Pacific Northwest
NATIONAL LABORATORY

Towards Automated Generation of Chiplet-Based Systems

Ankur Limaye, Claudio Barone,
Nicolas Bohm Agostini, Marco Minutoli,
Joseph Manzano, Vito Giovanni Castellana,
Giovanni Gozzi, Michele Fiorito,
Serena Curzel, Fabrizio Ferrandi, Antonino Tumeo

Antonino Tumeo

Chief Scientist, High Performance Computing Group

U.S. DEPARTMENT OF ENERGY BATTELLE

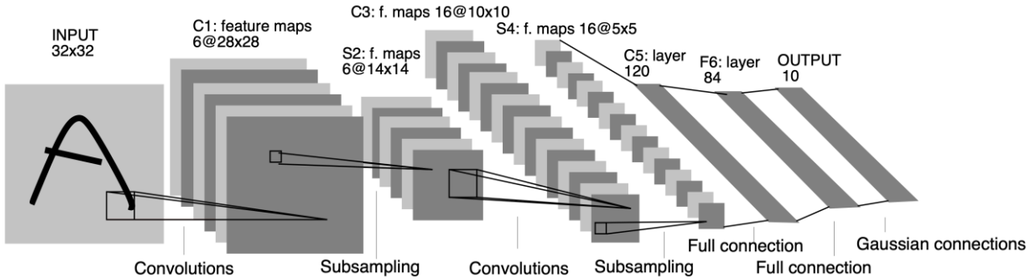
PNNL is operated by Battelle for the U.S. Department of Energy

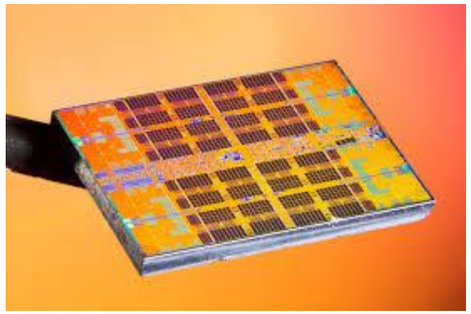
Motivations

- 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…

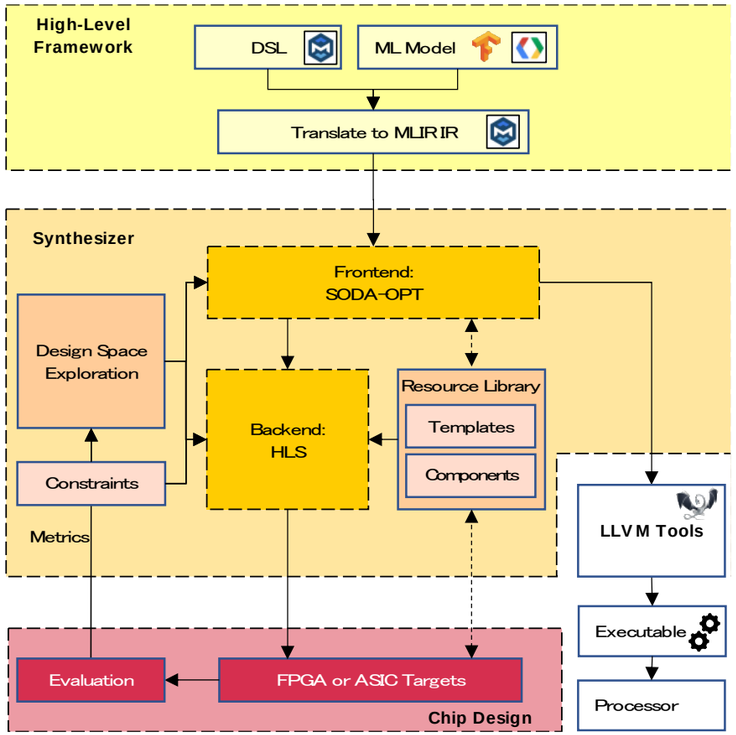
- ***Need tools to quickly transition from formulation of an algorithm to the accelerator implementation and explore the accelerator design along different dimensions***

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, as well as a Coarse-Grained Reconfigurable Array (CGRA) generator
- 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

