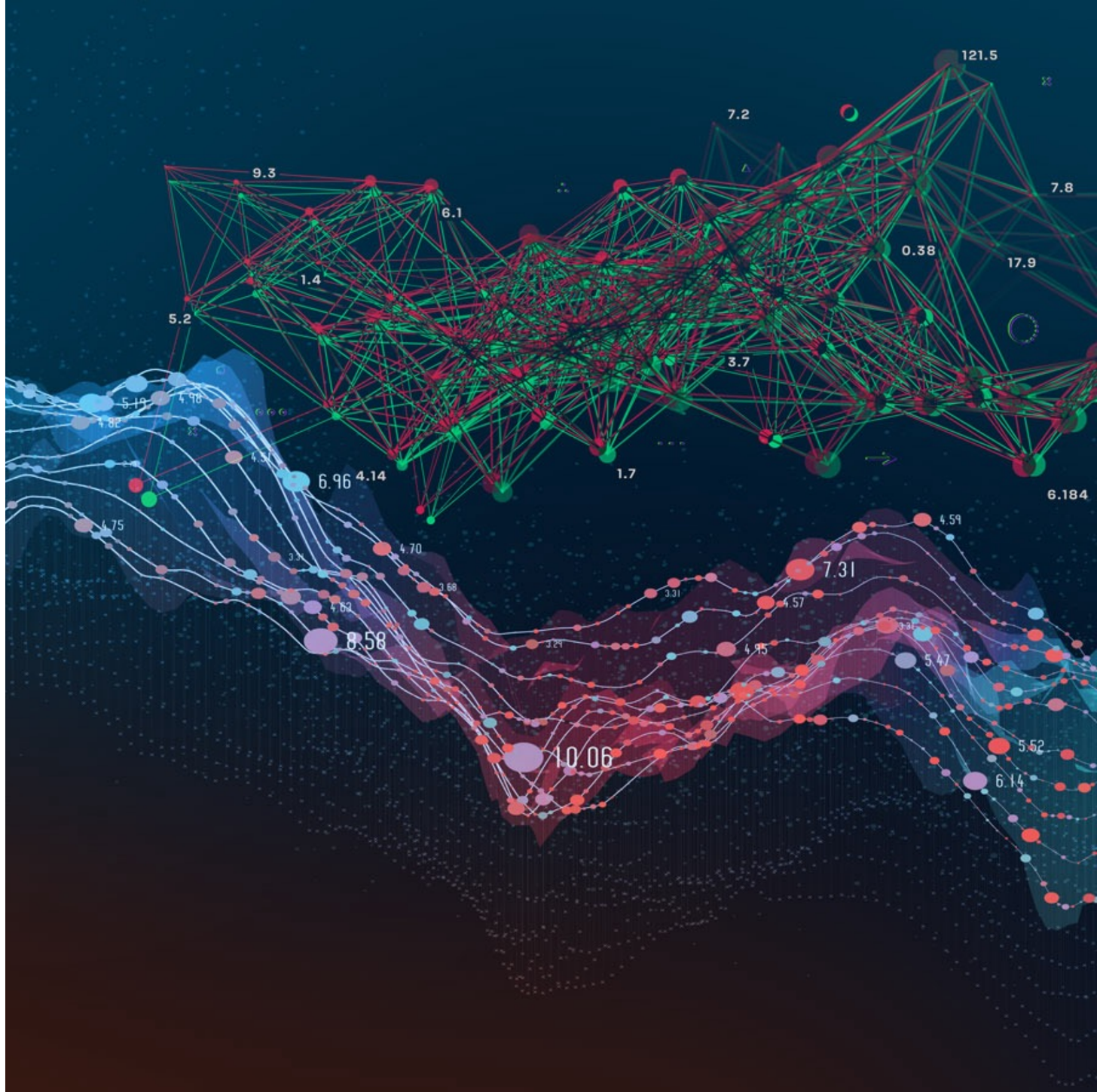
Pacific Northwest
NATIONAL LABORATORY

AI for Hardware Synthesis and Hardware Synthesis for AI



Antonino Tumeo

Chief Scientist, High Performance Computing Group

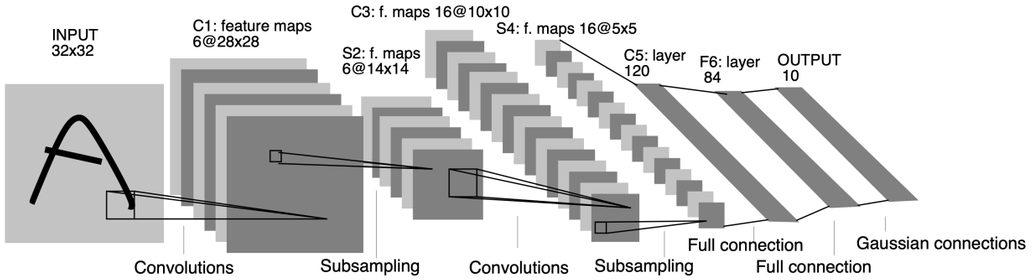
U.S. DEPARTMENT OF ENERGY BATTELLE

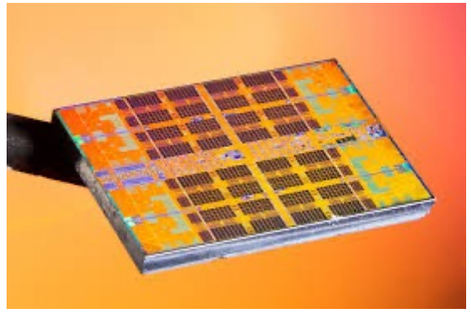
PNNL is operated by Battelle for the U.S. Department of Energy

