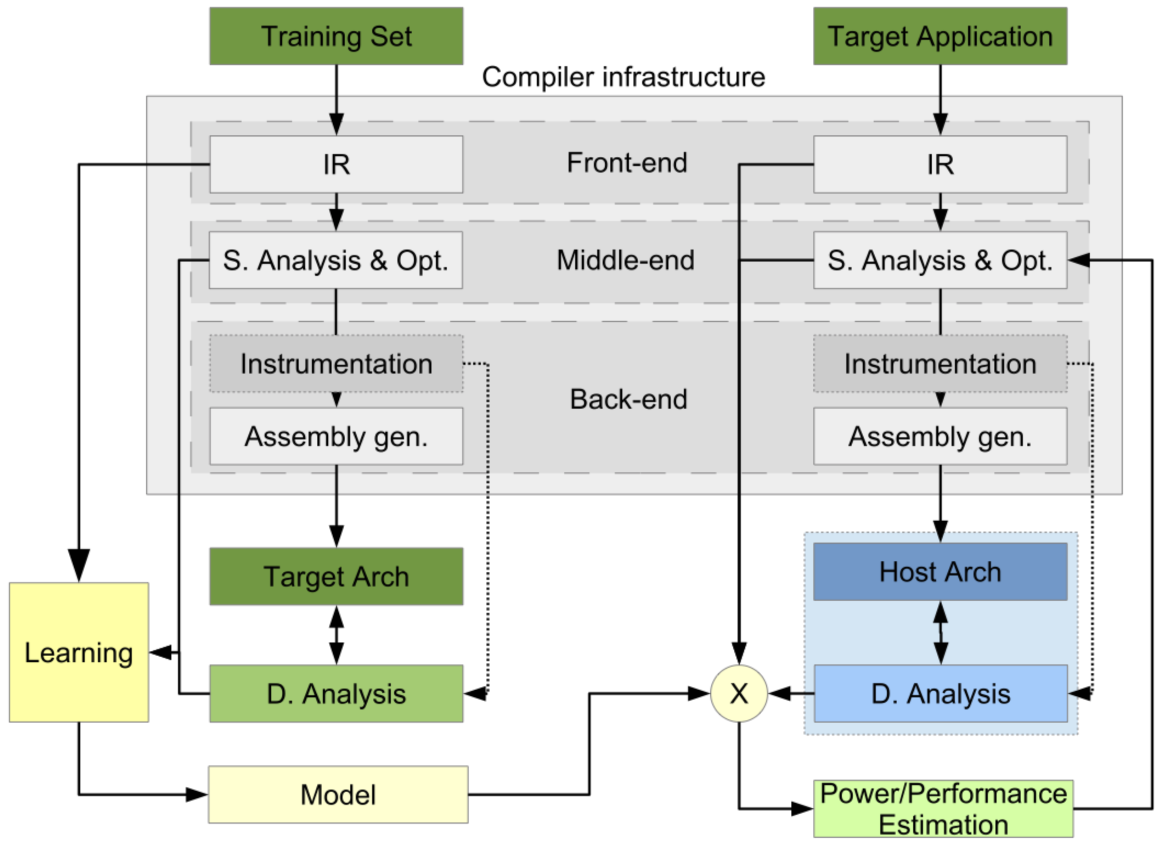


(e) Dataflow architecture

- SODA provides a methodology to translate outlined kernels in two architectures:
  - A centralized architecture with a microprocessor
  - A dynamically scheduled, automatically generated, dataflow architecture

Research Opportunities: Profile-driven Synthesis

- Compiler infrastructure:
  - Provides ecosystem for static & dynamic analysis
- Dynamic analysis possible by automated instrumentation and profiling
  - E.g., capturing data-dependent patterns and memory transactions
  - Information can be fed back to the synthesis engine to facilitate design space exploration of the memory and the overall architecture design



[A. Tumeo, *et al.*, "Architecture independent integrated early performance and energy estimation," *IGSC 2017*]

Research Opportunities: Effective memory hierarchy & interfaces

- Compiler-based instrumentation can be used to capture dynamic memory traces
  - Synthesis engine can use the information to balance computational intensity and memory latency
- Tolerate memory access latencies
  - Manage computational intensity: change scheduling algorithms
  - Optimize memory hierarchy & interfaces
- Optimizing for memory accesses
  - Design effective memory hierarchy (e.g., include data buffers, scratchpads, prefetch engines)
  - Compiler optimizations for data restructuring (e.g., array partitioning, matrix transpose) to effectively utilize underlying memory hierarchy
  - Hierarchical memory interface that routes signals from different accelerators to a multi-port/multi-bank shared memory to maximize the available bandwidth utilization

Conclusions

- SODA: end-to-end compiler-based toolchain for generating domain-specific accelerators
  - Modular, multi-level, and extensible
  - Completely based on interoperating open-source technologies
  - Can target reconfigurable architectures (e.g., FPGAs, CGRAs) as well as 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 Tutorial: *DATE 2022*

SODA Docker Image

SODA-Opt

Panda-Bambu HLS (v 0.9.7)

Acknowledgements

- This work was partially supported by:
  - The US DOE Office of Science project: "*Advanced Memory to support Artificial Intelligence for Science*" at PNNL
  - The "*Software Defined Architectures for Data Analytics*" (SO(DA)^2) project under PNNL Data-Model Convergence Laboratory Directed Research & Development
  - The "*Software Defined Accelerators from Learning Tools Environment*" (SODALITE) project under the DARPA Real Time Machine Learning (RTML) program

Pacific Northwest
NATIONAL LABORATORY

Thank you!