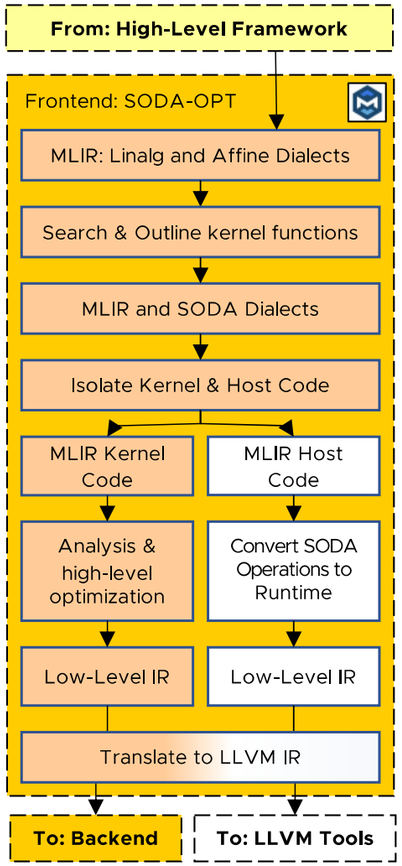
[N. Bohm Agostini, S. Curzel, J. Zhang, A. Limaye, C. Tan, V. Amatya, M. Minutoli, V.G. Castellana, J. Manzano, A. Tumeo: Bridging Python to Silicon: The SODA Toolchain. IEEE Micro Magazine 2022]

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, D. Kaeli, A. Tumeo: SODA-OPT: System-Level Design in MLIR for HLS. SC 21 Poster]

[N. Bohm Agostini, S. Curzel, V.C. 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 2022]

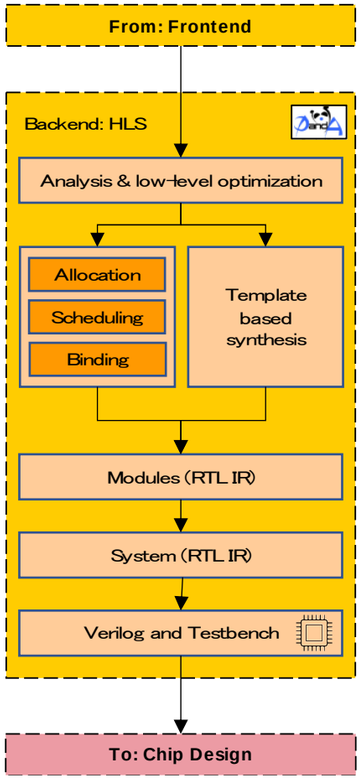


SODA-OPT: System Overview

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
- Note: SODA-OPT now also supports output to commercial HLS tools (Vits-HLS)



Bambu HLS

https://panda.dei.polimi.it

[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]

Pacific Northwest
NATIONAL LABORATORY

Why an HLS Backend?

- Provides the necessary **generality** to deal with novel algorithms
- Provides opportunities for **specialized** and optimized templates by recognizing specific computational patterns

- The SODA Approach relies on **progressive lowerings** of compiler intermediate representations (IRs), rather than rewriting annotated C/C++
  - Reduces semantic mismatches between high-level and low-level descriptions
  - Provides further opportunities to apply optimizations at the right level of abstraction