Motivations

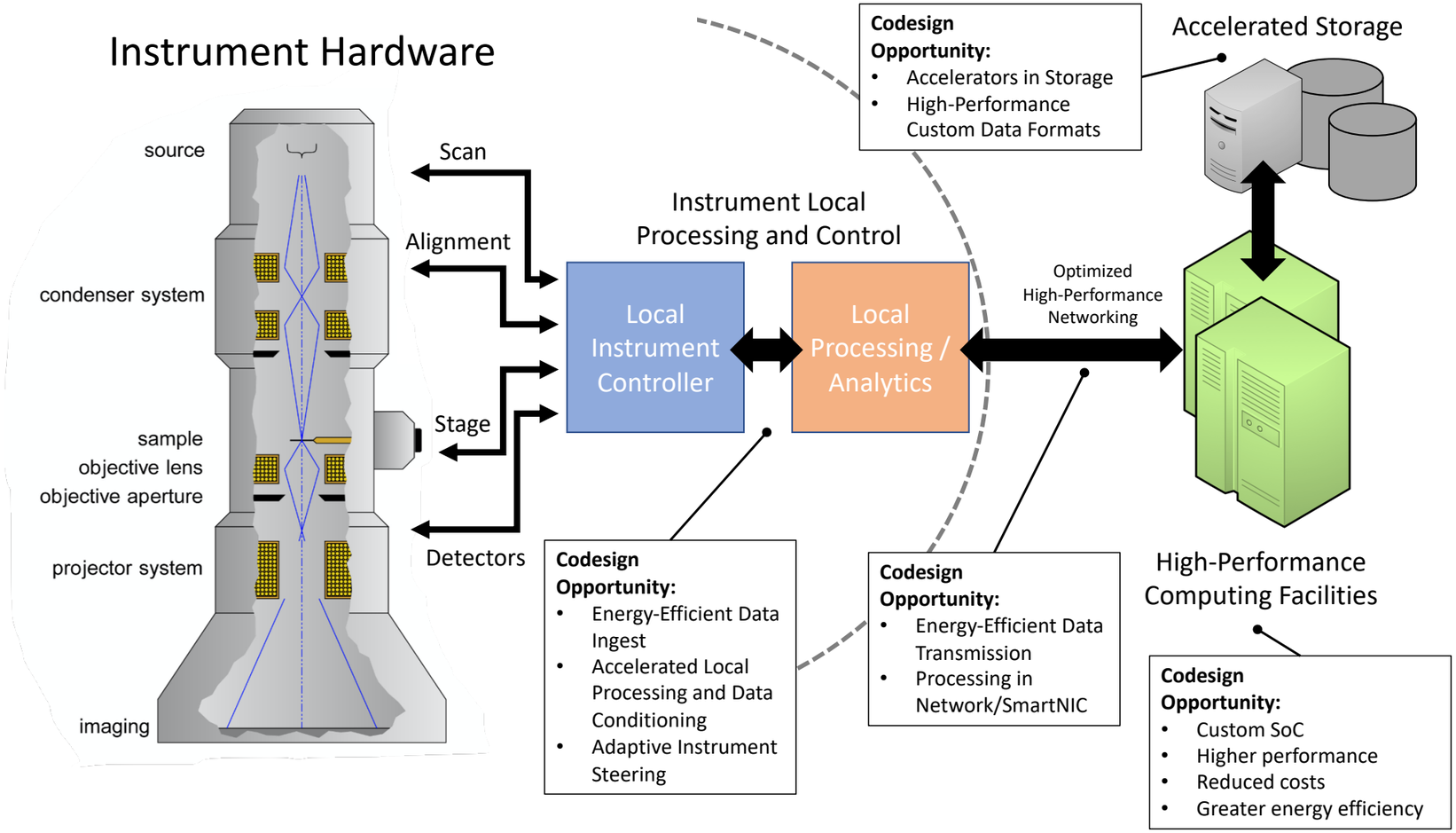
- Data science algorithms, approaches, and frameworks are quickly evolving
- Domain-specific accelerators are the only possible approach to keep increasing performance in tight constraints
- Existing accelerators start from specific models (i.e., mostly deep neural networks) or only try to accelerate 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***
- ***Artificial Intelligence and Machine Learning have become a key enabler for design exploration and optimizaiton***

LeNet architecture from the original paper

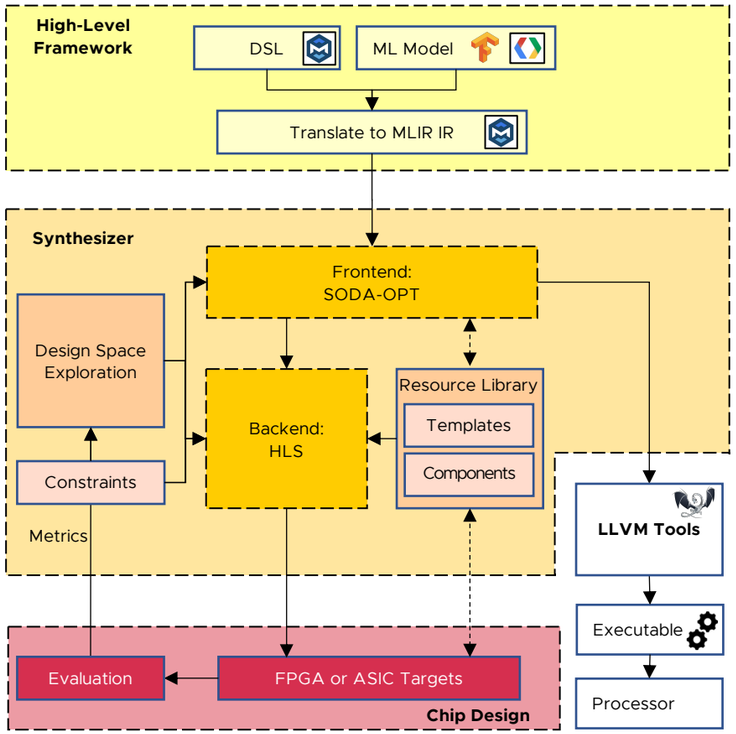




An example: AI for Synthesis of (AI) Accelerators



SODA Synthesizer: Overview



- 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

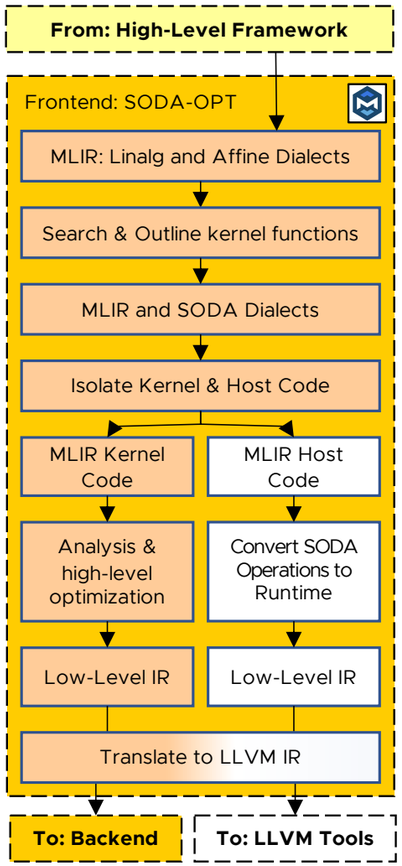
[M. Minutoli, V. G. Castellana, C. Tan, J. Manzano, V. Amatya, A. Tumeo, D. Brooks, G-Y. Wei: SODA: a New Synthesis Infrastructure for Agile Hardware Design of Machine Learning Accelerators. ICCAD 2020: 98:1-98:7]