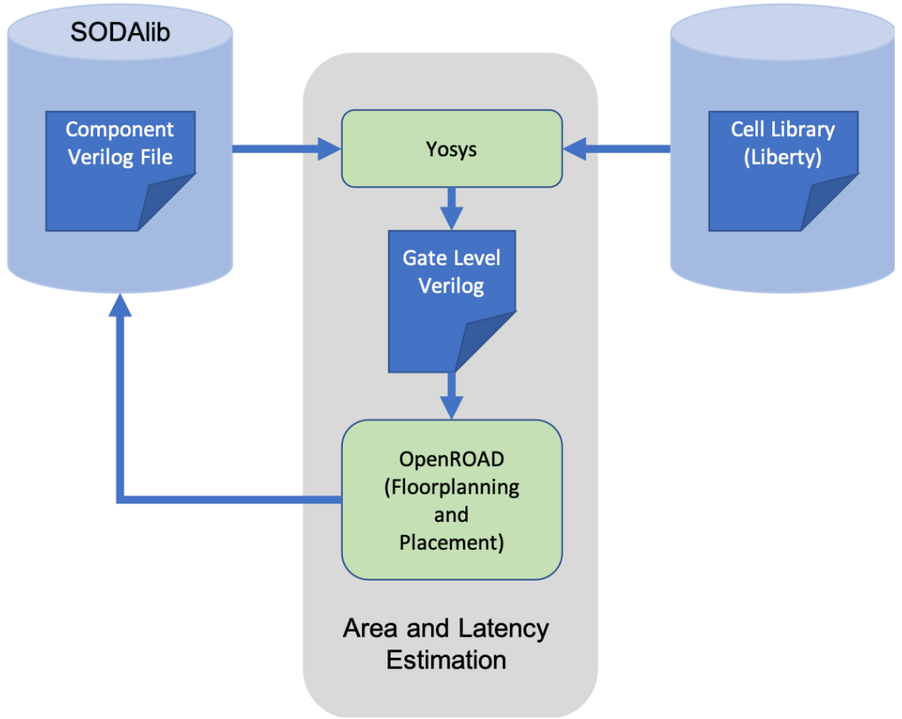
- New optimizations as additional compiler passes
- Design space exploration formulated as a compiler optimization problem

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

- ASIC targets:
  - **Commercial Tools** (Synopsys Design Compiler with Global Foundries 12/14 nm cells)
  - **OpenROAD suite** (OpenPDK 45nm and ASAP 7nm cell libraries)

- Backend' resources characterized for the target technology:
  - **Eucalyptus** tool in Bambu, allows driving hardware synthesis algorithms to optimize for area, latency, etc



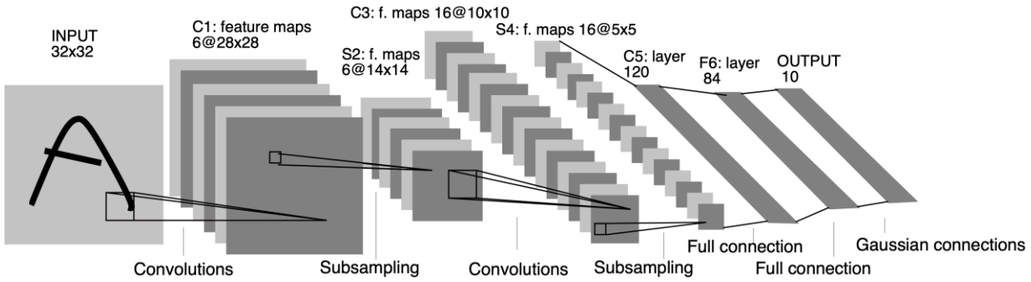
SODA characterization flow. The characterization flow can be extended to synthesize HLS generated designs, or used to estimate their area-latency-power profiles to drive the Design Space Exploration engine