[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]

[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



SODA-OPT: System Overview

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

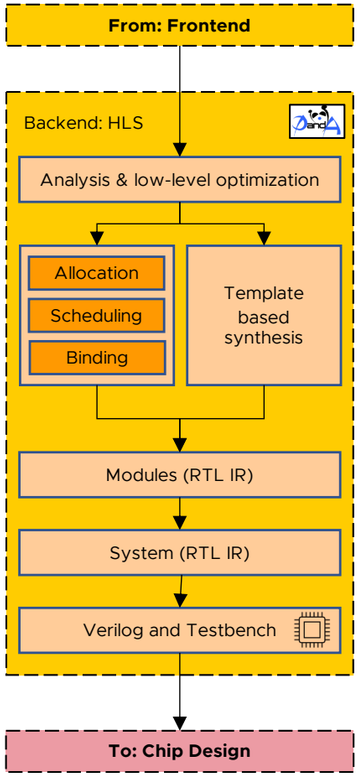
[N. Bohm Agostini, S. Curzel, V.C. Amatya, C. Tan, M. Minutoli, V. G. Castellana, J. Manzano, D. Kaeli, A. Tumeo, An MLIR-based Compiler Flow for System-Level Design and Hardware Acceleration. To appear at ICCAD 2022]

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

Pacific Northwest NATIONAL LABORATORY

SODA Synthesizer: HLS Backend

- The synthesizer backend take as input the properly optimized low-level IR and generate the hardware descriptions of the accelerators
  - SODA supports an open-souce HLS backend, PandA-Bambu, and the commercial AMD/Xilinx Vitis HLS
- PandA-Bambu is an open-source state-state-of-the-art high-level synthesis (HLS)
  - Unique features include **parallel accelerator designs**, **modular HLS**, and **ASIC support**
  - Provides automated testing and verification of the generated designs



Bambu HLS

https://panda.dei.polimi.it

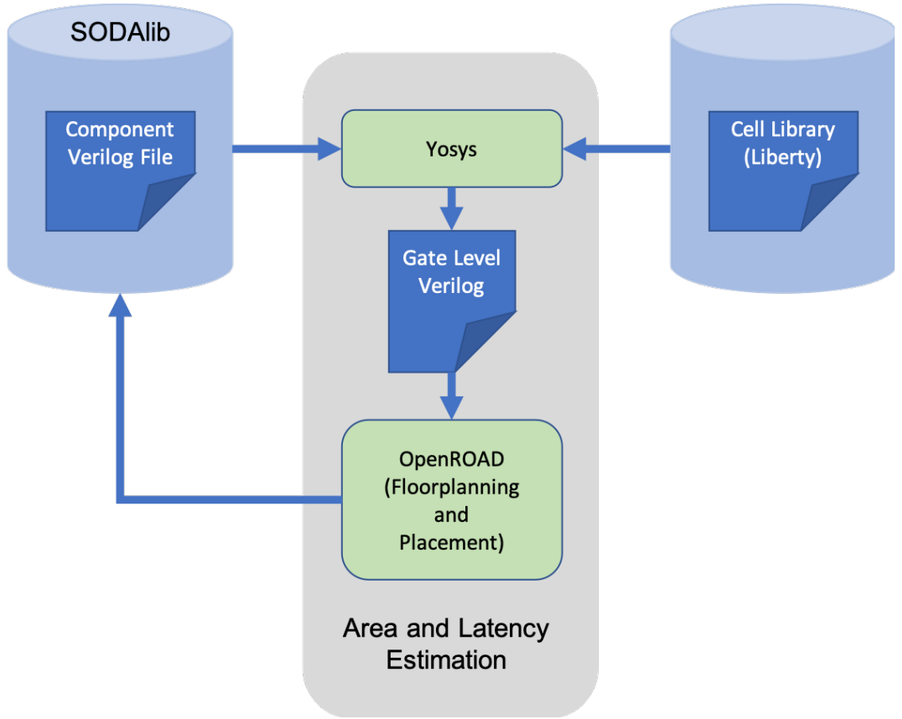
[Fabrizio Ferrandi, Vito Giovanni Castellana, Serena Curzel, Pietro Fezzardi, Michele Fiorito, Marco Lattuada, Marco Minutoli, Christian Pilato, Antonino Tumeo: Invited: Bambu: an Open-Source Research Framework for the High-Level Synthesis of Complex Applications. DAC 2021: 1327-1330]

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



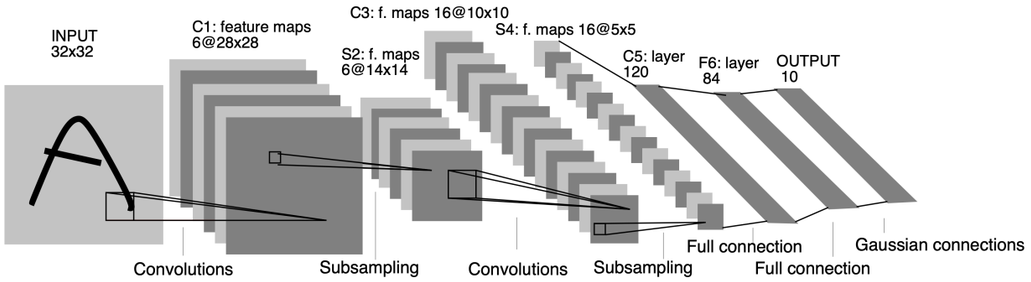
SODA characterization flow. The characterization flow can be extended to synthesize HLS generated designs, or used to estimate their area-latency-power profiles to drive the Design Space Exploration engine

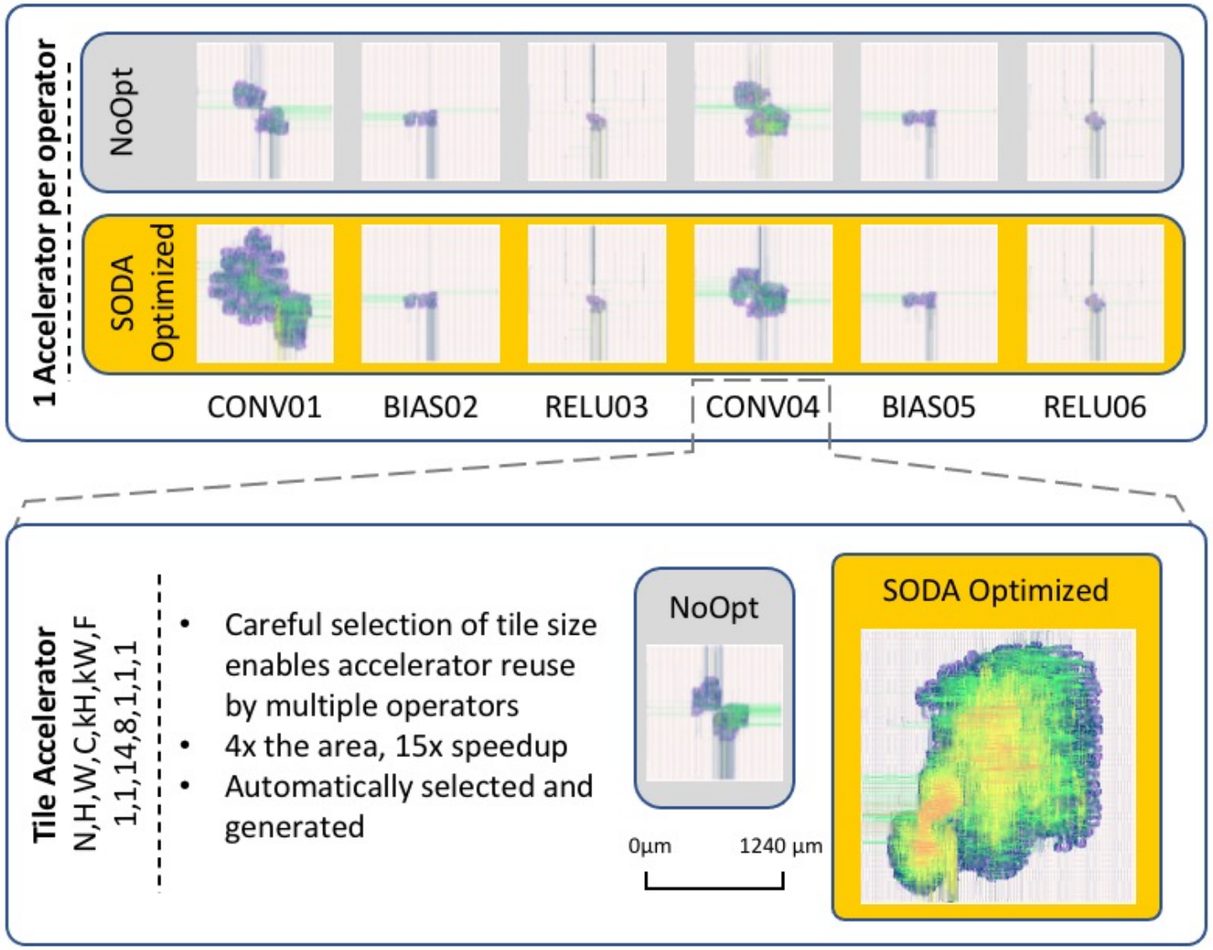
OpenROAD

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**

LeNet with OpenPDK 45 nm

Kernel	No Optimizations			With optimizations			Trade-offs	
	Cycles	Area (μm^2)	Power eff. (GFLOPS/W)	Cycles	Area (μm^2)	Power eff. (GFLOPS/W)	Speedup	Area Overhead
CONV_01	10,262,618	29,073	4.43	4,627,982	124,255	2.68	2.22	4.27
BIAS_02	251,694	10,395	11.48	40,826	60,048	9.01	6.17	5.78
RELU_03	151,342	7,385	41.55	38,446	35,695	38.39	3.94	4.38
CONV_04	85,380,948	36,814	3.32	83,380,180	37,556	3.34	1.02	1.02
BIAS_05	62,932	10,409	11.00	10,222	60,007	8.41	6.16	5.76
RELU_06	37,844	7,464	41.75	9,620	35,950	37.04	3.93	4.82

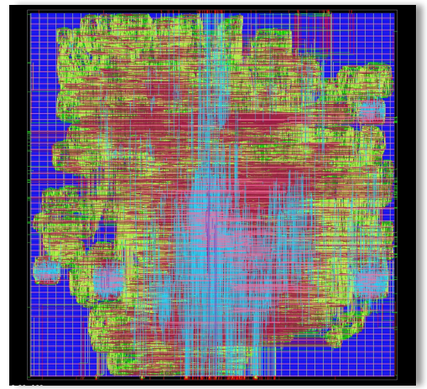
Table 2: Evaluation of non optimized and optimized LeNet operators in ASIC technology (FreePDK 45 nm at 500 MHz)

- SODA-OPT's optimizations provide a speedup proportional to the increase in the area
- Power efficiency may be slightly reduced due to an increase in power consumption of the faster solutions

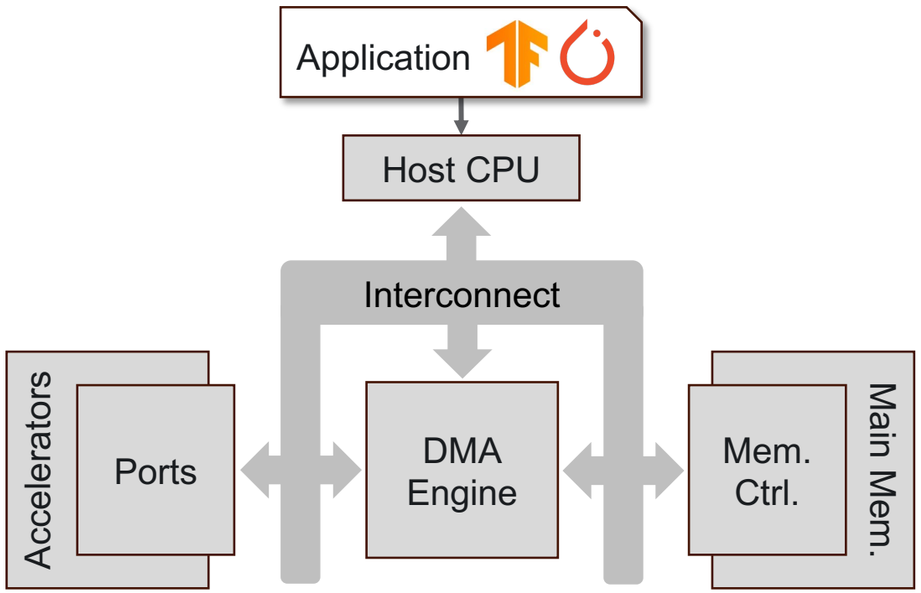
From generative AI model to ASIC with 30min of work and 2.5h of compilation time