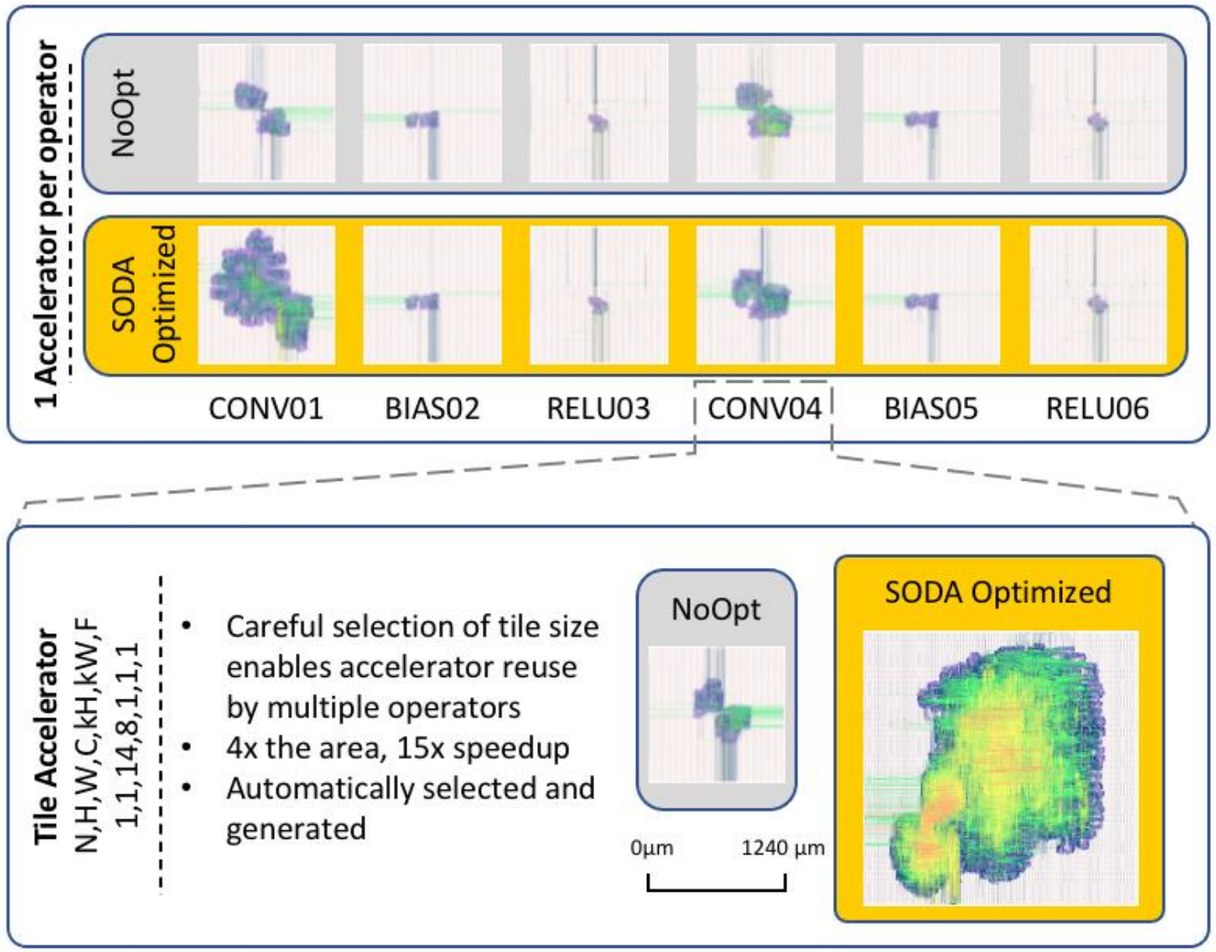
OpenROAD

https://theopenroadproject.org

From Python to optimized ASIC

LeNet architecture from the original paper





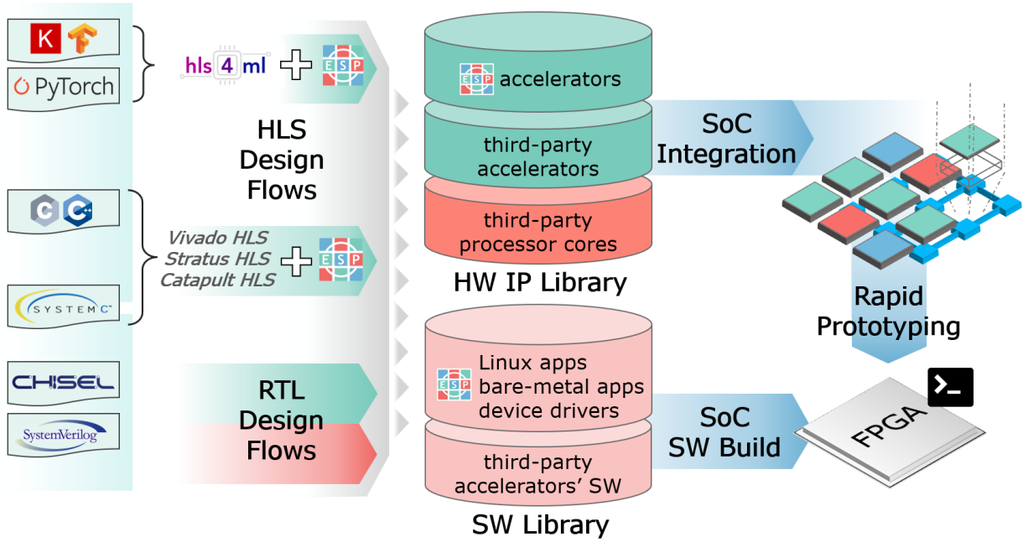
- LeNet example
- Each of the operator is synthesized to an **ASIC accelerator** using OpenROAD and FreePDK 45 nm
- SODA-Opt optimized accelerators are bigger, but also much **faster**