Compilation Toolchain

LLaMa2.c Model



Synthesized for 200MHz



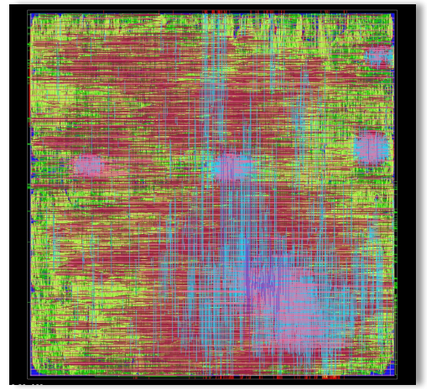

```
while (pos < steps) {

    // forward the transformer to get logits for the next token
    transformer(token, pos, &config, &state, &weights);

    // sample the next token
    if(temperature == 0.0f) {
        // greedy argmax sampling
        next = argmax(state.logits, config.vocab_size);
    } else {
        // apply the temperature to the logits
        for (int q=0; q<config.vocab_size; q++) { state.logits[q] /= temperature; }
        // apply softmax to the logits to get the probabilities for next token
        softmax(state.logits, config.vocab_size);
        // we now want to sample from this distribution to get the next token
        next = sample(state.logits, config.vocab_size);
    }
    printf("%s", vocab[next]);
    fflush(stdout);

    // advance forward
    token = next;
    pos++;
}
```

Selected relevant code



Synthesized for 500MHz

Adapted from: N. B. Agostini et al., "Bridging Python to Silicon: The SODA Toolchain," in IEEE Micro, vol. 42, no. 5, pp. 78-88, 1 Sept.-Oct. 2022

Pacific Northwest NATIONAL LABORATORY

PolyBench with ASAP 7nm

Kernel	Size	No optimizations				With optimizations				Trade-offs	
		Cycles	Max. freq. (MHz)	Area (μm^2)	Energy (nJ)	Cycles	Max. freq. (MHz)	Area (μm^2)	Energy (nJ)	Speedup	Area Overhead
atax	2	118	1164.28	1,658	1.63	38	986.65	2,539	1.13	3.11	1.53
	4	463	1384.20	1,970	8.20	63	1182.30	8,211	5.20	7.35	4.17
	8	1,819	1371.31	3,056	33.03	113	634.15	22,677	34.21	16.10	7.42
bicg	2	113	1778.91	1,605	1.26	24	1206.61	4,425	0.72	4.71	2.76
	4	458	1532.42	1,881	4.00	39	1098.86	12,015	3.87	11.74	6.39
	8	1,810	1314.11	2,974	34.98	78	400.86	30,744	21.01	23.21	10.34
gemm	2	161	1401.78	3,100	4.66	27	1411.24	5,067	1.09	5.96	1.63
	4	1,258	1286.68	4,597	37.74	51	782.38	21,089	16.36	24.67	4.59
	8	10,450	1451.15	2,590	143.30	139	–	–	–	75.18	–
gemver	2	246	1250.11	3,778	9.01	66	1374.54	5,832	2.78	3.73	1.54
	4	974	1031.52	7,835	68.27	91	652.62	18,595	43.50	10.70	2.37
	8	3,833	1121.75	7,547	292.83	141	–	–	–	27.18	–
gesummv	2	142	1554.66	2,336	2.25	35	1235.27	4,866	1.12	4.06	2.08
	4	514	1032.78	2,468	12.09	50	1128.52	10,431	5.10	10.28	4.23
	8	1,922	1279.50	3,614	44.16	101	638.13	25,487	52.07	19.03	7.05
mvt	2	114	1583.37	1,737	1.09	24	892.39	4,408	1.07	4.75	2.54
	4	450	1355.68	4,760	20.15	41	1125.35	13,366	3.59	10.98	2.81
	8	1,819	1216.38	3,544	60.56	81	559.19	32,757	16.08	22.46	9.24
three_mm	2	340	1155.26	3,577	9.42	42	1176.22	9,424	3.50	8.10	2.63
	4	2,719	1106.28	7,994	163.69	75	305.15	40,428	90.69	36.25	5.06
	8	22,130	1243.23	5,614	656.84	231	–	–	–	95.80	–
two_mm	2	274	1234.98	4,082	11.51	45	1337.67	6,915	0.78	6.09	1.69
	4	2,163	1019.49	7,063	128.57	75	422.71	34,326	20.94	28.84	4.86
	8	17,762	1281.06	4,251	508.85	–	–	–	–	–	–

Table 1: PolyBench results with OpenROAD and the ASAP 7nm technology library

- Representative of **algorithmic patterns** in scientific computing or high-level data science frameworks
- Tensors of different **dimensions**
- Target frequency: ***1 GHz***
- Significant **speedups**, increased **energy efficiency** and comparatively small **area overheads**
- 1 GHz constraint met by ***all the non-optimized cases***
- Maximum frequency for the optimized cases **reduced for larger tensor sizes**
- ASIC designs **not generated** for a few optimized cases with larger tensor sizes because of **routing congestion**

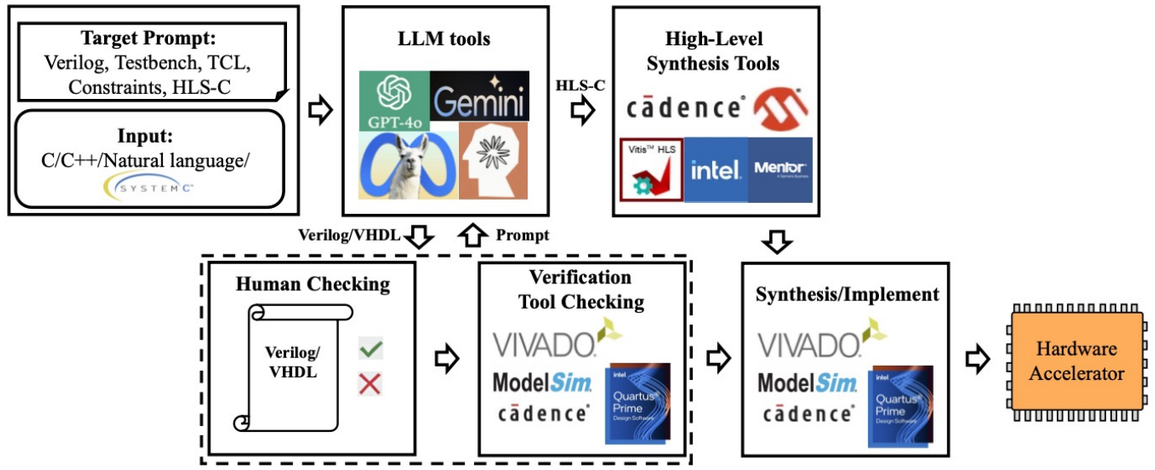
Pacific Northwest
NATIONAL LABORATORY

Applications of AI/ML to Hardware Design

- Top tier design automation conferences are now dominated by AI/ML
- Applied at all levels of the flow:
  - High-Level Synthesis (HLS)
  - Electronic Design Automation frontend
  - Electronic Design Automation backend
- Cadence and Synopsys have become ML tools companies
- Internal hardware design “foundation models” at major semiconductor companies
  - E.g., NVIDIA’s ChipNEMO
- Several startups leveraging generative AI for hardware design
  - E.g., PrimisAI

Applications of Large Language Models to HLS

- High-level Synthesis: the process of converting a specification in a high-level language (any high-level language!) to a design in a hardware design language
- Large Language Models as a way to generate Verilog?
  - People have been evaluating different approaches:
  - Specifications (in natural language or programming languages such as C) to Verilog
  - Specifications to C annotated for HLS tools (e.g., Vivado)



[Adapted from Y. Liao, T. Adegbija, R. Lysecky, “Are LLMs Any Good for High-Level Synthesis?” ICCAD 2024 Special Session]

Applications of Large Language Models to HLS

Table 1: The number of Prompts for LLM-based approaches

Benchmark	HLS-C	Verilog	TCL	Testbench	XDC
syrk	4	50	9	12	5
syr2k	1	20	3	7	3
mvt	1	36	3	7	2
k3mm	1	21	3	5	2
k2mm	1	29	3	6	3
gesummv	1	23	3	7	3
gemm	1	22	3	6	3
bicg	1	16	3	6	3
atax	1	11	3	8	3

Table 2: Place & routing results for *C→LLM→Verilog* and *C→HLS→Verilog* approaches

Benchmark	Approach	Execution cycles	FF	LUT	Slice	DSP	BRAM	Power (W)	CP
syrk	LLM	1859983	954	5300	1649	6	8	0.164	9.934
	HLS	3744260	662	521	221	5	32	0.350	8.191
syr2k	LLM	2125846	542	472	197	2	12	0.181	9.446
	HLS	9028229	1042	960	1649	5	56	0.383	6.872
mvt	LLM	44996	8628	2663	4404	2	4	0.197	9.312
	HLS	119492	713	991	342	5	12	0.332	6.55
k3mm	LLM	2371593	623	328	236	2	28	0.207	9.924
	HLS	10277509	927	956	1649	5	56	0.398	6.646
k2mm	LLM	1863816	537	311	202	2	20	0.189	9.967
	HLS	7963269	929	659	313	5	56	0.400	6.814
gesummv	LLM	65991	437	288	170	2	16	0.176	9.253
	HLS	148805	795	561	228	5	20	0.316	6.855
gemm	LLM	1601739	488	332	200	2	16	0.178	9.697
	HLS	4542980	807	505	238	5	32	0.359	6.551
bicg	LLM	46478	505	194	198	2	20	0.196	9.251
	HLS	119492	711	429	223	5	12	0.333	6.599
atax	LLM	57669	453	257	164	2	16	0.167	9.952
	HLS	119492	741	428	209	5	11	0.309	6.573

- Note that pragmas in HLS tools significantly influence performance and quality of results for the synthesis
- What we are probably seeing here is that ChatGPT-4o is not well trained on the necessary set of optimizations and coding rules needed for “HLS C” code – see Collini, Garg, Karri, “C2HLSC: Can LLMs Bridge the Software-to-Hardware Design Gap?”

[Adapted from Y. Liao, T. Adegbija, R. Lysecky, “Are LLMs Any Good for High-Level Synthesis?” ICCAD 2024 Special Session]

Pacific Northwest
NATIONAL LABORATORY

Extending HLS with AI/ML: Opportunities to Improve the HLS Process

- Applications of techniques to identify intermediate representation (IR) correlations and patterns
  - Some works using graph neural networks to automatically place pragmas
- Implementing co-pilots
  - Enable domain scientists to identify design constraints through conversational interface
- Design verification
  - Generation of testbenches
  - Debugging input HLS codes
- Improve design space exploration
  - Using open-source compiler toolchains (e.g., SODA-OPT/MLIR and Bambu), it is possible to train large language models on effects of compiler optimizations…

Pacific Northwest
NATIONAL LABORATORY

Extending HLS with AI/ML: Opportunities from the Integration With Logic Synthesis Tools

- Better prediction models for coarse grained metrics
  - Timing of critical paths, area
  - Better prediction of timing of critical paths provide synthesis flow with more opportunities to generate designs that meet timing closure
- Start embedding layout information at the high-level
  - Enable easier routing and reduce wiring congestion
- Provide methods that can adapt to different Process Development Kits (PDKs) and device technology

Pacific Northwest NATIONAL LABORATORY

Example of Intelligent System Design with bio-inspired Heuristics

- Artificial intelligence and machine learning are not necessarily only neural networks
- High-Level and System Synthesis need to solve several NP-complete problems
  - High-Level Synthesis: Resource Allocation, Scheduling, Resource Binding, Interconnection…
  - System Synthesis: HW/SW partitioning, Scheduling, Mapping, Communication orchestration…
- Brute-force methods and Integer Linear Programming fomulations require too much time
  - Problems also are strictly correlated, and executed in different orders
- The community has been using heuristics and approximations since a long time
- Bio-inspired heuristics, including genetic algorithms and swarm optimization are another form of intelligence
  - We should augment, rather than replace, these methods

Pacific Northwest
NATIONAL LABORATORY

Example: Genetic Algorithm for High-Level Synthesis

- Genetic Algorithms (GAs) enable exploring non-convex design spaces by evolving a population of solutions
  - mutation introduces local variations
  - crossover allows jumping across areas of the space and exit from local minima or maxima
  - selection of the fittest then guides the search along the most promising areas

- We apply GAs High-Level Synthesis process
  - Each chromosome represent a full synthesis process
  - By considering the full synthesis process, we can explore much larger design space than considering each synthesis “task” alone

NSGA-II for synthesis example

+1	+2	...	!=1	...	-8	-9	SchedA	RegA	ConnA
<plus;0>	<plus;1>	...	<cmp;0>	...	<minus;0>	<minus;0>	1	2	1