Towards Chiplets Integration with SODA

- SODA does not take care of the physical implementation
- However, it needs to support standard (electrical) interfaces and protocols
- Electrical Interfaces
  - Intel Advanced Interface Bus (ABI), provided as IP in the DARPA CHIPS program
  - Universal Chip Interface Express (UCIe), standard specification
- Protocols
  - Compute eXpress Link (CXL)
  - Advanced eXtensible Interface (AXI)
- Connect to electrical interface and support appropriate protocols on top

AXI Interfaces

- Bambu now generates ports supporting the AXI protocols:
  - Master port
  - Slave port
  - AXI-STREAM Interface
- AXI typically used in FPGAs (e.g., Xilinx) to interface with memory controllers
- AXI can be used to interface accelerators generated with our synthesis flow through tiled-based prototyping platforms
  - E.g., Columbia University Embedded Scalable Platforms (ESP)
- AXI also been used as communication protocol between chiplets
  - E.g., with Intel's AIB



Pacific Northwest NATIONAL LABORATORY

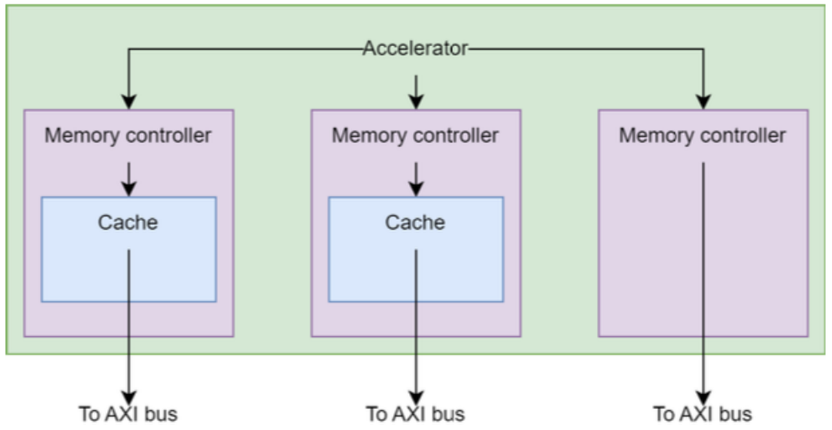
Improving Memory Access through AXI

- Conventional HLS tools require code restructuring to enable AXI burst transfers
  - Limited to a specific burst size
  - Applicable only to certain code patterns
- A possible approach, when the accelerator needs to directly access memory, is to implement specialized prefetch and burst buffers or caches
  - However, accelerators are highly specific
  - Cache and buffers needs way to be properly configured depending on the synthesized kernel

AXI Caches Design

- Design inspired by IOb caches
- Customizable along several parameters:
  - Size, ways, behaviors (write-back/write-through)
- AXI master interface can read or write transactions in bursts
  - Size of an entire cache line
- Can generate a specialized cache for each memory channel of the accelerator
  - (e.g., each function arguments, if it is a pointer to a data structure in memory)
- Can add caches in two ways:
  - C/C++ pragmas (implemented with Clang plugin)
  - Parameters determined by SODA-OPT Analysis and feed to Bambu through a configuration file

```
48  #pragma HLS_interface a m_axi direct bundle = gmem0
49  #pragma HLS_interface b m_axi direct bundle = gmem1
50  #pragma HLS_interface output m_axi direct bundle = gmem2
51
52  #pragma HLS_cache bundle = gmem0 way_size = 16 line_size = 16
53  #pragma HLS_cache bundle = gmem1 way_size = 16 line_size = 16
54  #pragma HLS_cache bundle = gmem2 way_size = 16 line_size = 16
55  void mmult(int* a, int* b, int* output)
```



Schematic of an accelerator with three memory channels (two with caches, one cacheless)

[C. Barone, G. Gozzi, M. Fiorito, A. Limaye, A. Tumeo, F. Ferrandi. "Improving Memory Interfacing in HLS-Generated Accelerators with Custom Caches", SC23 Best Research Poster Candidate.]