- Non-Dominated Sorting Genetic Algorithm II
- Chromosome encoding
  - Binding of Operations to Functional Units
  - Algorithms for Scheduling, Register Allocation, Interconnection
- Mutation and crossover
- Elitism preserves diversity into the population
  - Crowded-comparison operator based on density estimation, allows obtaining the crowding distance
- Selection: non-dominated rank and crowding distance
  - Solutions are ranked also inside a non-dominated level: if they have the same rank, they belong to the same front and selection prefers the. less crowded region

[C. Pilato, A. Tumeo, G. Palermo, F. Ferrandi, P. L. Lanzi, D. Sciuto: Improving evolutionary exploration to area-time optimization of FPGA designs. J. Syst. Archit. 54(11): 1046-1057 (2008)]

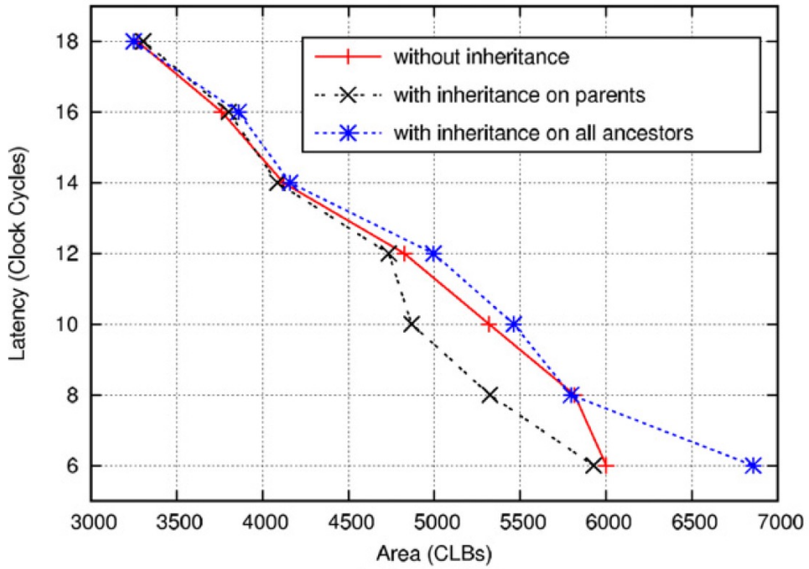
GA for HLS Results with JPEG

Clock-cycles breakdown of the execution of the JPEG compression algorithm on a single MicroBlaze architecture with and without the hardware modules synthesized with our evolutionary flow

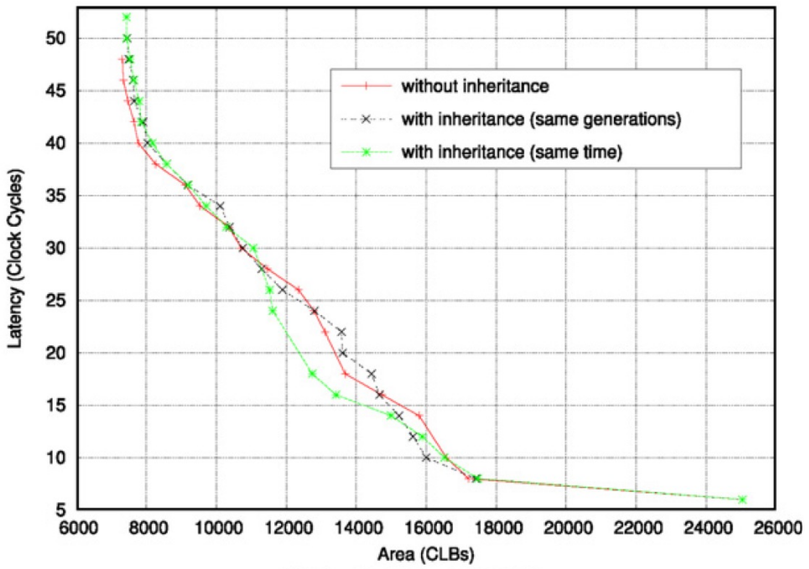
Phase	SW	SW/HW
File reading	169,512,465	171,672,577
RGB to YUV	1,598,251,574	1,643,508
Exp and downsample	678,638	679,306
Set quant. table	60,910	61,027
DCT and quant.	489,865,154	190,905,503
Entropic coding	599,810,406	592,318,457
Total	2,858,179,147	957,280,378

Comparison among number of fitness function (real) evaluations without inheritance and with inheritance on ancestors and on parents ($p_i = 0.557$)

Benchmark	Size	#Eval. standard	#Eval. ancestors	#Eval. parents
DCT	48	60.277	28.148	45.123
RGB2YUV	152	60.713	29.274	46.566



(a) *DCT*



(b) *RGB2YUV*

Ant Colony Optimization for Scheduling and Mapping

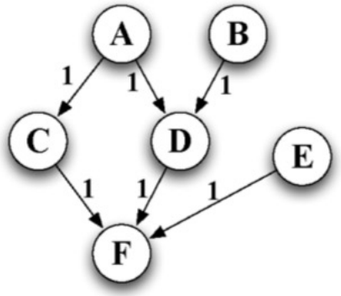
- Ant Colony Optimization: multi-agent optimization heuristic
  - Ants randomly explore different paths to the food.
- At each decision point:

$$p_{x,y} = \frac{[\tau_{x,y}]^{\alpha} * [\eta_{x,y}]^{\beta}}{\sum_{l \in \Omega_x} [\tau_{x,l}]^{\alpha} * [\eta_{x,l}]^{\beta}}$$

- They deposit pheromone proportionally to the length of the path, which suggests other ants to follow the same trail. Pheromone also evaporates with time.

$$\tau_{x,y} = (1 - \rho) * \tau_{x,y} + \epsilon$$

Scheduling and Mapping Example

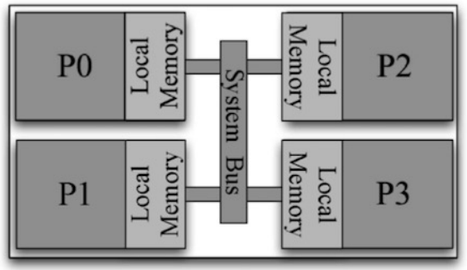


Tasks

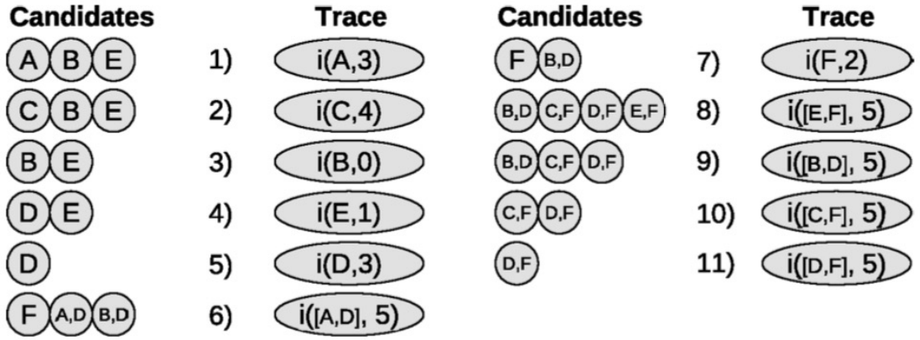
A B C D E F

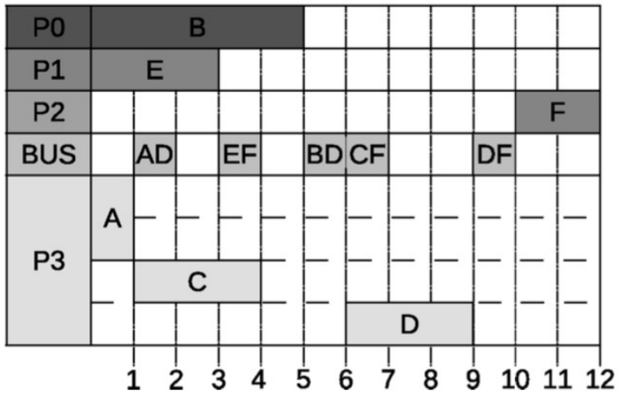
Communications

A,C A,D B,D C,F D,F E,F



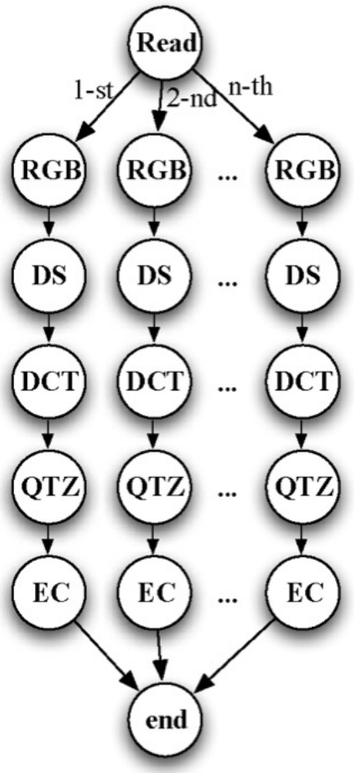
Task	$P0$		$P1$		$P2$		$P3$			
	$i_{t,0}$		$i_{t,1}$		$i_{t,2}$		$i_{t,3}$		$i_{t,4}$	
	Time	q_0	Time	q_1	Time	q_2	Time	q_3	Time	q_3
A	8	2	2	3	6	4	1	2	3	1
B	5	3	10	4	6	3	2	2	4	1
C	4	4	6	2	4	1	1	2	3	1
D	8	1	2	2	7	3	3	1	1	3
E	10	3	3	1	8	3	1	3	2	1
F	3	4	7	3	2	2	1	2	2	1





[F.Ferrandi, P.L. Lanzi, C. Pilato, D. Sciuto, A. Tumeo: Ant Colony Heuristic for Mapping and Scheduling Tasks and Communications on Heterogeneous Embedded Systems. IEEE Trans. on CAD of Integrated Circuits and Systems 29(6): 911-924 (2010)]

ACO Results with JPEG



JPEG compression task graph

Phase	PPC	MB	FPGA	
			Time	#Slices
Read	93 880 530	42 203 215	–	–
RGB	897 522	457 674	486 384	200
Downsampling (DS)	365 548	166 893	–	–
DCT	38 415 925	10 465 720	231 377	2760
Quantization (QTZ)	47 603 256	4 388 594	–	–
EC	109 677 075	38 008 544	–	–

Task characteristics

#Tasks/ #Edges	ACO		Platform		Diff.
	Avg.	%RSD	Avg.	%RSD	
20/20	170 416 083	2%	174 236 480	8%	2.19%
30/24	236 037 486	2%	268 569 432	7%	12.11%
40/32	324 599 846	1%	358 962 446	12%	9.57%
50/40	410 332 336	1%	427 313 801	7%	3.97%

Architecture contains one PPC, one MicroBlazes and 8 400 free slices.

Performance varying number of tasks

MB/Area	ACO		Platform		Diff.
	Avg.	%RSD	Avg.	%RSD	
3/2800	259 729 940	2%	248 492 658	15%	−4.52%
2/5600	301 159 209	7%	283 861 438	11%	−6.09%
1/8400	410 332 336	1%	427 313 801	7%	3.97%

Application is composed of 50 tasks.

Performance varying architecture characteristics

Pacific Northwest NATIONAL LABORATORY

Research Opportunities: AI/ML with the Open-Source Hardware-Design Ecosystem

- SODA provides initial integration for several open-source tools
  - Example of open interfaces between SODA-OPT and Bambu and Bambu and OpenROAD with the Open-Source PDKs
- Need to further strengthen this integration, providing application programming interfaces (APIs) to access data at various stages of the synthesis
  - Provides the necessary data to train AI/ML models
  - Provides opportunity to predict effects of optimizations from one layer to the other
  - Provides opportunities for co-optimization, that do not necessarily only use neural networks
- Only through open interfaces we can strengthen the research methods
  - Proprietary tools could also be part of the flows, but they need to provide ways to exchange data and information

Pacific Northwest
NATIONAL LABORATORY

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*

Pacific Northwest
NATIONAL LABORATORY

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
  - Considers system-level implications
  - Enables automated design space exploration and agile hardware design

- **The SODA Synthesizer provides a foundation to explore novel AI/ML methods applied to high-level synthesis to significantly improve the quality of results**
- **We argue that we should not replace HLS with generative AI techniques, but augment it with a variety of approach (also including generative AI)**

Pacific Northwest
NATIONAL LABORATORY

Thank you

This research has been partially supported by the SODALITE Project in the DARPA RTML program, the SO(DA)^2 project in the Data-Model Convergence LDRD Initiative, the SODA ULTRA project in the AT SCALE LDRD Initiative, the DOE ASCR project Compilers Frameworks and Hardware Generators to Support Innovative US Government Designs, the DOE SC project Advanced Memory to Support Artificial Intelligence for Science (AMAIS), and Project 38 at Pacific Northwest National Laboratory (PNNL)