Pacific Northwest NATIONAL LABORATORY

AXI Caches Design – Additional Considerations

- Our design supports outstanding write requests
  - Can initiate a new write transactions before receiving response of the previous one, further reducing channel latency
- Our design includes a flushing mechanism
  - Can write back dirty cache lines to external memory before accelerator signals completion of the execution
- No coherency mechanisms required
  - User or analysis guarantees that ports operate on data in different memory regions (no pointer aliasing and data sharing)

Pacific Northwest NATIONAL LABORATORY

AXI Caches – Experimental Setup

- Synthesized five kernels from the PolyBench suite
  - 2mm, atax, bigc, doitgen, mvt.
- Simulated the generated accelerators with Verilator to compare execution delays with and without caches
  - Sizes starting from 16 to 256 words of 4 bytes each
  - Varying external memory latency from 5 to 50 cycles
- Synthesized kernels for a Virtex 7 FPGA using Vivado 2020.2 to compare resource utilization
- Inputs: 10 elements for every vector, 10x10 elements for every matrix
- Instantiate a different AXI port for each input matrix unless there is no cache contention

Experimental Results – All Benchmarks, 50 clock cycles latency

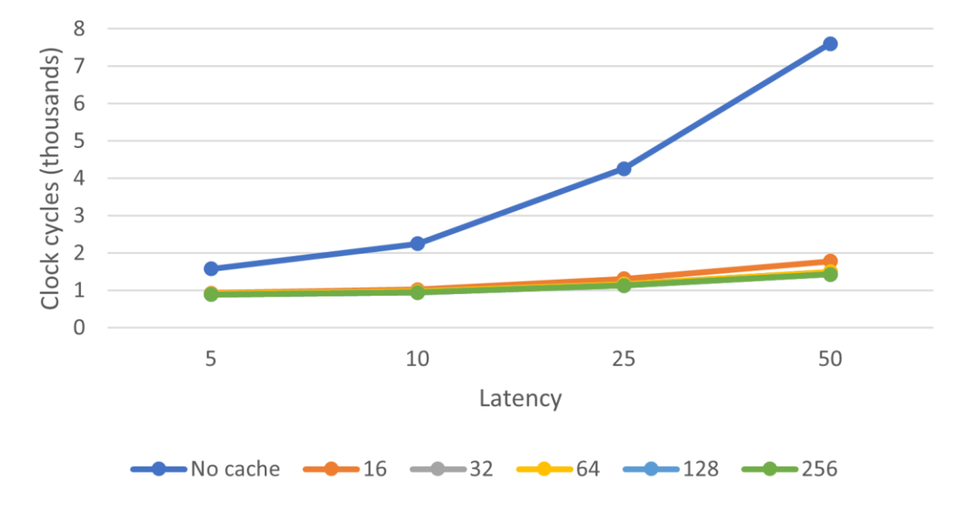
TABLE I: Resource utilization overhead (for registers, LUTs - R,L) and speed up (as execution delay in clock cycles - C) with 50 clock cycles of memory latency - 2mm and doitgen

	No cache			16, 256			32, 256			64, 256			128, 256			256, 256			256, 16			256, 32			256, 64			256, 128		
	L	R	C	L	R	C	L	R	C	L	R	C	L	R	C	L	R	C	L	R	C	L	R	C	L	R	C	L	R	C
2mm	6671	6629	136161	1.36	1.19	9.1	1.44	1.27	9.75	1.45	1.27	9.75	1.44	1.28	9.75	1.42	1.27	9.75	1.36	1.24	1.01	1.42	1.28	1.31	1.42	1.27	1.61	1.43	1.27	9.75
doitgen	4520	4901	739564	1.36	1.18	9.62	1.49	1.23	10.05	1.45	1.23	10.05	1.46	1.23	10.05	1.44	1.23	10.05	1.32	1.17	1.07	1.46	1.23	1.36	1.45	1.23	1.56	1.45	1.23	10.05

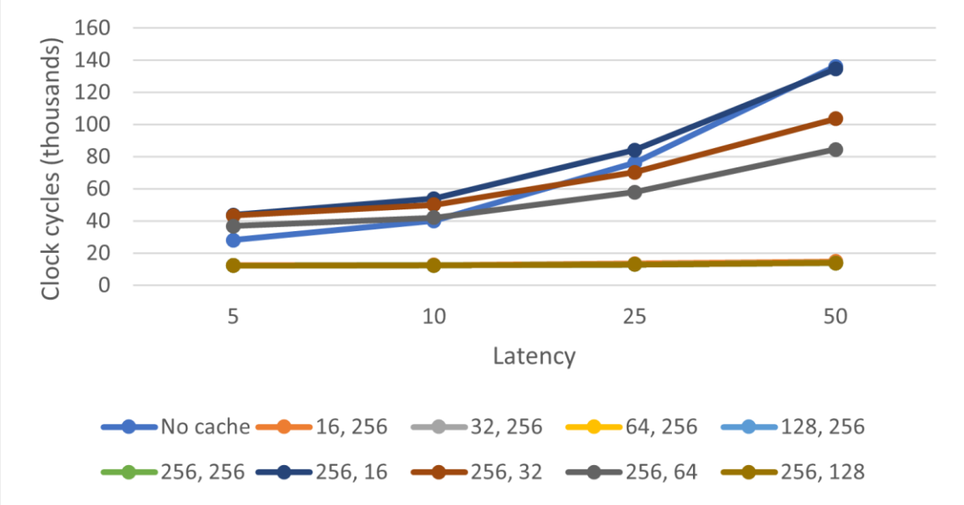
TABLE II: Resource utilization overhead (for registers, LUTs - R,L) and speed up (as execution delay in clock cycles - C) with 50 clock cycles of memory latency - atax, bicg, mvt

	No cache			16			32			64			128			256		
	L	R	C	L	R	C	L	R	C	L	R	C	L	R	C	L	R	C
atax	7171	6393	7606	1.05	1.03	4.27	1.11	1.07	4.79	1.11	1.07	5.08	1.11	1.07	5.33	1.11	1.07	5.33
bicg	8094	6855	7332	1.06	1.04	2.97	1.11	1.07	3.32	1.11	1.07	3.94	1.24	1.07	4.09	1.11	1.07	6.37
mvt	4787	5513	14680	1.07	1.06	1.48	1.16	1.10	1.90	1.17	1.10	2.27	1.16	1.10	4.10	1.16	1.10	7.36

Experimental Results – atax and 2mm, full latency sweep



atax



2mm

Pacific Northwest
NATIONAL LABORATORY

Research Opportunities: Open-Source Ecosystem

- SODA demonstrates how several Open-Source tools can seamlessly integrate
- SODA also provides initial support to commercial backends:
  - SODA-OPT generated LLVM IR can already be fed to Xilinx Vitis HLS
  - SODA also targets commercial ASIC logic synthesis tools
- Integration of proprietary tools, however, still is a significant challenges
- Significant opportunities in supporting:
  - Open-source intellectual property (IP) blocks as components in the resource libraries
  - Open-source system prototyping platforms
  - Open-source domain-specific FPGA generators to enable specialization starting from the high-level specifications
- Enabling generation and composition of highly specialized accelerator chiplets
  - Several techniques for host-to-accelerator and accelerator-to-accelerator communication in development

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: *DATE 2022*

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
- We are extending the framework to enable automated generation of specialized accelerators chiplets
- **The SODA Synthesizer provides a no-human-in-the-loop toolchain from algorithmic formulation to hardware implementation for complex workloads**

Pacific Northwest
NATIONAL LABORATORY

Thank you